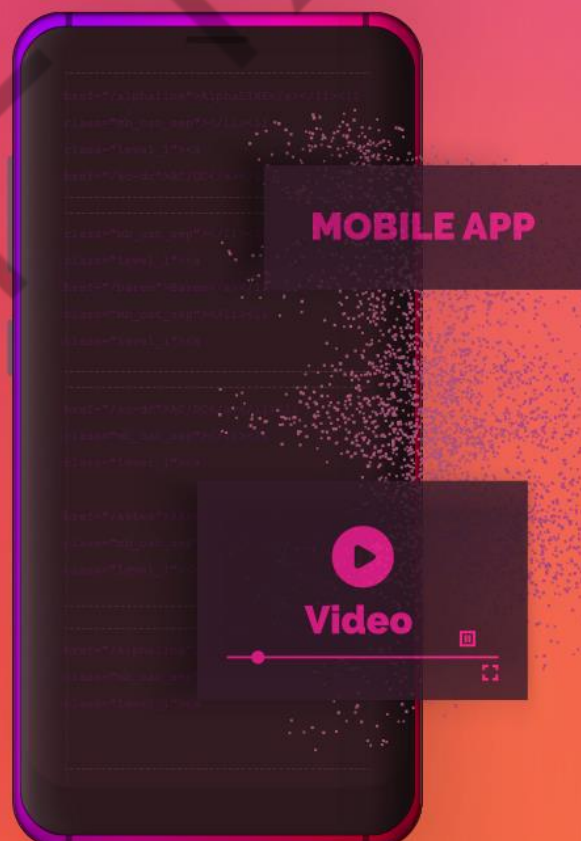


SERVICES ARCHITECTURE, API E MOBILE ARCHITECTURE IMPLEMENTANDO UM APLICATIVO **HÍBRIDO COM IONIC - PARTE I**

DIEGO GARCIA E CHRISTIANE DE PAULA REIS



LISTA DE FIGURAS

Figura 6.1 – Selecionando um projeto <i>blank</i>	6
Figura 6.2 – Integração com o Cordova	6
Figura 6.3 – Recusando a integração com SDK do Ionic e geração do projeto .	6
Figura 6.4 – Estrutura do projeto	7
Figura 6.5 – Rodando o comando ionic serve	8
Figura 6.6 – Página Blank rodando no navegador	8
Figura 6.7 – Gerando uma Page sem módulo.....	9
Figura 6.8 – Importando DetailPage.....	10
Figura 6.9 – HTML DetailPage	10
Figura 6.10 – Trocando a Page inicial para DetailPage	12
Figura 6.11 – Tela gerada pelo código html implementado.....	13
Figura 6.12 – Tela gerada pelo código.....	17
Figura 6.13 – Trocando novamente a HomePage, como Page inicial.....	18
Figura 6.14 – Criando o serviço Games	18
Figura 6.15 – Tela implementada pelo código.....	21

LISTA DE CÓDIGOS-FONTE

Código-fonte 6.1 – Instalando o Ionic e o Cordova pelo npm.....	5
Código-fonte 6.2 – Gerando um projeto pelo <i>client</i> do Ionic.....	5
Código-fonte 6.3 – Rodar a aplicação no servidor do <i>client</i> do Ionic.....	8
Código-fonte 6.4 – Gerando uma Page.....	9
Código-fonte 6.5 – Código detail.html	12
Código-fonte 6.6 – Instalando o ngx-qrcode-2 no projeto.....	13
Código-fonte 6.7 – Instalando plugins Cordova de leitura do QR Code	13
Código-fonte 6.8 – Importando componentes no app.module.ts	14
Código-fonte 6.9 – Implementando detail.ts	15
Código-fonte 6.10 – Implementando detail.html	16
Código-fonte 6.11 – Criando o serviço Games.....	18
Código-fonte 6.12 – Implementando o serviço GamesProvider.ts	19
Código-fonte 6.13 – Implementando o home.html.....	20
Código-fonte 6.14 – Implementando o home.ts.....	20
Código-fonte 6.15 – Refatorando o detail.ts	23
Código-fonte 6.16 – Refatorando o detail.html	25

SUMÁRIO

6 IMPLEMENTANDO UM APLICATIVO HÍBRIDO COM IONIC	5
6.1 Introdução	5
6.2 Gerando o projeto pelo Ionic <i>client</i>	5
6.3 Pages	7
6.3.1 Criando uma Page.....	9
6.4 Implementando a <i>DetailPage</i>	10
6.5 Implementando um componente externo	13
6.6 <i>NavController</i>	17
6.7 <i>NavParams</i>	21
CONCLUSÃO.....	26
REFERÊNCIAS.....	27

6 IMPLEMENTANDO UM APLICATIVO HÍBRIDO COM IONIC

6.1 Introdução

Agora que você já viu todos os conceitos que envolvem o desenvolvimento de aplicativos mobile, neste capítulo, vamos implementar nosso primeiro projeto. Para fixar a parte teórica que foi apresentada, a ideia aqui é criar um simples aplicativo gerador e leitor de QR Code especificamente para *games*.

Neste projeto, vamos utilizar o *framework* Ionic acessando o Cordova.

Para desenvolver e editar seus códigos-fontes, você deve ter instalado em sua máquina algum editor de texto. Escolhemos o Visual Studio Code para Windows.

Link para download: Visual Studio Code – <<https://code.visualstudio.com/download>>.

6.2 Gerando o projeto pelo Ionic *client*

Assim como o Angular, o Ionic possui seu próprio *client*, que será responsável por gerar nosso projeto. Vamos aproveitar e instalar o Cordova junto, sendo que a ferramenta de linha de comando Cordova é executada no Node.js e distribuída no NPM (*Node Package Manager*).

Para realizar a instalação, basta abrir o seu terminal e utilizar o comando.

```
npm install -g ionic cordova
```

Código-fonte 6.1 – Instalando o Ionic e o Cordova pelo npm
Fonte: Elaborado pelo autor (2018)

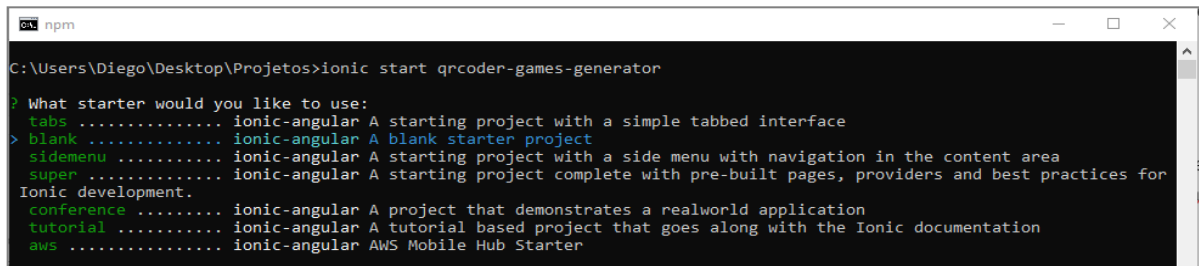
Com o *client* instalado globalmente pelo Node, vamos iniciar nosso projeto com o seguinte comando.

```
Ionic start qrcode-games-generator
```

Código-fonte 6.2 – Gerando um projeto pelo *client* do Ionic
Fonte: Elaborado pelo autor (2018)

Logo após, o *client* nos fornecerá algumas opções para gerar nosso projeto. Podemos gerar projetos vazios, com tabs, com menu lateral e até mesmo projetos

completos de exemplo, como o *super*, porém, vamos começar em um aplicativo do zero. Selecione a opção *blank*.

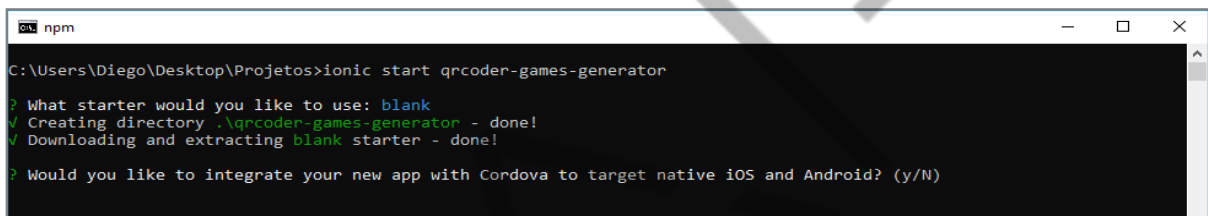


```
C:\Users\Diego\Desktop\Projetos>ionic start qrcoder-games-generator

? What starter would you like to use:
  tabs ..... ionic-angular A starting project with a simple tabbed interface
> blank ..... ionic-angular A blank starter project
  sidemenu ..... ionic-angular A starting project with a side menu with navigation in the content area
  super ..... ionic-angular A starting project complete with pre-built pages, providers and best practices for
Ionic development.
  conference ..... ionic-angular A project that demonstrates a realworld application
  tutorial ..... ionic-angular A tutorial based project that goes along with the Ionic documentation
  aws ..... ionic-angular AWS Mobile Hub Starter
```

Figura 6.1 – Selecionando um projeto *blank*
Fonte: Elaborado pelo autor (2018)

Após selecionar a opção *blank*, o *client* nos solicitará a integração do Ionic com o Cordova, vamos aceitar digitando *y* e apertando Enter.



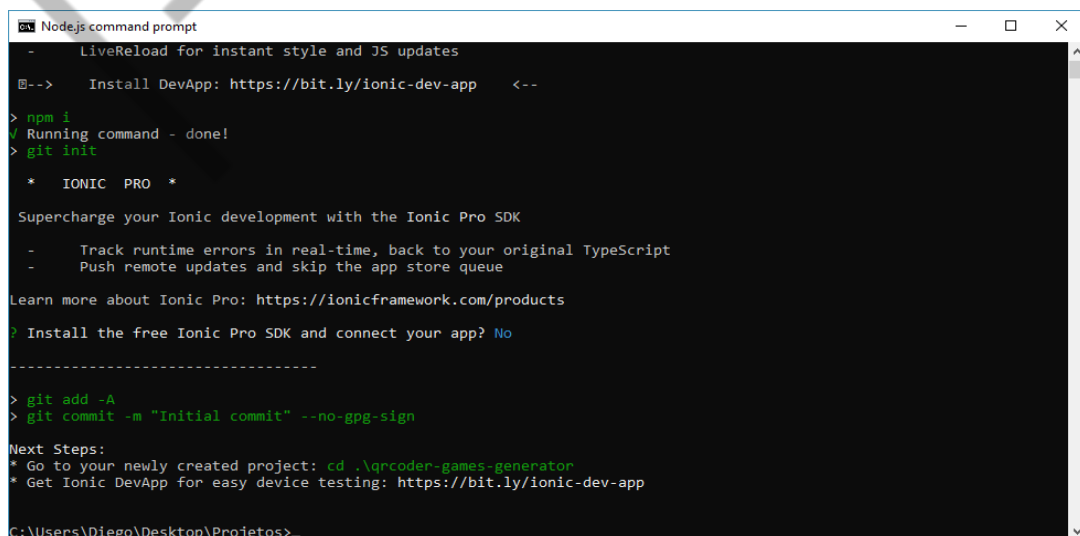
```
C:\Users\Diego\Desktop\Projetos>ionic start qrcoder-games-generator

? What starter would you like to use: blank
✓ Creating directory .\qrcoder-games-generator - done!
✓ Downloading and extracting blank starter - done!

? Would you like to integrate your new app with Cordova to target native iOS and Android? (y/N)
```

Figura 6.2 – Integração com o Cordova
Fonte: Elaborado pelo autor (2018)

Feito isso, nosso *client* realizará a instalação do nosso projeto e perguntará se desejamos conectá-lo no SDK (*Software Development Kit*) do Ionic, que permite realizar algumas funcionalidades, como Push Notification, *deploy* em nuvem, geração da Splash Screen (tela de abertura no Android) e ícones etc. Nesse projeto, não vamos utilizar esses serviços, então, vamos digitar *N* e apertar <Enter>.



```
Node.js command prompt

- LiveReload for instant style and JS updates
@--> Install DevApp: https://bit.ly/ionic-dev-app <--

> npm i
✓ Running command - done!
> git init

* IONIC PRO *

Supercharge your Ionic development with the Ionic Pro SDK

- Track runtime errors in real-time, back to your original TypeScript
- Push remote updates and skip the app store queue

Learn more about Ionic Pro: https://ionicframework.com/products

? Install the free Ionic Pro SDK and connect your app? No

-----

> git add -A
> git commit -m "Initial commit" --no-gpg-sign

Next Steps:
* Go to your newly created project: cd .\qrcoder-games-generator
* Get Ionic DevApp for easy device testing: https://bit.ly/ionic-dev-app

C:\Users\Diego\Desktop\Projetos>
```

Figura 6.3 – Recusando a integração com SDK do Ionic e geração do projeto
Fonte: Elaborado pelo autor (2018)

Depois de realizar a instalação, abra seu projeto no Visual Studio Code. No Ionic, temos algumas pequenas mudanças na estrutura. A primeira grande mudança são os componentes gerados e chamados Page.

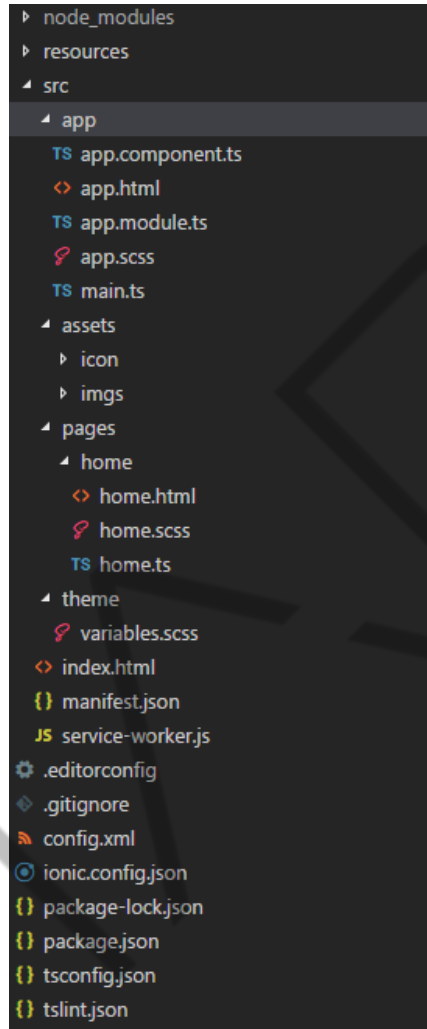


Figura 6.4 – Estrutura do projeto
Fonte: Elaborado pelo autor Diego (2018)

6.3 Pages

Page nada mais é que um componente responsável por uma tela do seu aplicativo. Toda Page possui seu próprio ciclo de vida e é controlada automaticamente pelo sistema de navegação de pilhas do Ionic, chamado **Navigation Controller**.

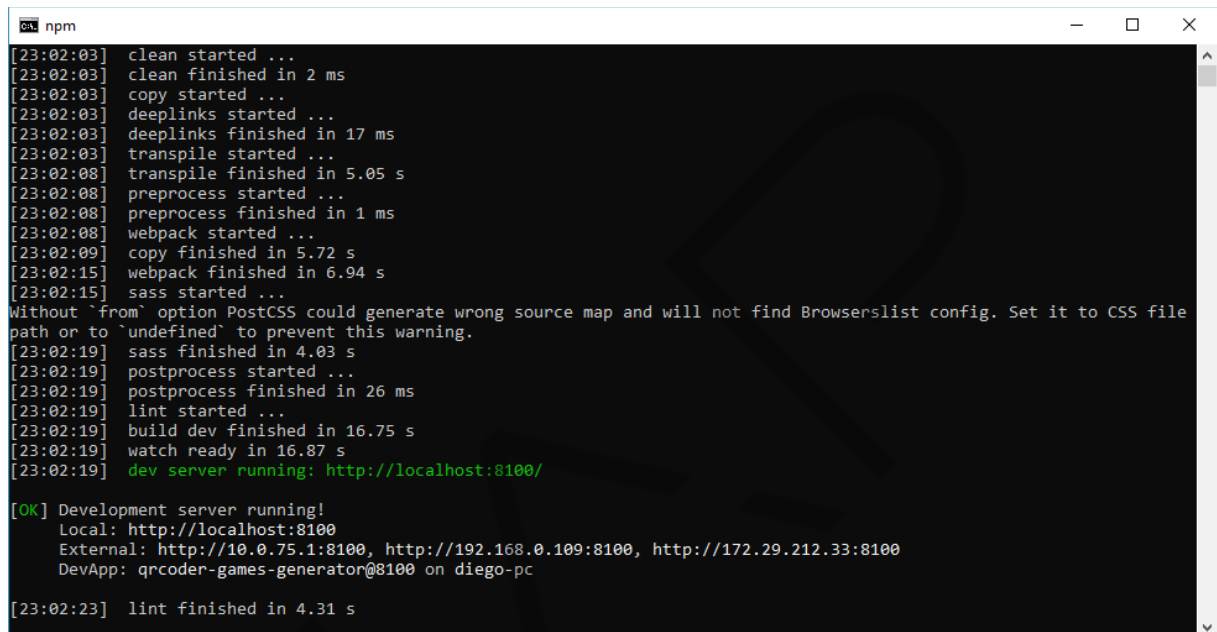
Você pode saber tudo sobre as principais funcionalidades desse serviço aqui: [<https://ionicframework.com/docs/api/navigation/NavController/>](https://ionicframework.com/docs/api/navigation/NavController/).

Vamos ver a HomePage que o Ionic criou automaticamente. Para isso, vamos rodar nossa aplicação no servidor do *client* do Ionic, digitando apenas **ionic serve** dentro da pasta do seu projeto.

```
ionic serve
```

Código-fonte 6.3 – Rodar a aplicação no servidor do *client* do Ionic

Fonte: Elaborado pelo autor (2018)



```
npm
[23:02:03] clean started ...
[23:02:03] clean finished in 2 ms
[23:02:03] copy started ...
[23:02:03] deeplinks started ...
[23:02:03] deeplinks finished in 17 ms
[23:02:03] transpile started ...
[23:02:08] transpile finished in 5.05 s
[23:02:08] preprocess started ...
[23:02:08] preprocess finished in 1 ms
[23:02:08] webpack started ...
[23:02:09] copy finished in 5.72 s
[23:02:15] webpack finished in 6.94 s
[23:02:15] sass started ...
Without `from` option PostCSS could generate wrong source map and will not find Browserslist config. Set it to CSS file path or to `undefined` to prevent this warning.
[23:02:19] sass finished in 4.03 s
[23:02:19] postprocess started ...
[23:02:19] postprocess finished in 26 ms
[23:02:19] lint started ...
[23:02:19] build dev finished in 16.75 s
[23:02:19] watch ready in 16.87 s
[23:02:19] dev server running: http://localhost:8100/

[OK] Development server running!
Local: http://localhost:8100
External: http://10.0.75.1:8100, http://192.168.0.109:8100, http://172.29.212.33:8100
DevApp: qr-coder-games-generator@8100 on diego-pc

[23:02:23] lint finished in 4.31 s
```

Figura 6.5 – Rodando o comando ionic serve

Fonte: Elaborado pelo autor (2018)

O comando vai, automaticamente, abrir uma página em nosso navegador, rodando nossa aplicação, com a Page principal criada de maneira automática pelo Ionic.

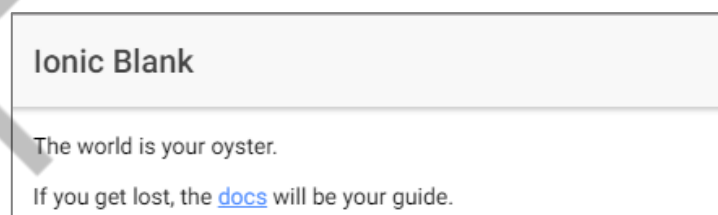


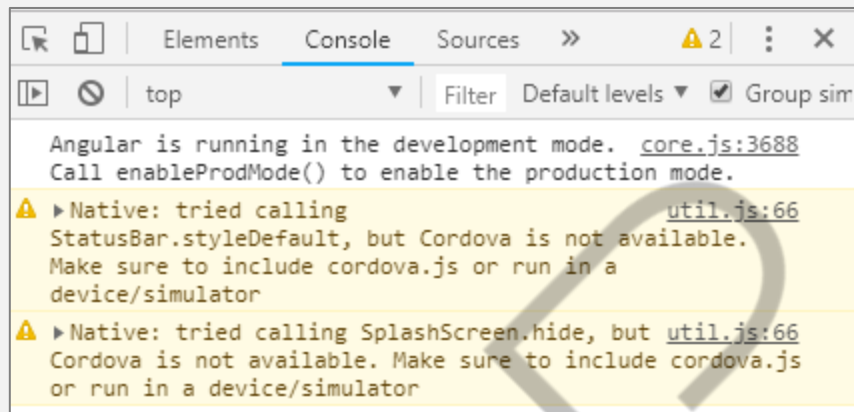
Figura 6.6 – Página Blank rodando no navegador

Fonte: Elaborado pelo autor (2018)

DICA:

Utilizar o navegador de modo comum é muito complicado, já que nossas aplicações serão desenvolvidas para dispositivos mobile, com uma tela bem menor.

Para isso, se você estiver utilizando o Google Chrome, basta abrir o inspetor de elementos (F12 e clicar no ícone do celular).



Seu navegador simulará um dispositivo mobile e será muito mais fácil para desenvolver a aplicação.

6.3.1 Criando uma Page

Nossa aplicação será bem simples, possuindo apenas duas Pages (uma para listar nossos jogos e outra para registrar e editar). Vamos criar uma Page chamada Detail, com o seguinte comando:

```
ionic g page Detail --no-module
```

Código-fonte 6.4 – Gerando uma Page

Fonte: Elaborado pelo autor (2018)

Como você pode ver, o *client* do Ionic também possui vários geradores, nesse caso, estamos gerando uma Page sem dependência de módulos.

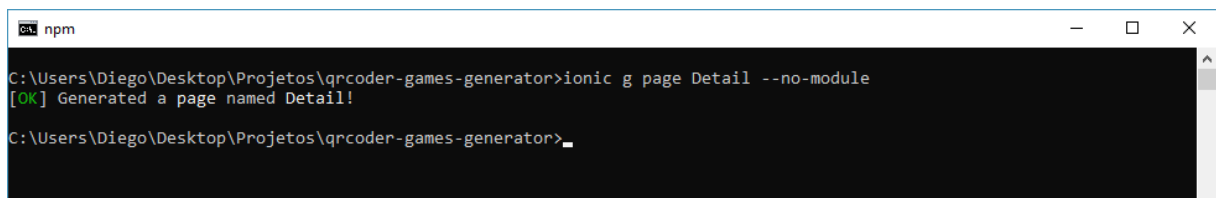


Figura 6.7 – Gerando uma Page sem módulo

Fonte: Elaborado pelo autor (2018)

O Ionic criará nossa Page, porém, diferentemente do Angular CLI, ele não importará de maneira automática nosso componente no **app.module.ts**, teremos que adicioná-lo manualmente na propriedade **declarations** e **entryComponents**.

```
@NgModule({
  declarations: [
    MyApp,
    HomePage,
    DetailPage
  ],
  imports: [
    BrowserModule,
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    DetailPage,
  ],
  providers: [
    StatusBar,
    SplashScreen,
    {provide: ErrorHandler, useClass: IonicErrorHandler}
  ]
})
export class AppModule {}
```

Figura 6.8 – Importando DetailPage
Fonte: Elaborado pelo autor (2018)

6.4 Implementando a DetailPage

Abra o arquivo localizado em **pages/detail/detail.html** gerado pelo *client*.

```
<!--
  Generated template for the DetailPage page.
  See http://ionicframework.com/docs/components/#navigation for more info on
  Ionic pages and navigation.
-->
<ion-header>
  <ion-navbar>
    <ion-title>Detail</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
</ion-content>
```

Figura 6.9 – HTML DetailPage
Fonte: Elaborado pelo autor (2018)

Essa é a estrutura básica de toda Page. Ela é composta pelo **ion-header**, que será responsável pelo header de nossa Page junto com o **ion-navbar** e o **ion-title**, no qual podemos mudar o título da NavBar de nossa Page. Em seguida, temos o **ion-content**, dentro dele, criaremos toda a estrutura html, pois esse componente já é responsável por gerenciar o *scroll* e a flexibilidade do *layout*.

Vamos implementar o seguinte código html:

```
<ion-header>

  <ion-navbar>
    <ion-title>Detalhes do jogo</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <ion-list>
    <ion-list-header>Detalhes</ion-list-header>

    <ion-item>
      <ion-label color="primary" stacked>Nome do jogo</ion-label>
      <ion-input type="text"></ion-input>
    </ion-item>

    <ion-item>
      <ion-label>Nintendo Switch</ion-label>
      <ion-checkbox></ion-checkbox>
    </ion-item>

    <ion-item>
      <ion-label>PC</ion-label>
      <ion-checkbox></ion-checkbox>
    </ion-item>

    <ion-item>
      <ion-label>Playstation 4</ion-label>
      <ion-checkbox></ion-checkbox>
    </ion-item>

    <ion-item>
      <ion-label>Xbox One</ion-label>
      <ion-checkbox></ion-checkbox>
    </ion-item>

    <ion-list-header>QRCode</ion-list-header>
    <button ion-button full>Gerar QRCode</button>
```

```
</ion-list>
</ion-content>
```

Código-fonte 6.5 – Código detail.html
Fonte: Elaborado pelo autor (2018)

Aqui, já utilizamos alguns componentes do Ionic. O `ion-list`, o `ion-list-header` e o `ion-item` criarão todo o estilo de lista de itens. O `ion-label`, o `ion-input` e o `ion-checkbox` serão responsáveis por criar nossos campos *input* e *checkbox*. O *button* possui a diretiva `ion-button` a fim de criar o estilo para o dispositivo.

Como você deve ter percebido, a Page inicial é a `HomePage`, vamos trocar rapidamente para `DetailPage`, apenas para visualizar o *layout* que implementamos. Abra o `app/app.component.ts` e implemente o `DetailPage` no atributo `rootPage` (não se esqueça de implementar a classe.).

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

import { HomePage } from '../pages/home/home';
import { DetailPage } from '../pages/detail/detail';

@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage:any = DetailPage;

  constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {
    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might need.
      statusBar.styleDefault();
      splashScreen.hide();
    });
  }
}
```

Figura 6.10 – Trocando a Page inicial para `DetailPage`
Fonte: Elaborado pelo autor (2018)

Assim, se o seu servidor ainda estiver aberto, dando apenas um `save` no código, você já verá todo o nosso código implementado.

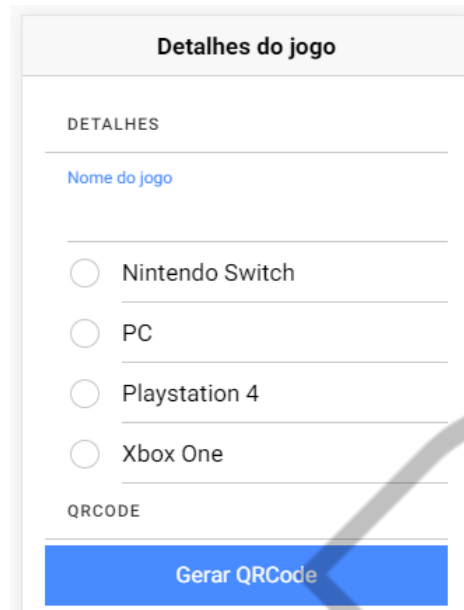


Figura 6.11 – Tela gerada pelo código html implementado
Fonte: Elaborado pelo autor Diego (2018)

6.5 Implementando um componente externo

Para gerarmos nossos QR Codes, vamos usar um componente externo **chamado ngx-qrcode-6**. No terminal do seu projeto, utilize o seguinte comando node.

```
npm install ngx-qrcode-2
```

Código-fonte 6.6 – Instalando o ngx-qrcode-2 no projeto

Fonte: Elaborado pelo autor (2018)

Já vamos adicionar o plugin Cordova, que será responsável por ler esses QR Codes:

```
ionic cordova plugin add phonegap-plugin-barcodescanner  
npm install --save @ionic-native/barcode-scanner
```

Código-fonte 6.7 – Instalando plugins Cordova de leitura do QR Code

Fonte: Elaborado pelo autor (2018)

Após instalar esse pacote no seu projeto, só precisamos importar em nosso **app.module.ts**:

```
import { StatusBar } from '@ionic-native/status-bar';

import { MyApp } from '../app.component';
import { HomePage } from '../pages/home/home';
import { DetailPage } from '../pages/detail/detail';

// Componentes externos
import { NgxQRCodeModule } from 'ngx-qrcode2';
import { BarcodeScanner } from '@ionic-native/barcode-scanner';

@NgModule({
  declarations: [
    MyApp,
    HomePage,
    DetailPage
  ],
  imports: [
    BrowserModule,
    FormsModule,
    IonicModule.forRoot(MyApp),
    NgxQRCodeModule
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    DetailPage,
  ],
  providers: [
    StatusBar,
    SplashScreen,
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    BarcodeScanner
  ]
})
export class AppModule {}
```

Código-fonte 6.8 – Importando componentes no app.module.ts

Fonte: Elaborado pelo autor (2018)

No código, também importamos o **FormModule** do Angular, que será responsável por controlar os dados de input de nossos campos.

Vamos agora implementar nosso código no **detail/detail.ts**:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';

@Component({
  selector: 'page-detail',
  templateUrl: 'detail.html',
})
export class DetailPage {
  name: string = "";
  platform: object[] = [];
  qrCode: string = null;

  constructor(public navCtrl: NavController, public navParams:
NavParams) {}

  createCode() {
    const qrData = { name: this.name, ...this.platform };
    this.qrCode = JSON.stringify(qrData);
  }

  updatePlatform(checked, name) {
    if(checked) {
      this.platform.push(name);
    } else {
      this.platform.splice(this.platform.indexOf(name), 1);
    }
  }
}
```

Código-fonte 6.9 – Implementando detail.ts

Fonte: Elaborado pelo autor Diego (2018)

Para gerar um QR Code, precisamos transformar todos os nossos dados do formulário em *String*. Para realizar isso, criamos os atributos **name**, **platform** e **qrCode**. Criamos também o método **createCode()**, que será responsável por transformar todos os objetos em *String* e armazenar no atributo QR Code; por fim, o método **updateMode()** realiza a atualização do nosso Array de plataformas.

```
<ion-header>

  <ion-navbar>
    <ion-title>Detalhes do jogo</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <ion-list>
    <ion-list-header>Detalhes</ion-list-header>

    <ion-item>
```

```
<ion-label color="primary" stacked>Nome do jogo</ion-label>
<ion-input [(ngModel)]="name" type="text"></ion-input>
</ion-item>

<ion-item>
  <ion-label>Nintendo Switch</ion-label>
  <ion-checkbox #switch
    (ionChange)="updatePlatform(switch.checked, 'Nintendo Switch')"></ion-checkbox>
</ion-item>

<ion-item>
  <ion-label>PC</ion-label>
  <ion-checkbox #pc
    (ionChange)="updatePlatform(pc.checked, 'PC')"></ion-checkbox>
</ion-item>

<ion-item>
  <ion-label>Playstation 4</ion-label>
  <ion-checkbox #ps4
    (ionChange)="updatePlatform(ps4.checked, 'Playstation 4')"></ion-checkbox>
</ion-item>

<ion-item>
  <ion-label>Xbox One</ion-label>
  <ion-checkbox #xbox
    (ionChange)="updatePlatform(xbox.checked, 'Xbox One')"></ion-checkbox>
</ion-item>

<ion-list-header>QrCode</ion-list-header>
</ion-list>

<ion-card *ngIf="qrCode">
  <ngx-qr-code [qrc-value]="qrCode"></ngx-qr-code>
</ion-card>

<button ion-button full (click)="createCode()">Gerar QRCode</button>
</ion-content>
```

Código-fonte 6.10 – Implementando detail.html

Fonte: Elaborado pelo autor (2018)

Ao implementar esses códigos, você já terá um gerador de QR Code simples:

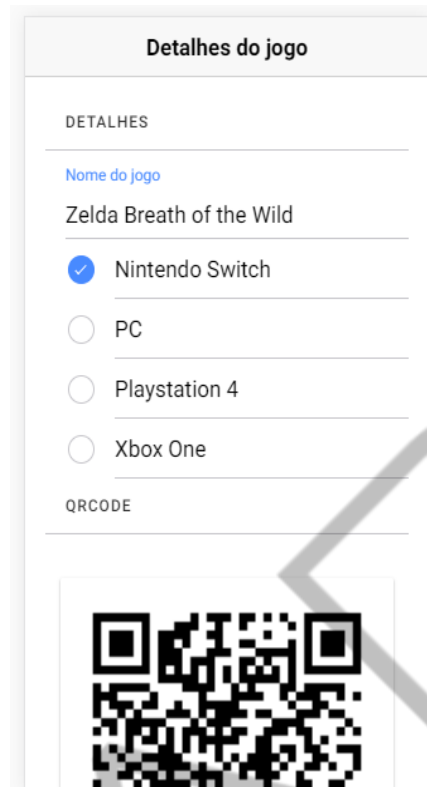


Figura 6.12 – Tela gerada pelo código
Fonte: Elaborado pelo autor (2018)

6.6 NavController

Agora, vamos transferir esses dados para uma lista, de uma forma bem simples, por meio do NavController. Você deve ter percebido que toda Page criada pelo *client* é gerada com a instância de **NavController** e **NavParams**. Essas classes são responsáveis por controlar o fluxo de navegação entre uma Page e outra e passar parâmetros entre elas. Criaremos nossa lista, que representará todos os QR Codes que cadastramos no HomePage, mas, antes, voltaremos à **app/app.component.ts** e deixaremos novamente o HomePage como a Page inicial de nossa aplicação.

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

import { HomePage } from '../pages/home/home';
import { DetailPage } from '../pages/detail/detail';

@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage:any = HomePage;

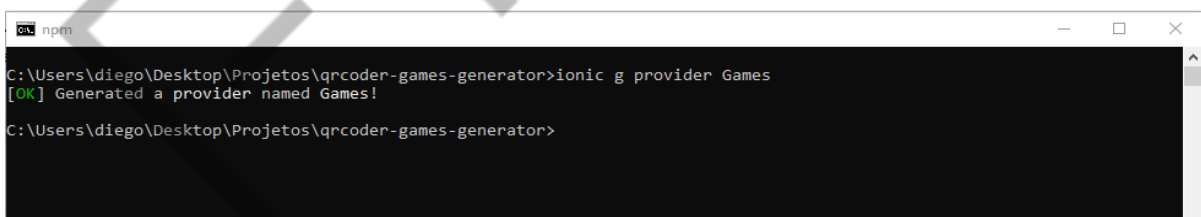
  constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {
    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might need.
      statusBar.styleDefault();
      splashScreen.hide();
    });
  }
}
```

Figura 6.13 – Trocando novamente a HomePage, como Page inicial
Fonte: Elaborado pelo autor (2018)

Para realizar uma pequena persistência de dados, durante a execução de nossa aplicação, vamos criar um serviço, pois ela é uma instância *Singleton*, ou seja, o seu estado de dados é o mesmo durante a execução do aplicativo. Para isso, basta utilizar o seguinte comando no terminal:

```
ionic g provider Games
```

Código-fonte 6.11 – Criando o serviço Games
Fonte: Elaborado pelo autor (2018)



```
C:\Users\diego\Desktop\Projetos\qrcoder-games-generator>ionic g provider Games
[OK] Generated a provider named Games!
C:\Users\diego\Desktop\Projetos\qrcoder-games-generator>
```

Figura 6.14 – Criando o serviço Games
Fonte: Elaborado pelo autor (2018)

Será gerada uma pasta chamada providers, com a pasta e a classe gerada pelo *client*, já importada em nosso módulo. Um *provider* é a mesma coisa que um Serviço. Agora, abra-o e implemente o seguinte código:

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

@Injectable()
export class GamesProvider {

  games = [{
    name: 'Zelda Breath of the Wild',
    platform: ['Nintendo Switch']
  },
  {
    name: 'Fortnite',
    platform: ['Nintendo Switch', 'PC', 'Xbox One']
  }
  ];

  constructor() {}

  insert(game) {
    this.games.push(game);
  }

  update(game, index) {
    this.games[index] = game;
  }
}
```

Código-fonte 6.12 – Implementando o serviço GamesProvider.ts
Fonte: Elaborado pelo autor (2018)

Abra o **home/home.html** e implemente o seguinte código:

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Lista de jogos
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content>
  <ion-list>
    <ion-item *ngFor="let game of games;let i = index"
      (click)="openDetail(game, i)">
      <ion-thumbnail item-start>
        <ngx-qr-code [qrc-
value]="convertObjectToString(game)"></ngx-qr-code>
      </ion-thumbnail>
      <h2>{{ game.name }}</h2>
      <p>{{ convertPlatformToString(game.platform) }}</p>
    </ion-item>
  </ion-list>
```

```
<ion-fab bottom right (click)="openDetail()">
  <button ion-fab mini><ion-icon name="add"></ion-
icon></button>
</ion-fab>
</ion-content>
```

Código-fonte 6.13 – Implementando o home.html
Fonte: Elaborado pelo autor (2018)

No nosso **home/home.ts** será implementado o seguinte o código:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';
import { DetailPage } from '../detail/detail';

import { GamesProvider } from '../../providers/games/games';

@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {

  games: object[] = []

  constructor(
    public gamesProvider: GamesProvider,
    public navCtrl: NavController,
    public NavParams: NavParams) {

    this.games = this.gamesProvider.games
  }

  convertObjectToString(obj) {
    return JSON.stringify(obj)
  }

  convertPlatformToString(platform) {
    return platform.join(' - ')
  }

  openDetail(game, index) {
    this.navCtrl.push(DetailPage, { game, index })
  }
}
```

Código-fonte 6.14 – Implementando o home.ts
Fonte: Elaborado pelo autor Diego (2018)

Aqui, além de criarmos uma pequena lista de jogos, elaboramos métodos para converter nossos dados em String e mostrar na tela. No método **openDetail()**, utilizamos o serviço NavController para enviar um game para nossa DetailPage e adicionar essa Page na pilha da navegação, abrindo-a. O método **push** “empurra” nossa Page na pilha, ou seja, é realizada uma animação quando aparece a tela; ela possui um botão de voltar em seu header.

O **flat button** abrirá a mesma DetailPage, só que sem enviar nenhum dado, para podermos criar um novo game em nossa lista.

Assim, com esses códigos implementados, você terá o seguinte resultado:

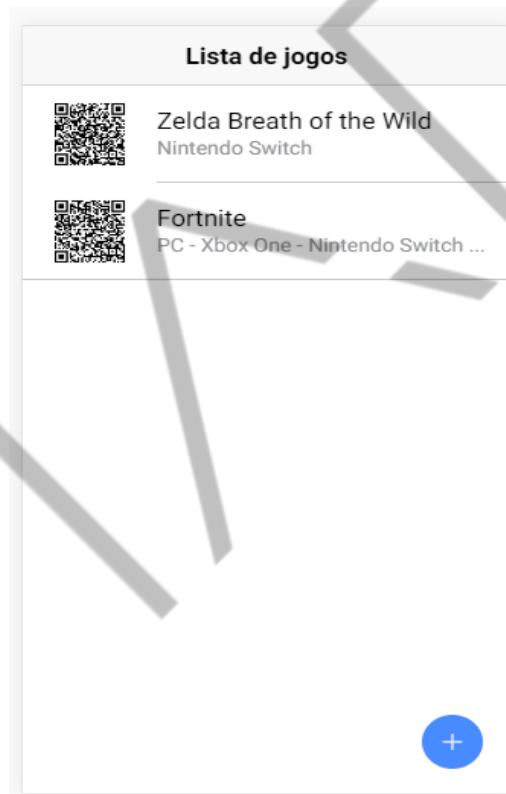


Figura 6.15 – Tela implementada pelo código
Fonte: Elaborado pelo autor (2018)

6.7 NavParams

Agora, vamos receber os dados em nossa DetailPage e inserir novos dados na HomePage. Implemente o seguinte código no **detail/detail.ts**:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';
import { GamesProvider } from '../providers/games/games';

@Component({
  selector: 'page-detail',
  templateUrl: 'detail.html',
})
export class DetailPage {
  name: string = "";
  platform: object[] = [];
  qrCode: string = null;

  switch: boolean;
  pc: boolean;
  ps4: boolean;
  xbox: boolean;

  constructor(
    public gameProvider: GamesProvider,
    public navParams: NavParams) {
  }

  ionViewWillEnter() {
    const game = this.navParams.get('game')

    if(game) {
      this.name = game.name;
      this.platform = game.platform;

      this.updateCheckBoxes(this.platform);
    }
  }

  ionViewWillLeave() {
    const game = {
      name: this.name,
      platform: this.platform
    }

    if(this.navParams.get('game')) {
      this.gameProvider.update(game,
this.navParams.get('index'))
    } else {
      this.gameProvider.insert(game)
    }
  }

  createCode() {
    const qrData = { name: this.name, ...this.platform };
    this.qrCode = JSON.stringify(qrData);
  }
}
```

```
}

updatePlatform(checked, name) {
  if(checked) {
    this.platform.push(name);
  } else {
    this.platform.splice(this.platform.indexOf(name), 1);
  }
}

updateCheckBoxes(platform) {
  platform.forEach(name => {
    if(name === 'Nintendo Switch') {
      this.switch = true
    }
    if(name === 'PC') {
      this.pc = true
    }
    if(name === 'Playstation 4') {
      this.ps4 = true
    }
    if(name === 'Xbox One') {
      this.xbox = true
    }
  })
}
```

Código-fonte 6.15 – Refatorando o detail.ts

Fonte: Elaborado pelo autor (2018)

Refatoramos o `DetailPage` para utilizar os métodos **`ionViewWillEnter()`** e **`ionViewWillLeave()`**, que serão executados durante o ciclo de vida de uma `Page` (no caso, quando a tela entrar e sair).

No **`ionViewWillEnter()`**, pegamos o dado passado com o serviço **`navParams`** e executamos o método **`updateCheckBoxes()`** a fim de preenchermos no *checkbox*, se houver alguma plataforma selecionada. Adicionamos tudo isso dentro de uma condição para que, caso seja um novo jogo, os dados permaneçam vazios.

No **`ionViewWillLeave()`**, realizamos a inserção ou alteração de nossa lista, de acordo com a condição do `navParams`, para verificar se é um novo *game* ou é uma alteração.

No HTML, realizaremos a seguinte atualização:

```
<ion-header>

  <ion-navbar>
    <ion-title>Detalhes do jogo</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <ion-list>
    <ion-list-header>Detalhes</ion-list-header>

    <ion-item>
      <ion-label color="primary" stacked>Nome do jogo</ion-label>
      <ion-input [(ngModel)]="name" type="text"></ion-input>
    </ion-item>

    <ion-item>
      <ion-label>Nintendo Switch</ion-label>
      <ion-checkbox [(ngModel)]="switch"
(ionChange)="updatePlatform(switch, 'Nintendo Switch')"></ion-checkbox>
    </ion-item>

    <ion-item>
      <ion-label>PC</ion-label>
      <ion-checkbox [(ngModel)]="pc"
(ionChange)="updatePlatform(pc, 'PC')"></ion-checkbox>
    </ion-item>

    <ion-item>
      <ion-label>Playstation 4</ion-label>
      <ion-checkbox [(ngModel)]="ps4"
(ionChange)="updatePlatform(ps4, 'Playstation 4')"></ion-checkbox>
    </ion-item>

    <ion-item>
      <ion-label>Xbox One</ion-label>
      <ion-checkbox [(ngModel)]="xbox"
(ionChange)="updatePlatform(xbox, 'Xbox One')"></ion-checkbox>
    </ion-item>

    <ion-list-header>QrCode</ion-list-header>
  </ion-list>

  <ion-card *ngIf="qrCode">
    <ngx-qr-code [qrc-value]="qrCode"></ngx-qr-code>
  </ion-card>
```



```
<button ion-button full (click)="createCode()">Gerar  
QRCode</button>  
</ion-content>
```

Código-fonte 6.16 – Refatorando o detail.html
Fonte: Elaborado pelo autor (2018)

EMANIP

CONCLUSÃO

Em apenas um capítulo, já geramos nosso primeiro aplicativo!

Todo o código aplicado neste conteúdo está disponível no github deste link: <https://github.com/diegodsgarcia/qrcoder-games-generator/tree/part-I>.

Porém, vimos nosso aplicativo apenas em um servidor. Que tal, finalmente, começarmos a vê-lo no celular, adicionando algumas funcionalidades e utilizando recursos nativos de smartphone?

Veja tudo isso no próximo capítulo.

REFERÊNCIAS

APACHE CORDOVA. **Overview.** Disponível em: <<https://cordova.apache.org/docs/en/latest/guide/overview/index.html>>. Acesso em: 10 out. 2019.

IONIC ACADEMY. **Ionic QR Code Generator & Reader [v3].** 2017. Disponível em: <<https://ionicacademy.com/ionic-qr-code-generator-reader/>>. Acesso em: 10 out. 2019.

IONIC FRAMEWORK. **Ionic 4 is coming.** [s.d.]. Disponível em: <<https://ionicframework.com/docs/>>. Acesso em: 10 out. 2019.