DESENVOLVIMENTO DE APLICATIVO DE CADASTRAMENTO PARA PROFISSSIONAIS DA CONSTRUÇÃO CIVIL

Development of Registration Application for Civil Construction Professions

Bruno Cícero Lyrio[1]
Renan Aparecido Peralta[2]
Leticia Martelo Pagoto[3]
Marcos Antônio Estremote[4]

RESUMO

Este trabalho apresenta o desenvolvimento de um aplicativo para cadastramento de profissionais do setor da construção civil. O interesse surgiu por considerar o alto índice de desemprego no setor e percebendo a necessidade de viabilizar, cada vez mais, ferramentas que visam a contratação de profissionais. Nesse sentido, visando corroborar com a empregabilidade do setor, o objetivo principal foi o de desenvolver um aplicativo funcional. O desenvolvimento do aplicativo concentra-se em ambiente de programação Java, sendo complementado por plataformas que visam torná-lo mais atraente e de fácil acesso. Após conclusão do desenvolvimento do aplicativo, o mesmo foi submetido a teste, o qual, a partir de relatos, comprovaram o êxito do recurso e da aplicação.

Palavras-chave: Aplicativo; Código; Programação

ABSTRACT

This work presents the development of an application for registering professionals in the civil construction sector. The interest arose from considering the high unemployment rate in the sector and realizing the need to increasingly enable tools aimed at hiring professionals. In this sense, aiming to corroborate the employability of the sector, the main objective was to develop a functional application. The development of the application focuses on the Java programming environment, being complemented by platforms that aim to make it more attractive and easily accessible. After completing the development of the application, it was subjected to testing, which, based on reports, proved the success of the resource and application.

Keyword: Application; Code; Programming

1 INTRODUÇÃO

No mercado de trabalho atual, cada vez mais dinâmico e competitivo, não é raro encontrar profissionais com dificuldades em exercer sua função em uma área de

[1] Graduando do curso de Engenharia Civil, das Faculdades Integradas “Rui Barbosa” (FIRB), Universidade Brasil – e-mail: bruno_lirio@hotmail.com
[2] Graduando do curso de Engenharia Civil, das Faculdades Integradas “Rui Barbosa” (FIRB), Universidade Brasil – e-mail: raperalta@outlook.com
[3] Orientadora: Mestra em Engenharia Civil pela Universidade Estadual Paulista Júlio de Mesquita Filho, Brasil (2019), Colaboradora da Universidade Estadual Paulista Júlio de Mesquita Filho, docente no curso de Engenharia Civil, nas Faculdades Integradas “Rui Barbosa” (FIRB), Universidade Brasil – e-mail: leppagoto@yahoo.com.br
[4] Coorientador: Doutor em Engenharia Elétrica pela Universidade Estadual Paulista Júlio de Mesquita Filho (2017). Professor titular do Centro Universitário de Jales, docente no curso de Engenharia Civil, nas Faculdades Integradas “Rui Barbosa” (FIRB), Universidade Brasil, FUNEC em Santa Fé do Sul e professor titular do Centro Estadual de Educação Tecnológica Paula Souza – e-mail: estremot@gmail.com

atuação, além do mais, problemas com a economia vem assombrando o país há anos, o que gera desemprego, mudança de profissão e até mesmo o encobrimento de um ensino superior no currículo do profissional. Observa-se, então, que com uma ferramenta tão poderosa quanto a tecnologia, faz-se necessário, em âmbito profissional, sempre estar atualizado para que se possa precaver um futuro desemprego, uma estagnação no trabalho e o descarte de um ensino superior.

No setor da construção civil, um grande empecilho para o trabalhador é encontrar atividades para que possa exercer sua profissão, uma vez que muitos estão na condição da informalidade; na mesma situação encontra-se o empregador, que tem dificuldades em encontrar um profissional totalmente qualificado para a função que deseja ser exercida, pois, cada parte constituinte de uma obra exige um profissional qualificado, para atingir resultado final satisfatório.

Segundo a Câmara Brasileira de Indústria da Construção (CBIC, 2020), houve uma recuperação na área da construção civil em 2020. O setor registrou expansão de 1,6% em seu Produto Interno Bruto (PIB), no ano de 2018, e esperava consolidar a retomada de suas atividades, porém passou por cinco anos de queda (2014-2018) apresentando um longo período de mau desempenho. Considerando o cenário descrito, é premente viabilizar, cada vez mais, maneiras de alavancar a taxa de empregabilidade no setor da construção civil, impulsionando outros setores e buscando uma projeção de melhora para a economia, para que todos sejam beneficiados.

Logo, visando prosperar em um setor que representa um pilar do país e vivendo o auge da tecnologia, aplicada frequentemente para reduzir custo, otimizar tempo e melhorar desempenho de uma ação, a junção entre o que se espera e a referida ferramenta torna-se mais viável para se obter bons resultados. Em contrapartida, em se tratando de um setor tão desenvolvido e, ao mesmo tempo, tão tradicional, é inevitável refutar a abrangência do aplicativo para todos os profissionais do ramo, visto que ainda há um número significativo de profissionais que não lidam com a tecnologia frequentemente. Nesse sentido, o aplicativo, apesar de ser um recurso tecnológico, tem a possibilidade de abranger a divulgação de profissionais e facilitar o acesso à quem deseja contratar devido ao ambiente ser de fácil acesso com símbolos já assimilados no meio social.

O desenvolvimento de um aplicativo, capaz de mediar e facilitar o encontro entre o prestador de serviço e o empregador, objeto de estudo proposto neste artigo, reveste-se de importância uma vez que tem o intuito de contribuir com mais um recurso

para que o profissional consiga prestar um maior número de serviços em sua área de atuação.

Visando cumprir a finalidade a qual o presente artigo se propõe, tem-se como base a programação, sendo esta imprescindível para o desenvolvimento do aplicativo, o qual foi nomeado por “Emprega Civil”, uma vez que, toda a estrutura da aplicação é baseada em códigos de linguagem de programação. Logo, optou-se pela utilização da linguagem “Java” e, como leitor da linguagem, foi utilizado o “Android Studio”, por apresentar uma interface mais atraente em relação aos demais. Para armazenamento de informações, concluiu-se que a utilização da plataforma *“Firebase”* atendia melhor à recomendação, devido à rápida implementação na aplicação. Assim, foi possível atestar que a programação é fundamental para o desenvolvimento de qualquer aplicação, como desenvolvida para este aplicativo.

O presente artigo teve como objetivo desenvolver um aplicativo de cadastramento de profissionais do setor da construção civil, funcionando como uma vitrine virtual de prestadores de serviço, utilizando a linguagem de programação “Java”, por meio do “Android Studio”, contribuindo com os mesmos no exercício de sua profissão. Em complemento ao desenvolvimento do aplicativo, testes de utilização como profissionais e empregadores foram realizados a fim de avaliar as funções do mesmo e a sua aplicabilidade na sociedade.

2 FUNDAMENTAÇÃO TEÓRICA

A construção civil ocupa lugar significativo na vida em sociedade há séculos. Desde os primórdios, utilizavam-se construções feitas de pedra e barro, assim, até os dias atuais, a sociedade está envolta em construções, como residências, edificações empresariais, e mesmo a pavimentação. Observa-se que, além de oferecer qualidade de vida, o setor da construção civil tem a capacidade de impulsionar a economia, elevar as taxas de empregabilidade e contribuir positivamente com o PIB do país.

Em paralelo aos avanços da construção civil, encontra-se o avanço tecnológico, necessário em todas as áreas, nos dias atuais. Segundo Bekaert (2018), “a inovação é uma característica natural e necessária da construção civil, sendo a responsável por sua constante evolução”. Além de benefícios na atuação de profissionais do setor, a tecnologia influencia diretamente nas buscas e contratações, auxiliando departamentos

de recursos humanos, pequenas empresas e pessoas físicas a captar, avaliar e contratar profissionais adequados aos preenchimentos de vagas disponíveis.

Sabe-se que, atualmente, diversas empresas intermediam a contração de profissionais por meio de plataformas virtuais. Nesse sentido atua a “Catho” realizando recrutamento de profissionais, contando com mais de 220.000 vagas de emprego e retendo, exclusivamente, 38% dos anúncios de vagas das empresas brasileiras (CATHO, 2020), representando parte considerável nos intermédios de contratações.

De maneira parecida, atua a rede social corporativa “*Linkedin”,* que utiliza suas plataformas para apresentar as aptidões de profissionais de maneira que facilite a busca dos mesmos e os encaminhem para o mercado de trabalho. Segundo Guilherme Amado, jornalista da revista “Época”, a rede conta com 41 milhões de usuários no Brasil. Todavia, a quantidade de usuários não reflete diretamente a quantidade de contratações, especialmente, no setor de construção civil, onde se encontra taxas de instabilidades altas no segmento (AMADO, 2020).

Apesar do setor da construção civil ter apresentado bons sinais de recuperação para 2020, a taxa de emprego do setor continuou defasada. De acordo com a economista do Banco de Dados da CBIC Ieda Vasconcelos (CBIC, 2020), o país não apresentava uma queda tão grande desde 2012. Segundo a economista, “Apesar de grande parte relativa aos três primeiros meses do ano ainda estar fora do cenário da pandemia provocada pela Covid-19, é preciso considerar que o impacto no mercado de trabalho do setor já foi maior do que o esperado, especialmente considerando que a projeção de crescimento para o PIB setorial, nos primeiros meses do ano, era de 3%”.

Considerando o cenário econômico, a defasagem do setor e a necessidade de promover empregabilidade, unindo com as possibilidades da tecnologia, o desenvolvimento de um aplicativo de cadastramento de profissionais da construção civil surge para auxiliar tanto o cadastramento como a consulta. Para tanto, usou-se a linguagem de programação “Java” para inserir os códigos que estruturaram a aplicação, pois é uma programação orientada a objetos, projetada para facilitar a ação do programador, favorecendo o uso em vários sistemas operacionais. Ademais, a principal característica do Java é que possui uma estrutura própria: todo o código é escrito dentro de uma classe e tudo se refere a objeto. Isso proporciona uma plataforma segura para desenvolver e executar aplicativos, protegendo a privacidade dos dados e evitando uma futura quebra de código, ou seja, não permite sua própria corrupção.

3 MATERIAIS E MÉTODOS

O método pelo qual as etapas do artigo foram desenvolvidas, cumpre a finalidade descritivo-explicativa, uma vez que a intenção se concentrou em descrever cada fase do processo de desenvolvimento do aplicativo, nomeado como Emprega Civil, cuja função principal foi o cadastramento e busca de profissionais do setor da construção civil.

A fim de conhecimento para o desenvolvimento do aplicativo, foram utilizados materiais da área de programação, como "Java™ como programar" (DEITEL,2005), "Aprenda Programação Orientada a Objetos em 21 dias" (SINTES, Anthony,2002) e "Sistema de Banco de Dados" (ELMASRI, Ramez e NAVATHE, Shamkant B,2005).

Posteriormente, as fórmulas foram utilizadas na conversão em linguagem Java, as quais foram otimizadas no "Firebase". O desenvolvimento final do aplicativo foi por meio do "Android Studio" e a publicação por meio da plataforma "Play Store", do "Google".

A partir de tais considerações, o projeto atendeu a natureza qualitativa, visto que nele foram inseridas descrições dos passos do desenvolvimento do aplicativo, os quais foram considerados para as análises descritivo-explicativas

3.1 Ambiente de programação

O aplicativo Emprega Civil foi desenvolvido em meio eletrônico, por meio do "Android Studio", que se trata de um ambiente de desenvolvimento integrado (IDE) para Android. Lançado oficialmente em dezembro 2014, o "Android Studio" foi feito especificamente para o desenvolvimento para Android e se tornou a IDE primária do "Google" de desenvolvimento nativo para Android. O "Android Studio" é disponibilizado gratuitamente sob a licença Apache 2.0.

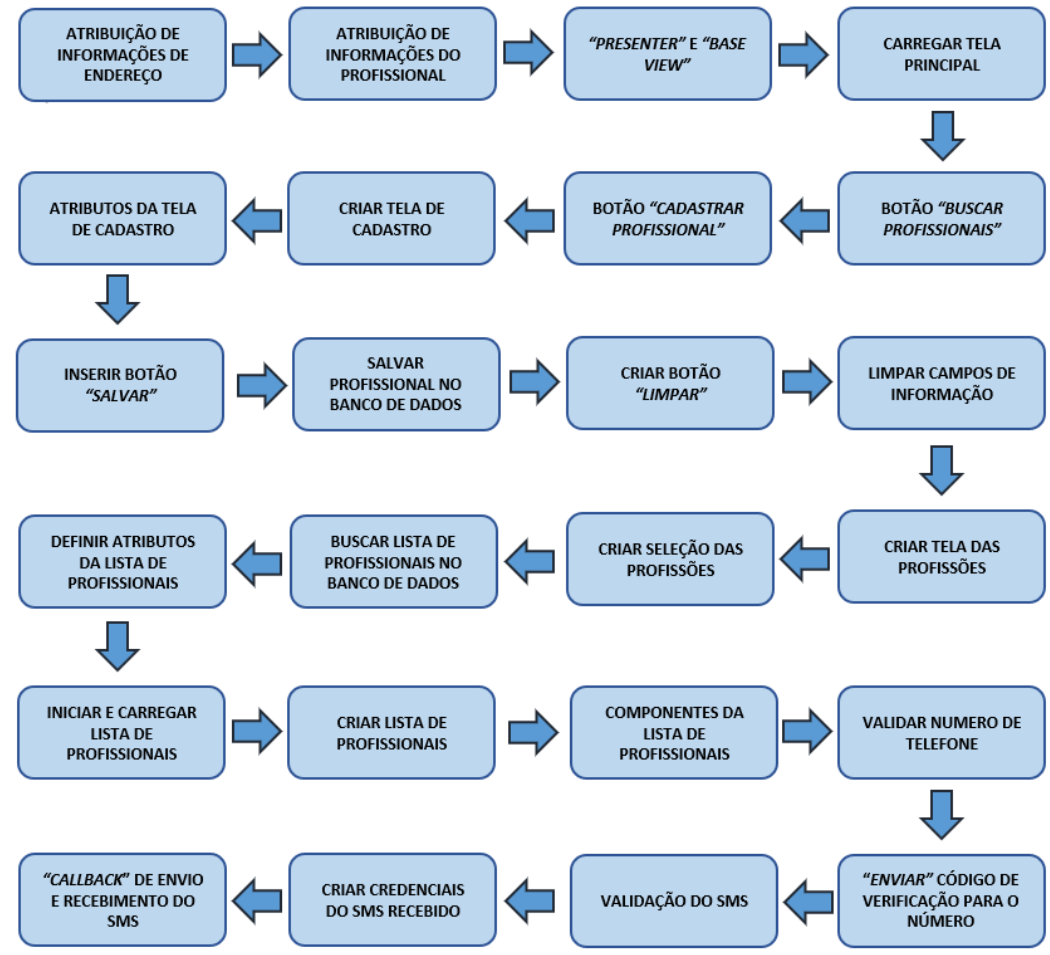
Vale destacar que, partindo do pressuposto de que pode haver alto índice de utilização e aplicação na comunidade, o banco de dados interno não seria suficiente para suprir uma quantidade de cadastros. Logo, para não limitar a aplicação a poucos usuários, foi utilizado o "Firebase", que pode ser definido como um "Baas" (Backend as a Service) para aplicações Web e Mobile do "Google", lançado em 2004. Assim como os outros "Bass" desenvolvidos, o "Firebase" auxilia os desenvolvedores a acelerar a criação de aplicações web e mobile e simplificam a criação de APIs. Além disso,

possibilita aos desenvolvedores montar os blocos de construção necessários e apenas escrever o código que os conecta. De uma forma mais clara, ele é responsável por otimizar a experiência do desenvolvimento da aplicação e é um serviço que permite salvar dados em nuvem.

Como etapa final da aplicação, a publicação da mesma foi por meio da “Play Store”, loja de aplicativos do “Google”. A *priori,* foi publicado na versão alfa da plataforma, configurada para teste fechado e disponibilizado apenas para alguns professores, orientadores e demais pessoas, como o intuito de possibilitar a experiência de um real funcionamento da aplicação e atestar a total funcionalidade do aplicativo.

Visando demostrar os processos de programação com mais clareza, melhorando a compreensão do desenvolvimento do aplicativo, foi organizado um fluxograma de processo das etapas de programação, como apresentado na Figura 1.

Figura 1 - Fluxograma de processo das etapas de programação do aplicativo.



Fonte: Elaborada pelos autores.

3.2 Manual de utilização

Um manual de utilização cumpre a função de orientar um usuário, de maneira simples e detalhada, como proceder para pôr em funcionamento determinado produto ou serviço. Nesse sentido, visando cumprir com o objetivo de dispor o aplicativo para usuários, que oferece informações para facilitar o cadastramento e o acesso aos profissionais do setor da construção civil, foi elaborado o manual de utilização do "Emprega Civil" por meio do software de designer gráfico, "*Corel Draw*", que permite a manipulação de imagens e criação de desenho vetorial bidimensional.

3.3 Testes no aplicativo

Os testes no aplicativo foram realizados por meio de contato virtual com possíveis profissionais e empregadores, em razão do distanciamento social ocasionado pela pandemia do Covid-19.

Foram listados dez profissionais, engenheiros, para testar o aplicativo na função de "Cadastrar Profissional". Cada profissional fez cadastro em uma profissão entre as dez fornecidas pelo aplicativo, a fim de garantir o funcionamento e estabilidade das funções. A escolha por profissionais engenheiros foi devido à facilidade de contato no ambiente de pesquisa, já que não foi possível estender o teste aos demais profissionais, devido às condições da pandemia do Covid-19.

Para o teste da função "Buscar Profissionais" foram listados, também, dez usuários, para avaliar a aplicabilidade do "Emprega Civil" como se fossem empregadores.

Ao final do teste de possíveis profissionais e empregadores, foi enviada a mensagem, para um questionário individual de quatro questões, como descrita abaixo.

"*Olá, somos da RB Soluções e estamos desenvolvendo um aplicativo de cadastramento de profissionais da construção civil. Ele visa intermediar a relação entre o contratante e o contratado. Após a utilização do aplicativo, poderia nos ajudar respondendo este questionário?*

Considerando as notas 0 (zero) como ruim e 10 (dez) como ótimo, responda:

1- De 0 a 10, avalie com uma nota, a facilidade de utilização do Emprega Civil.

2 - De 0 a 10, avalie com uma nota, o quanto considera que o Emprega Civil será útil para o setor da construção civil.

3 - De 0 a 10, avalie com uma nota, a utilização da tecnologia atualmente no setor da construção civil.

4 - Escolha um benefício, entre os listados, que o Emprega Civil pode trazer para o setor da construção civil?

a) Divulgação de profissionais

b) Oportunidades de prestação de serviços

c) Facilidade em encontrar mão de obra qualificada.

d) Facilidade na cotação de prestadores de serviços em função da lista de profissionais disponível."

4 DISCUSSÃO DOS RESULTADOS

4.1 Aplicativo

O aplicativo foi desenvolvido para o sistema Android do "Google", cujo processo de criação considera 2 (duas) etapas: a programação e a publicação.

4.1.1 Programação

O tópico consiste na exemplificação da inserção dos primeiros aos últimos códigos do programa.

a) Código da atribuição de informações de endereço

O desenvolvimento do programa teve início com a introdução do código desenvolvedor responsável por definir atributos necessários do endereço do usuário, com essas informações tais como: endereço, cidade, estado, número e CEP, futuramente permitirá que o usuário do aplicativo realize parte do seu cadastro.

A Figura 2 mostra a programação na etapa de definição dos atributos.

Figura 2- Código de atribuições de informações de endereço.

```
/
 * Classe que define os atributos do endereço
 */
public class Address {
    private String CEP;
    private String address;
    private String comp;
    private String city;
    private String state;

    public String getCEP() { return CEP; }

    public void setCEP(String CEP) { this.CEP = CEP; }

    public String getAddress() { return address; }

    public void setAddress(String address) { this.address = address

    public String getComp() { return comp; }

    public void setComp(String comp) { this.comp = comp; }

    public String getCity() { return city; }

    public void setCity(String city) { this.city = city; }

    public String getState() { return state; }

    public void setState(String state) { this.state = state; }
```

Fonte: Elaborada pelos autores.

O termo em azul “*public*” é o modificador responsável por tornar essa classe visível para todas as outras classes em qualquer lugar, diferente do termo “*private*”, também em azul, que especifica que o membro pode ser acessado apenas em sua própria classe.

Ressalta-se que o termo “*String get*” é, basicamente, o responsável pela inclusão dos dados no programa.

b) Código da atribuição de informações do profissional

Na etapa de atribuições de informações sobre o profissional é inserido um código, que indicará as referências informacionais que serão necessárias em relação à identificação (nome) e telefone para contato e também sobre a função e a atuação do profissional para a realização do cadastro, tal como mostra a Figura 3.

Figura 3 - Código de atribuição de informações do profissional

```
 * Classe que define os atributos de um profissional
 */
public class Professional {
    private String name;
    private String phone;
    private String function;
    private String experience;
    private Address address;

    public String getId() { return getName()+"-"+getPhone().replace( target: " ",  replacement: ""); }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getPhone() { return phone; }

    public void setPhone(String phone) { this.phone = phone; }

    public String getFunction() { return function; }

    public void setFunction(String function) { this.function = function; }

    public String getExperience() { return experience; }

    public void setExperience(String experience) { this.experience = experience; }

    public Address getAddress() { return address; }

    public void setAddress(Address address) { this.address = address; }
}
```

Fonte: Elaborada pelos autores.

Estes códigos são responsáveis por atribuir aos usuários as informações que serão necessárias para a realização dos cadastros no aplicativo.

c) Código que define o *"presenter" e a "base view"*

Esta classe de criação do código que define o "*presenter*" e a "*base view*" (Figura 4) tem a função de transmitir os dados inseridos para as telas, ou seja, quando uma informação é salvada no cadastro das informações, ela é processada na classe responsável pelos dados e gera uma informação de "salvo com sucesso" ou "erro" que posteriormente é exibido na tela do aplicativo.

Figura 4 - Código *"presenter"*

```
/**
 * Classe que define o presenter
 * @param <V>
 */
public class Presenter<V extends Presenter.BaseView> {

    private V view;

    public V getView() { return view; }

    public void setView(V view) { this.view = view; }

    public void initialize (String userId) {}

    /**
     * Interface da BaseView
     */
    interface BaseView {
        void showLoading();
        void hideLoading();
        void showErrorMessage(String message);
    }
}
```

Fonte: Elaborada pelos autores

O termo *"presenter"* pode ser definido como o provedor da comunicação entre os dados com a interface, a "baseview", que tem a função de assimilar métodos usados em várias classes diferentes, definindo assim a assinatura.

d) Código responsável por carregar a tela principal do aplicativo

O código responsável por carregar a tela principal do aplicativo (Figura 5) é responsável por exibir o *logo* do aplicativo enquanto a mesma tem a função de carregar as interações iniciais, como, por exemplo, os botões "Buscar Profissionais" e "Cadastrar Profissional" demostrados nos tópicos seguintes**.**

Figura 5 – Código da tela principal do aplicativo

```
 * Classe Principal do App, em que é carregada a tela inicial
 */
public class MainActivity extends AppCompatActivity {

    @BindView(R.id.bt_serch)
    Button btSearch;

    @BindView(R.id.bt_register)
    Button btRegister;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ButterKnife.bind( target: this);
        getSupportActionBar().hide();

        clickRegister();
        clickSearchProfessional();
    }
```

Fonte: Elaborada pelos autores

e) Código responsável pelo clique do botão “*Buscar Profissionais*”

O código responsável pelo clique do botão “Buscar Profissionais” (Figura 6) é responsável por, na lista de profissionais cadastrados, direcionar o usuário para a tela das listagens das funções. Este é apenas um código de direcionamento.

Figura 6 – Código do botão “Buscar Profissionais”.

```
 * Método que define o Clique do botão buscar profissionais
 */
public void clickSearchProfessional() {
    btSearch.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(getApplicationContext(), SelectProfessionalsActivity.class);
            startActivity(intent);
        }
    });
}
```

Fonte: Elaborada pelos autores

f) Código responsável pelo clique do botão “*Cadastrar Profissional*”

O código responsável pelo clique do botão “Cadastrar Profissional” é idêntico ao anterior (item e), responsável por direcionar para a tela de cadastramento dos profissionais, como apresentado na Figura 7.

Figura 7 – Código do botão “Cadastrar Profissional”

```
    * Método que define o Clique do botão Cadastrar Profissional
    */
    public void clickRegister() {
        btRegister.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(getApplicationContext(), RegisterActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

Fonte: Elaborada pelos autores

g) Código responsável por criar a tela de cadastro

O código responsável por criar a tela de cadastro (Figura 8) tem a função de, após o clique do botão específico, criar uma tela contendo campos para inserir informações as quais, posteriormente, serão inseridas pelo usuário para a realização do cadastro.

Figura 8 – Código da tela de cadastro

```
    * Método que cria a tela de cadastro
    * @param savedInstanceState
    */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);
        ButterKnife.bind( target: this);

        etCep.addTextChangedListener(MaskEditUtil.mask(etCep, MaskEditUtil.FORMAT_CEP));
        etPhone.addTextChangedListener(MaskEditUtil.mask(etPhone, MaskEditUtil.FORMAT_PHONE));
        configureToolbar();

        professionalPresenter = new ProfessionalPresenter();
        professionalPresenter.setView(this);

        clickSave();
        clickClear();
        clickBack();
        getProfessional1(getIntent().getExtras());
        mDialog = new Dialog( context: this);

        mAuth = FirebaseAuth.getInstance();
    }
```

Fonte: Elaborada pelos autores

h) Código responsável por definir os atributos da tela de cadastro

O código relacionado aos atributos da tela de cadastro é responsável por definir as informações que o profissional deve inserir para realizar o cadastro. Os campos de digitação anteriormente inseridos devem ser preenchidos com as devidas informações de cada usuário, como apresentado pela programação da Figura 9.

Figura 9 – Código de atribuições de informações para realizar o cadastro

```
/* Atributos da tela de cadastro*/
@BindView(R.id.progress_bar)
ProgressBar progressBar;

@BindView(R.id.et_name)
EditText etName;

@BindView(R.id.et_phone)
EditText etPhone;

@BindView(R.id.et_experience)
EditText etExperience;

@BindView(R.id.et_cep)
EditText etCep;

@BindView(R.id.et_address)
EditText etAddress;

@BindView(R.id.et_comp)
EditText etComp;

@BindView(R.id.et_city)
EditText etCity;

@BindView(R.id.et_state)
EditText etState;

@BindView(R.id.tv_function)
TextView tvFunction;

@BindView(R.id.spn_functions)
Spinner spnFunctions;

@BindView(R.id.bt_save)
Button btRegister;

@BindView(R.id.bt_clear)
Button btClear;

@BindView(R.id.bt_cancel)
Button btCancel;
```

Fonte: Elaborada pelos autores

O termo em azul *“Edittext”* é responsável por tornar o campo vazio em um campo editável pelo usuário, abrindo o teclado para que ele possa digitar suas informações.

i) Código responsável por inserir o botão “Salvar”

Este código é responsável por inserir o botão “Salvar” (Figura 10) cuja função é, quando o usuário do aplicativo clicar, ser direcionado para o salvamento das informações que foram digitadas anteriormente pelo profissional.

Figura 10 – Código de implementação do botão *“Salvar”.*

```
/*
 * Implementando click do botão salvar cadastro
 */
public void clickSave() {
    btRegister.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Address address = new Address();
            address.setCEP(etCep.getText().toString());
            address.setAddress(etAddress.getText().toString());
            address.setComp(etComp.getText().toString());
            address.setCity(etCity.getText().toString());
            address.setState(etState.getText().toString());

            professional = new Professional();
            professional.setName(etName.getText().toString());
            professional.setPhone(etPhone.getText().toString());
            professional.setFunction(spnFunctions.getSelectedItem().toString());
            professional.setExperience(etExperience.getText().toString());
            professional.setAddress(address);

            if (professional.getName().isEmpty()) {
                Toast.makeText( context: RegisterActivity.this, "Preencha o nome do profissional antes de salvar", Toast.LENGTH_LONG).show();
            } else if(professional.getPhone().isEmpty()) {
                Toast.makeText( context: RegisterActivity.this, "Preencha o telefone do profissional antes de salvar", Toast.LENGTH_LONG).show();
            } else if (professional.getFunction().equalsIgnoreCase("Selecione uma Profissão")) {
                Toast.makeText( context: RegisterActivity.this, "Selecione a função do profissional antes de salvar", Toast.LENGTH_LONG).show();
            } else if(professional.getExperience().isEmpty()) {
                Toast.makeText( context: RegisterActivity.this, "Preencha o expecialidade do profissional antes de salvar", Toast.LENGTH_LONG).show();
            } else {
                openDialog("Validando telefone", "Aguarde...", showProgress: true, close: false, btVerify: false);
                sendVerificationCode( number: "+55"+etPhone.getText().toString().replace( target: "(", replacement: "").replace( target: ")", replacement: "").replace( target: "-", replacement: ""));

            }
        }
    });
}
```

Fonte: Elaborada pelos autores

O termo *“if”,* em azul, serve para, se o profissional deixar os campos como nome, telefone, e especialização vazios, ou se não escolher a sua profissão, não consiga realizar seu cadastro. Na sequência, aparecerá uma mensagem na tela induzindo o profissional a preencher as informações faltantes para assim realizar o cadastro. Tomando, como exemplo, caso o profissional esqueça de preencher o campo de informação do telefone, ao tentar clicar no botão *“Salvar”* aparecerá uma mensagem na tela *“Preencha o telefone do profissional antes de salvar”.*

j) Código responsável por salvar os dados do profissional no banco de dados.

O código responsável por salvar os dados do profissional no banco de dados encontra-se no *"firebase",* ou melhor, no banco de dados, possibilitando, assim, que, futuramente, apareça na pesquisa de algum empregador. Como tal, a programação está descrita na Figura 11.

Figura 11 – Código que salva o profissional no banco de dados.

```
 * Método que salva um profissional no firebase
 * @param professional
 * @param context
 */
public void requestSaveProfessional(final Professional professional, final Context context) {

    FirebaseFirestore db = FirebaseFirestore.getInstance();

    Map<String, Object> user = new HashMap<>();
    user.put("id", professional.getId());
    user.put("name", professional.getName());
    user.put("phone", professional.getPhone());
    user.put("function", professional.getFunction());
    user.put("experience", professional.getExperience());
    user.put("cep", professional.getAddress().getCEP());
    user.put("address", professional.getAddress().getAddress());
    user.put("comp", professional.getAddress().getComp());
    user.put("city", professional.getAddress().getCity());
    user.put("state", professional.getAddress().getState());
    getView().showLoading();

    db.collection( collectionPath: "professionals").document( documentPath: "professions") DocumentReference
            .collection(professional.getFunction()).document(professional.getId()) DocumentReference
            .set(user) Task<Void>
            .addOnSuccessListener(new OnSuccessListener<Void>() {
                @Override
                public void onSuccess(Void aVoid) {
                    Log.d( tag: "TAG",  msg: "DocumentSnapshot successfully written!");
                    getView().showSucessRegister("Profissional cadastrado com sucesso");
                    getView().hideLoading();
                }
            }) Task<Void>
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.w( tag: "TAG",  msg: "Error writing document", e);
                    getView().showErrorMessage("Erro ao cadastrar Profissional");
                    getView().hideLoading();
                }
            });
}
```

Fonte: Elaborada pelos autores.

k) Código responsável por criar o botão *"Limpar"*

O código responsável por criar o botão "*Limpar*" (Figura 12) é responsável por criar o botão que permitirá que sejam apagadas informações digitadas.

Figura 12 – Código de implementação do botão *"Limpar"*

```
    * Implementando click do botão Limpar
    */
public void clickClear() {
    btClear.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { clearDatas(); }
    });
}

/**
 * Implementando click do botão voltar
 */
public void clickBack() {
    btCancel.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { onBackPressed(); }
    });
}
```

Fonte: Elaborada pelos autores.

l) Código responsável por limpar os campos de informações.

O código responsável pela criação do botão "limpar" dispõe um botão para excluir todas as informações que foram digitadas nos campos editáveis, tal como mostrado na tela de programação na Figura 13.

Figura 13 – Código responsável por excluir informações digitadas

```
    * Limpando dados da tela
    */
public void clearDatas() {
    etName.setText("");
    etPhone.setText("");
    etExperience.setText("");
    etCep.setText("");
    etAddress.setText("");
    etComp.setText("");
    etCity.setText("");
    etState.setText("");
}
```

Fonte: Elaborada pelos autores.

m) Código responsável por criar a tela das profissões

O código responsável por criar a tela das profissões (Figura 14) tem a função de, após o clique do botão específico, criar uma tela contendo 10 (dez) áreas de atuações do profissional no segmento da construção civil, tais como: Engenheiro civil, Engenheiro Eletricista, Arquiteto, Encanador, Pedreiro, Servente, Gesseiro, Carpinteiro, Pintor e Eletricista.

Figura 14 – Código da tela de profissões.

```
 * Classe em que é criada a tela de seleção da profissão
 */
public class SelectProfessionalsActivity extends AppCompatActivity {

    /* Atributos da clase de seleção de profissões */
    @BindView(R.id.civil_engineer)
    Button btCivilEngineer;

    @BindView(R.id.electric_engineer)
    Button btElectricEngineer;

    @BindView(R.id.architect)
    Button btArchitect;

    @BindView(R.id.plumber)
    Button btPlumber;

    @BindView(R.id.bricklayer)
    Button btBricklayer;

    @BindView(R.id.hodman)
    Button btHodman;

    @BindView(R.id.plasterer)
    Button btPlasterer;

    @BindView(R.id.carpenter)
    Button btCarpenter;

    @BindView(R.id.painter)
    Button btPainter;

    @BindView(R.id.electrician)
    Button btElectrician;

    public static final String FUNCTION_NAME = "functionName";
```

Fonte: Elaborada pelos autores

n) Código responsável por criar a tela de seleção das profissões

A função de criar a seleção de profissões é executada pelo código que, de maneira mais exemplificada, na tela das profissões, cria 10 botões (profissões), na qual o usuário pode selecionar e clicar na profissão que se deseja encontrar um prestador de serviço. A Figura 15 demonstra a programação.

Figura 15 – Código de criação da tela de seleção de profissões

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_select_professionals);
    configureToolbar();
    ButterKnife.bind( target: this);
    clickButton(btCivilEngineer);
    clickButton(btElectricEngineer);
    clickButton(btArchitect);
    clickButton(btPlumber);
    clickButton(btBricklayer);
    clickButton(btHodman);
    clickButton(btPlasterer);
    clickButton(btCarpenter);
    clickButton(btPainter);
    clickButton(btElectrician);
}
private void configureToolbar() {
    getSupportActionBar().setTitle("Buscar Profissionais");
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    getSupportActionBar().setDisplayShowHomeEnabled(true);
}
@Override
public boolean onSupportNavigateUp() {
    onBackPressed();
    return super.onSupportNavigateUp();
}
public void clickButton(Button button) {
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(getApplicationContext(), ProfessionalListActivity.class);
            intent.putExtra(FUNCTION_NAME, button.getText());
            startActivity(intent);
        }
    });
}
```

Fonte: Elaborada pelos autores

Ela apresenta ao usuário do aplicativo 10 (dez) opções (botões) de profissões permitindo ao mesmo que seja feita a pesquisa na profissão a qual o usuário deseja encontra um prestador de serviço.

o) Código responsável por buscar a lista de profissionais no banco de dados

A classe que cria o código responsável por buscar a lista de profissionais no banco de dados (Figura 16) disponibiliza um item para que o usuário clique em uma determinada função que possibilitará a ligação do banco de dados com a interface da aplicação.

Figura 16 – Código de busca da lista de profissionais

```
 * Método que busca a lista de profissionais no firebase
 * @param context
 * @param profession
 */
public void getProfessionalsList(final Context context, String profession) {
    final ArrayList<ItemProfessional> listProfessionalFirebase = new ArrayList<>();
    getView().showLoading();
    String EMPTY = "Nenhum "+ profession +" cadastrado até o momento!";

    FirebaseFirestore db = FirebaseFirestore.getInstance();

    db.collection( collectionPath: "professionals") CollectionReference
            .document( documentPath: "professions") DocumentReference
            .collection(profession) CollectionReference
            .get() Task<QuerySnapshot>
            .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {
                    if (task.isSuccessful()) {
                        // listPlatesFirebase.clear();
                        for (QueryDocumentSnapshot document : task.getResult()) {
                            Log.d( tag: "TAG", msg: document.getId() + " => " + document.getData());
                            if (document.getData() != null) {
                                ItemProfessional itemProfessional = new ItemProfessional();
                                itemProfessional.setName(document.getData().get("name").toString());
                                itemProfessional.setPhone(document.getData().get("phone").toString());
                                listProfessionalFirebase.add(itemProfessional);
                            }
                        }
                        getView().setupProfessionalList(listProfessionalFirebase);
                        getView().hideLoading();

                        if (listProfessionalFirebase.isEmpty()) {
                            getView().showErrorMessage(EMPTY);
                        }
                    } else {
                        Log.d( tag: "TAG", msg: "Error getting documents: ", task.getException());
                        getView().hideLoading();
                        getView().showErrorMessage(EMPTY);
                    }
                }
```

Fonte: Elaborada pelos autores

p) Código responsável por definir os atributos da lista de profissionais

A etapa de criação do código responsável por definir e atributos da lista de profissionais (Figura 17) é responsável pela definição das informações que serão

necessárias que sejam inseridas para que a mesmas sejam listadas e mostradas na lista de profissionais, ou seja, ela especifica que as informações da identidade (nome) e telefone do profissional sejam exibidas quando o usuário estiver na lista de profissionais.

Figura 17 – Código de atribuição da lista de profissionais

```java
/**
 * Classe que define os atributos do item da lista de profissionais
 */
public class ItemProfessional {

    private String name;
    private String phone;

    public ItemProfessional() {

    }

    public ItemProfessional(String name, String phone) {
        this.name = name;
        this.phone = phone;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getPhone() { return phone; }

    public void setPhone(String phone) { this.phone = phone; }
}
```

Fonte: Elaborada pelos autores

q) Código responsável por iniciar e carregar a lista de profissionais

Na etapa de criação do código responsável por iniciar e carregar a lista de profissionais, foi criado um código específico por carregar toda a lista de prestadores de serviços que foram cadastrados anteriormente. Ele busca no banco de dados os cadastros para que sejam mostrados na tela do usuário (Figura 18).

Figura 18 – Código de carregamento dos cadastros dos usuários

```
    * Iniciando e carregando o componente da lista de profissionais
    * @param listProfessionalFirebase
    */
    private void initRecyclerViewPlates(final ArrayList<ItemProfessional> listProfessionalFirebase) {
        mRecycleProfessional.setHasFixedSize(true);
        mRecentLayoutManager = new LinearLayoutManager( context: this);
        mRecycleProfessional.setLayoutManager(mRecentLayoutManager);


        mAdapterPlates = new RecyclerView.Adapter<ProfessionalListActivity.CustomViewHolder>() {
            @Override
            public ProfessionalListActivity.CustomViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
                View view = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.professional_item
                        , viewGroup,  attachToRoot: false);
                return new ProfessionalListActivity.CustomViewHolder(view);
            }

            @Override
            public void onBindViewHolder(ProfessionalListActivity.CustomViewHolder viewHolder, int i) {
                viewHolder.tvProfession.setText(listProfessionalFirebase.get(i).getName());
                viewHolder.tvPhone.setText(listProfessionalFirebase.get(i).getPhone());
                viewHolder.clickDetailProfessional();
            }

            @Override
            public int getItemCount() { return listProfessionalFirebase.size(); }

        };
        mRecycleProfessional.setAdapter(mAdapterPlates);
    }
```

Fonte: Elaborada pelos autores

r) Código responsável por criar a lista de profissionais

O código responsável por criar a lista de profissionais (Figura 19) é responsável por listar os profissionais que foram cadastrados e carregados no item *q.* Basicamente, tem a função de ordenar e organizar os profissionais cadastrados em listas e exibi-los na tela da aplicação.

Figura 19 – Código de carregamento dos cadastros dos usuários

```
 * Método que cria a classe da lista de profissionais
 * @param savedInstanceState
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_professional_list);
    ButterKnife.bind( target: this);

    getViewName(getIntent().getExtras());
    professionalListPresenter = new ProfessionalListPresenter();
    professionalListPresenter.setView(this);
    professionalListPresenter.getProfessionalsList(getApplicationContext(), profession);
}
```

Fonte: Elaborada pelos autores

s) Código responsável pelos componentes do item da lista de profissionais

No item *p* foi demostrado o código de atribuições de informações da lista de profissionais. Este código (Figura 20) é o responsável por reproduzir as informações atribuídas em lista, de forma mais clara, esse código utiliza as informações que lhe foram atribuídas e as exibe em lista.

Figura 20 – Código dos componentes do item da lista de profissionais

```
 * Classe que define os componentes do item da lista de profissionais
 */
public class CustomViewHolder extends RecyclerView.ViewHolder {
    TextView tvProfession;
    TextView tvPhone;
    ImageButton btDetailProfessional;

    public CustomViewHolder(View itemView) {
        super(itemView);
        tvProfession = (TextView) itemView.findViewById(R.id.tv_professional);
        tvPhone = (TextView) itemView.findViewById(R.id.tv_phone);
        btDetailProfessional = (ImageButton) itemView.findViewById(R.id.bt_detail_professional);
    }

    public void clickDetailProfessional() {
        btDetailProfessional.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(getApplicationContext(), RegisterActivity.class);
                intent.putExtra(PROFESSIONAL_ID,  value: tvProfession.getText().toString()+"-"+tvPhone.getText().toString().replace( target: " ",  replacement: ""));
                intent.putExtra(IS_DETAIL,  value: true);
                intent.putExtra(PROFESSION, profession);
                startActivity(intent);
            }
        });
    }
}
```

Fonte: Elaborada pelos autores

Em especial nesta aplicação, são exibidos os dados informacionais da identidade (nome) e telefone do profissional quando o usuário acessa a lista de profissionais, como uma forma de pré-visualização do profissional cadastrado.

t) Código responsável por validar o número de telefone

Este código (Figura 21) é responsável por realizar a verificação do número de telefone, ou seja, ele aponta se o telefone está errado e informa ao usuário ou se está certo para prosseguir com a validação**.**

Figura 21 – Validação do número de telefone

```
public void openDialog(String title, String action, boolean showProgress, boolean close, boolean btVerify) {

    if (mDialog != null && close) {
        mDialog.dismiss();
    } else {
        mDialog.setContentView(R.layout.dialog);

        final ProgressBar progressBar = (ProgressBar) mDialog.findViewById(R.id.progressBar);
        TextView tvTitleDialog = (TextView) mDialog.findViewById(R.id.tv_title_dialog);
        //TextView tvActionDialog = (TextView) mDialog.findViewById(R.id.tv_action_dialog);
        EditText etSMSCode = (EditText) mDialog.findViewById(R.id.et_sms_code);

        TextView btDialog = (Button) mDialog.findViewById(R.id.bt_dialog);

        tvTitleDialog.setText(title.toUpperCase());
        etSMSCode.setText(action);

        if(showProgress) {
            progressBar.setVisibility(View.VISIBLE);
            btDialog.setVisibility(View.GONE);
        } else {
            if (btVerify) {
                btDialog.setText("Validar telefone");
            }
            progressBar.setVisibility(View.GONE);
            btDialog.setVisibility(View.VISIBLE);
            btDialog.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    if (btVerify) {
                        String code = etSMSCode.getText().toString().trim();
                        if (code.isEmpty() || code.length() < 6) {
                            etSMSCode.setError("Erro ao validar telefone do profissional");
                            etSMSCode.requestFocus();
                            return;
                        }
                        verifyCode(code);
                    } else {
                        mDialog.dismiss();
                        professionalPresenter.requestSaveProfessional(professional, getApplicationContext());
                    }
                }
            });
        }

        mDialog.getWindow().setBackgroundDrawable(new ColorDrawable(android.graphics.Color.TRANSPARENT));
        mDialog.setCancelable(false);
```

Fonte: Elaborada pelos autores

u) Código responsável por enviar o código de verificação para o telefone do usuário

Nesta etapa, é enviado um código (Figura 22) para o telefone do usuário para que posteriormente possa ser verificado.

Figura 22 – Envio do código de verificação

```
    * Enviando código de verificação do telefoone
    * @param number
    */
    private void sendVerificationCode(String number) {
        progressBar.setVisibility(View.VISIBLE);
        PhoneAuthProvider.getInstance().verifyPhoneNumber(
                number,
                t 20,
                TimeUnit.SECONDS,
                TaskExecutors.MAIN_THREAD,
                mCallBack
        );

        progressBar.setVisibility(View.GONE);
    }
```

Fonte: Elaborada pelos autores

v) Código responsável por validar o SMS enviado para o telefone do usuário

Este código (Figura 23) é responsável por validar o SMS que foi recebido pelo usuário, recebendo a informação e apontando se o celular foi validado ou não, ou seja, é a conexão responsável por verificar se o número está correto e enviar essa informação para o celular.

Figura 23 – Validação do SMS enviado ao usuário

```
 * Validando SMS enviado para o telefone
 * @param credential
 */
private void signInWithCredential(PhoneAuthCredential credential) {
    mAuth.signInWithCredential(credential)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        openDialog("Validando telefone", smsCode, showProgress: false, close: true, btVerify: false);
                        openDialog("Telefone Validado", smsCode, showProgress: false, close: false, btVerify: false);
                    } else {
                        Toast.makeText( context: RegisterActivity.this, "Erro ao validar telefone do profissional", Toast.LENGTH_LONG).show();
                        progressBar.setVisibility(View.GONE);
                    }
                }
            });
}
```

Fonte: Elaborada pelos autores

w) Código responsável por criar as credenciais do SMS recebido pelo telefone do usuário

Este código (Figura 24) é responsável por, após a validação do telefone, trazer a informação para a verificação e repassar para a aplicação.

Figura 24 – Criando credenciais do SMS

```
    * Criando credenciais com SMS recebido e chamando a validacao do código
    * @param code
    */
    private void verifyCode(String code) {
        PhoneAuthCredential credential = PhoneAuthProvider.getCredential(verificationId, code);
        FirebaseAuth.getInstance().signOut();
        signInWithCredential(credential);
    }
```

Fonte: Elaborada pelos autores

x) Código responsável pelo *"Callback"* de envio e recebimento do SMS

Este código (Figura 25) é responsável pelo retorno da chamada de verificação do número de telefone.

Figura 25 – *"Callback"* de envio e recebimento do SMS

```
    * Callback de envio e recebimento do SMS
    */
    private PhoneAuthProvider.OnVerificationStateChangedCallbacks
            mCallBack = new PhoneAuthProvider.OnVerificationStateChangedCallbacks() {

        @Override
        public void onCodeSent(String s, PhoneAuthProvider.ForceResendingToken forceResendingToken) {
            super.onCodeSent(s, forceResendingToken);
            verificationId = s;
        }

        @Override
        public void onVerificationCompleted(PhoneAuthCredential phoneAuthCredential) {
            String code = phoneAuthCredential.getSmsCode();
            if (code != null) {
                smsCode = code;
                verifyCode(code);
            }
            openDialog("Telefone Validado", code, showProgress: false, close: false, btVerify: true);
        }

        @Override
        public void onVerificationFailed(FirebaseException e) {
            Toast.makeText( context: RegisterActivity.this, e.getMessage(), Toast.LENGTH_LONG).show();
            progressBar.setVisibility(View.GONE);
        }
    };
```

Fonte: Elaborada pelos autores

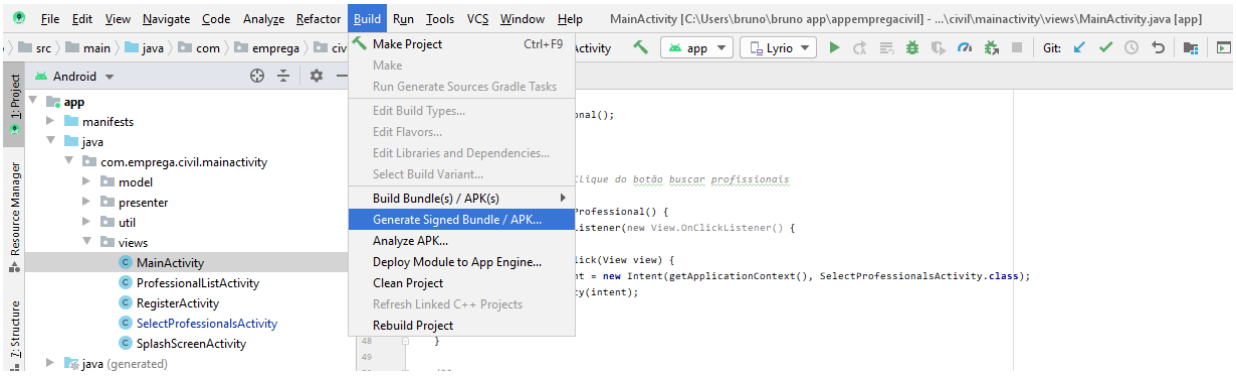
4.1.2 Publicação

O tópico consiste na exemplificação das etapas do processo de geração de um arquivo “APK” para a publicação do aplicativo. O arquivo “APK” (Android Application Pack) pode ser compreendido como uma extensão do Google capaz de proporcionar que usuários utilizem sua versão Alpha de uma aplicação sem que ela esteja inserida no Google Play, em outras palavras, é uma caixinha com todas as informações dos recursos necessários para instalar o aplicativo.

a) Criando um arquivo *“APK”* para publicação na “*Google Play Store”*

Como passo inicial, deve-se entender que essa geração de arquivo “APK”, objetiva a publicação no “*Google Play Store*”. Levando em consideração este pressuposto. é necessário gerar esse tipo de arquivo, pois, o local onde a aplicação será publicada exige chaves com assinaturas geradas pelo próprio programa para que se consiga efetivar o *upload* para o devido destino. O primeiro passo para gerar essa chave e clicar no botão *“Build”* e, após selecionar a opção *“Generate Signed Bundle”,* como tal mostrado no print de tela de computador na Figura 26, esses comandos encaminhará para o local onde será criada a chave de assinatura.

Figura 26 – Inicio da criação do arquivo *“APK”*

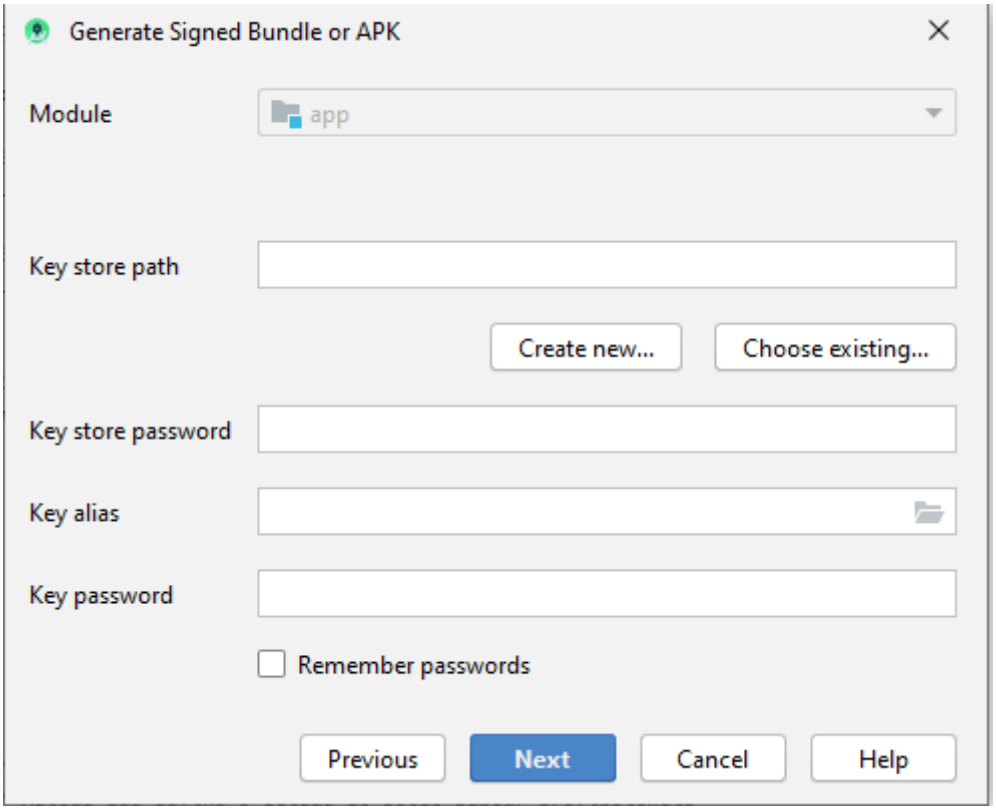


Fonte: Elaborada pelos autores

b) Criando uma nova chave

Na Figura 27, é apresentado o responsável por começar a criar a nova chave de assinatura. Ao clicar no botão *"Create New*" ele redimensionara a outra tela onde serão inseridos o local onde será salvo o arquivo gerado.

Figura 27 – Criando nova chave do arquivo *"APK*"



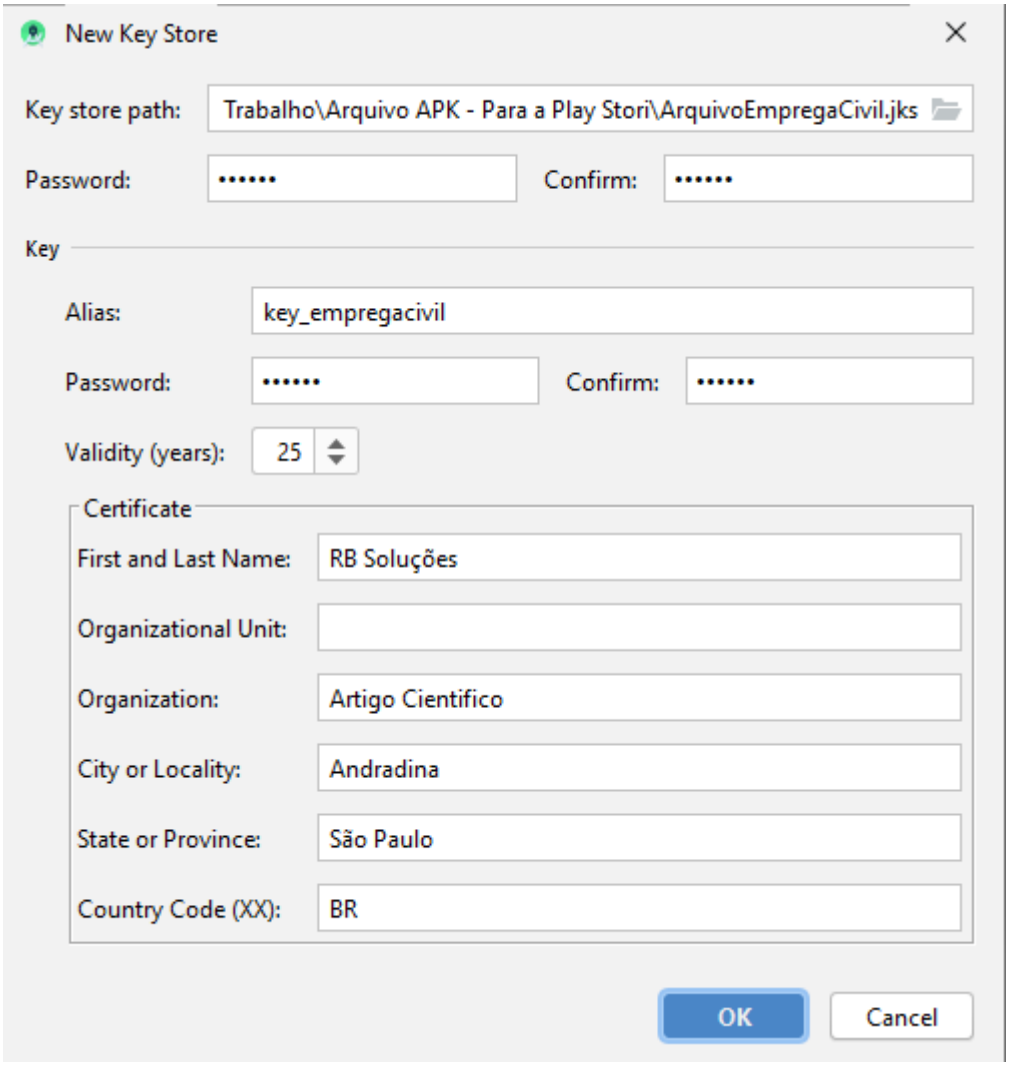
Fonte: Elaborada pelos autores

c) Inserindo informações das chaves

Neste passo (Figura 28), após ter dado início a criação da chave, no campo "*Key store path"* é inserido o local do software onde se deseja salvar as chaves criadas, juntamente ao campo "*Password"* onde se insere a senha para que a chave seja mantida de forma segura. Na seção *"Key"* devemos inserir um nome pra a chave, juntamente com uma senha. Essa será a chave que usaremos para fazer o *upload* da aplicação no "*Google Play Store*" e a mesma também permitirá que façamos atualizações na aplicação, essa chave dispõe de uma extrema importância, pois, se a mesma for perdida, ao gerar uma nova chave para a aplicação, o "*Google Play Store"*

fará uma comparação das certificações, ou seja, das assinatura das chaves, e por apresentarem divergências ela não permitirá que seja feita uma nova atualização da aplicação. É possível observar que a chave tem um período de duração de 25 anos, pode-se constatar esse padrão no campo *"Validity".*

Figura 28 – Atribuição de complementos da criação da chave do arquivo *"APK*"



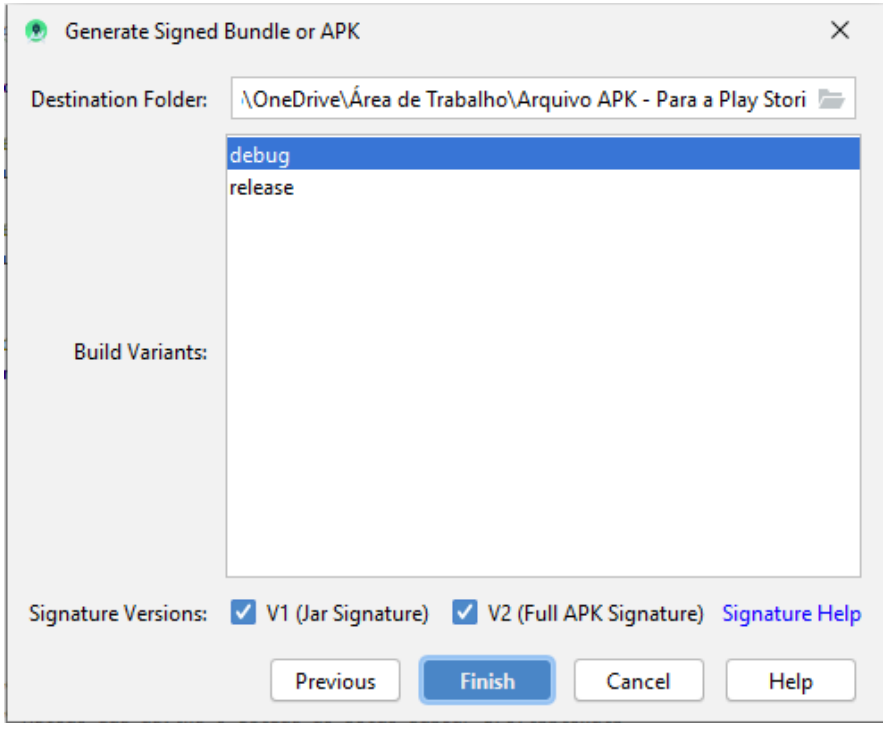
Fonte: Elaborada pelos autores

Na seção *"Certificate"* devem ser inseridas informações sobre a chave, como a identificação (nome) dos desenvolvedores, o endereço onde está sendo desenvolvido, e a organização funcional da chave. Após inserir os dados da autoria da chave, clica-se no botão "ok" para prosseguir com a próxima etapa.

d) Escolhendo gerador de arquivo

Após a execução do passo anterior aparecerá um botão *"Next"* e ao clica-lo será redirecionado a este passo (Figura 29), onde deve-se escolher entre gerar um arquivo de "*Debug*" ou "*Release"*, o primeiro tem a função de depurar o código da aplicação, ele contém mais informações, e o segundo tem a função de liberação. No caso da aplicação usaremos o *"Debug"* por permitir que utilizemos a aplicação antes do lançamento para que, caso houver um erro ser possível ser resolvido antes da publicação.

Figura 29 – Escolhendo gerador de arquivo e versões de assinatura.



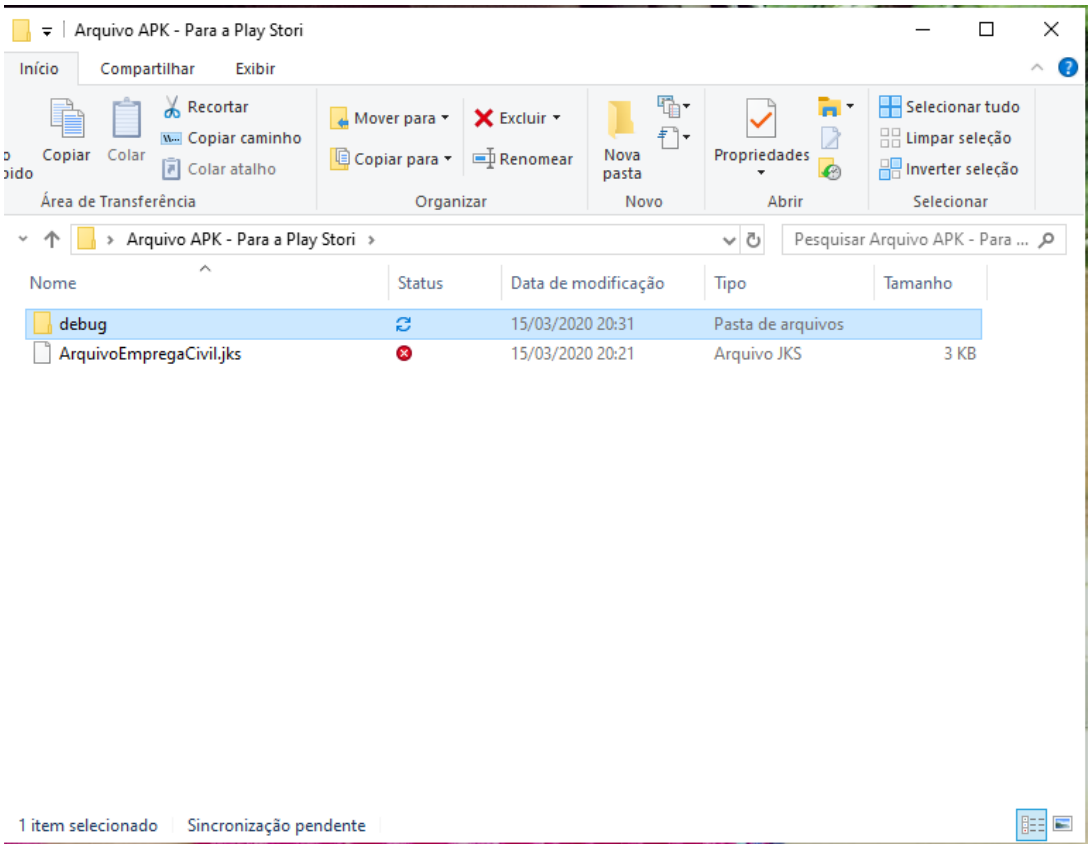
Fonte: Elaborada pelos autores

No campo *"Signature Versions"* aparecerá as opções de versões de assinatura do gerador *"APK*", no caso da aplicação que está sendo desenvolvida, foi desejável gerar o arquivo nas duas versões. Após esse processo clica-se no botão "*Finish"* para que seja gerado o arquivo *"APK".*

e) Pronto para *“Upload”*

Nessa etapa (Figura 30), o arquivo “APK” foi gerado, e está pronto para que seja feito o “Upload” da aplicação do “Google Play Store”. Pode-se por assim dizer que todo o processo da publicação da aplicação fora concluído.

Figura 30 – Arquivo *“APK”* gerado.



Fonte: Elaborada pelos autores

4.2 Manual de utilização do “Emprega Civil”

O manual de utilização do aplicativo “Emprega Civil” foi desenvolvido para instruir os usuários quanto às funções para sua utilização. Na Figura 31 é apresentado o manual, escrito em linguagem informal e com uso de figuras das telas do aplicativo, a fim de viabilizar as instruções de uso, estreitando a relação entre o “Emprega Civil” e o usuário.

Figura 31 - Manual de utilização do aplicativo “Emprega Civil”

Fonte: Elaborada pelos autores

4.3 Testes no aplicativo “Emprega Civil”

A finalidade dos testes foi verificar a estabilidade do aplicativo perante o uso dos profissionais e empregadores e obter o *feedback* dos mesmos, avaliado a partir de quatro situações.

Em três questões, o nível de funcionalidade e aplicabilidade foi por meio da atribuição de um valor de 0 (zero) sendo “ruim” e 10 (dez) como ótimo, abordando os seguintes aspectos:

a) A facilidade da utilização do aplicativo;
b) A utilidade do aplicativo para o ramo da construção civil; e,
c) A utilização da tecnologia, atualmente, no setor da construção civil.

A quarta questão propunha, dentre quatro benefícios que o “Emprega Civil” pode proporcionar aos usuários, a escolha da que mais foi representativa para quem estava testando o aplicativo, a citar:

a) A divulgação de profissionais (Alternativa A);
b) A oportunidade de prestação de serviços (Alternativa B);
c) A facilidade em encontrar mão de oba qualificada (Alternativa C); e,
d) A facilidade na cotação de prestadores de serviços em função da lista de profissionais (Alternativa D); .

Vale destacar que em todos os testes, nenhum usuário relatou problemas de funcionalidade do aplicativo e todos os cadastros relatados como executados pelos profissionais foram registrados no banco de dados com êxito.

Conforme havia sido proposto, o teste se dividiu em dois grupos de pessoas com dez em cada um, sendo que:

a) um grupo testou as funções de “Cadastro de Profissionais”; e,
b) o outro grupo testou as funções de “Busca de Profissionais”.

4.3.1 Primeiro grupo

As respostas dadas pelo primeiro grupo, que testou as funções do “Cadastro de Profissionais”, possibilitou que cada profissional fizesse o cadastro em uma profissão, dentre as dez fornecidas pelo aplicativo e, após o teste, respondeu ao questionário avaliativo. As respostas das questões de atribuição de notas são resumidas no Gráfico 1.

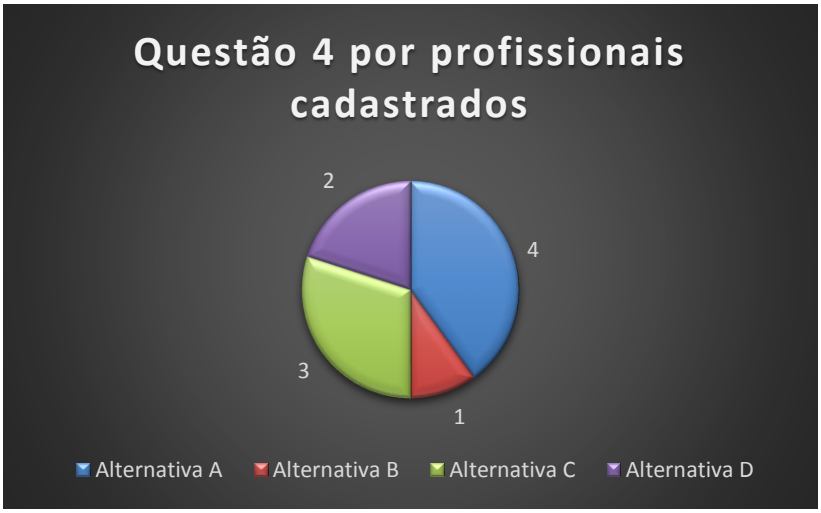
Gráfico 1 – Respostas do Grupo 1 – “Cadastro de Profissionais”

Fonte: Elaborada pelos autores

O Gráfico 1 demonstra que dos profissionais listados, para a questão 1 (facilidade de utilização do aplicativo) e questão 2 (utilidade do aplicativo para o ramo da construção civil), houve a predominância a média das notas 9. Já em relação à questão 3 (utilização da tecnologia, atualmente, no setor da construção civil) registrou média 7.

Considerando as respostas dos profissionais na quarta questão, quando escolheram um dentre os quatro benefícios listados, tem-se os resultados tal como expõe o Gráfico 2.

Gráfico 2 – Respostas do Grupo 1 sobre os benefícios do aplicativo



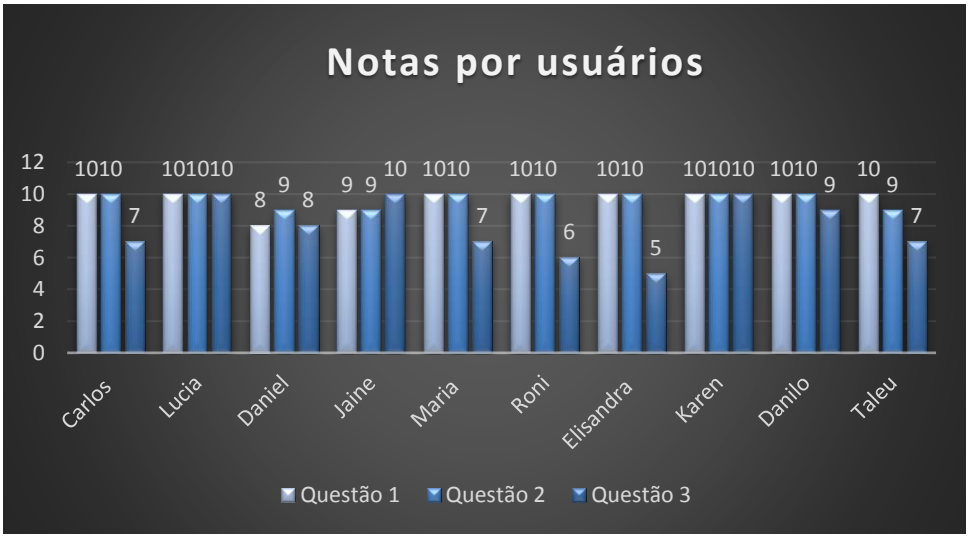
Fonte: Elaborada pelos autores

Observa-se, pelos dados do Gráfico 2, que os dez profissionais que participaram desta etapa de teste, concentraram as opiniões em apontar os benefícios de divulgação de profissionais (Alternativa A).

4.3.2 Segundo grupo

O segundo grupo, também composto por dez pessoas, testou as funções de “Busca de Profissionais”. Cada usuário realizou sua busca por profissionais cadastrados anteriormente e, a partir da sua utilização, atribuiu notas, as quais constam como resultado, expostas no Gráfico 3.

Gráfico 3 – Respostas do Grupo 2 – “Busca de Profissionais”

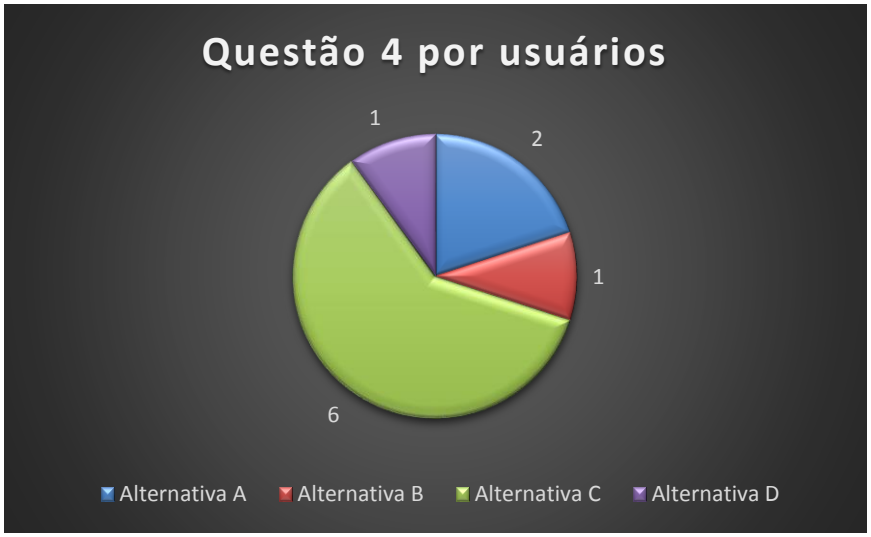


Fonte: Elaborada pelos autores

Os dados do Gráfico 3 demonstram que as pessoas que participaram desta etapa do teste demonstraram em relação à questão 1 e 2 (a facilidade da utilização do aplicativo e a utilidade do aplicativo para o ramo da construção civil, respectivamente), uma indicação de nota 9,7.

Este grupo também respondeu à quarta questão escolhendo um dos benefícios listados na mesma: alternativa A - divulgação de profissionais; alternativa B - oportunidade de prestação de serviços; alternativa C - facilidade em encontrar mão de oba qualificada; alternativa D - facilidade na cotação de prestadores de serviços em função da lista de profissionais. Os resultados possibilitaram a organização do Gráfico 4.

Gráfico 4 – Respostas sobre os benefícios do aplicativo



Fonte: Elaborada pelos autores

Recebido o f*eedback* dos dois grupos, foi possível perceber que a avaliação a respeito da utilização de todas as funções do “Emprega Civil” foi oportuna e positiva. Assim, possibilitou a organização dos resultados no Gráfico 5, com a média das respostas dadas às questões 1, 2 e 3, que testaram o cadastramento de profissionais e os de busca, atribuindo notas de 0 a 10.

Gráfico 5 – Média das respostas atribuídas pelos grupos de participantes

Fonte: Elaborada pelos autores

Analisando o gráfico pode-se comprovar que, por meio de avaliações, o objetivo de proporcionar facilidade na utilização do aplicativo foi atingido, registrando a média

de 9,5 nas respostas de utilização. Assim, é possível atestar que o aplicativo oferece uma dinâmica fácil e objetiva, o que desde o início foi tido como objetivo principal para o sucesso do “Emprega Civil”.

Totalizando com nota 9,45, pode-se comprovar que o intuito principal do aplicativo, quanto à utilidade no setor da construção civil, é significativo, o que beneficia todos os profissionais do setor, além de beneficiar também os que os procuram.

De acordo com a avaliação dos profissionais e usuários que utilizaram o aplicativo, o setor da construção civil ainda carece de tecnologia, com uma nota de 7,45 é possível registrar que o setor ainda necessita de avanços e iniciativas tecnológicas para alavancar o setor da construção civil.

Foi desenvolvido um gráfico que expõe a média da questão número quatro do questionário avaliativo, cujos resultados estão expostos no Gráfico 6, os quais representam quais benefícios os grupos julgam que o “Emprega Civil” trará para o setor da construção civil.

Gráfico 6 – Média das respostas da questão quatro

Fonte: Elaborada pelos autores

Por fim, por meio de respostas dos utilizadores que realizaram os testes no Emprega civil, os mesmos julgaram que o principal benefício que o aplicativo pode trazer é a facilidade em encontrar mão de obra qualificada (com 9 indicações nas avaliações, na alternativa C) e como benefício secundário a divulgação de profissionais do setor (com 6 indicações nas avaliações, na alternativa A).

CONSIDERAÇÕES FINAIS

O aplicativo para cadastramento e busca de profissionais do setor da construção civil, Emprega Civil, foi desenvolvido a fim de criar uma vitrine virtual de profissionais para facilitar a relação de profissionais e empregadores em uma plataforma de fácil manipulação.

Concluiu-se que o aplicativo é funcional, atestando seu êxito atingindo o objetivo proposto, apesar de esbarrar em questões burocráticas, as quais não permitiram seu uso comercial. O Emprega Civil teve sua funcionalidade constatada em teste fechado direcionado aos profissionais do mesmo segmento do público-alvo de aplicação.

Na questão da identificação das implicações na busca de um profissional do setor da construção civil, conclui-se que o Emprega Civil se mostrou eficiente, pois a forma como o cadastro foi disponibilizado obedece a uma estrutura simples e objetiva com informações básicas tanto da atuação como do contato com o mesmo.

Portanto, tanto nas funções de cadastramento de profissionais como na busca, conclui-se que o aplicativo está apto a ser colocado no mercado, em uso aberto. Destaca-se ainda que algumas melhorias ainda serão realizadas em trabalhos futuros, em busca de aumentar a capacidade de exposição de informações dos profissionais e disponibilização dessas informações para melhor escolha dos empregadores.

REFERÊNCIAS

AMADO, G. **As dez áreas com mais vagas no *Linkedin* em 2020**. Revista Época. 2020. Disponível em: https://epoca.globo.com/guilherme-amado/as-dez-areas-com-mais-vagas-no-linkedin-em-2020-24278385. Acesso em: 29 fev. 2020

BEKAERT, B. **A tecnologia na construção civil e seus desdobramentos**. 2018. Disponível em: < https://blog.belgobekaert.com.br/a-tecnologia-na-construcao-civil-e-seus-desdobramentos/> Acesso em 02 nov.2019

CÂMARA BRASILEIRA DE INDÚSTRIA DA CONSTRUÇÃO (CBIC). **Construção Civil perdeu 440 mil ocupações no primeiro trimestre do ano**. Disponível em: https://cbic.org.br/construcao-civil-perdeu-440-mil-ocupacoes-no-primeiro-trimestre-do-ano/. Acesso em: 1 mai. 2020

CATHO. **Vantagens de utilizar a Catho**. Disponível em: <cadastro.catho.com.br.> Acesso em 23 mai. 2020

ELMASRI, Ramez; NAVATHE, Shamkant B. Sistemas de Banco de Dados: 4. ed. Brasil: Pearson Brasil, ano 2005. 744 p.

HARVEY M. DEITEL (ed.). Java™: 6. ed. Brasil: Prentice-hall, 2005.1110 p.

SINTES, Anthony. Aprenda Programação Orientada a Objeto em 21 Dias. Brasil: Makron Books, 2002. 718 p.