JOÃO PEDRO TEODORO SILVA

RELATÓRIO DE ESTÁGIO - DESENVOLVIMENTO ANDROID NA EMPRESA IOASYS

LAVRAS – MG

2020

JOÃO PEDRO TEODORO SILVA

RELATÓRIO DE ESTÁGIO - DESENVOLVIMENTO ANDROID NA EMPRESA IOASYS

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para obtenção do título de Bacharel.

Prof. Dr. Paulo Afonso Parreira Júnior

Orientador

LAVRAS - MG

2020

JOÃO PEDRO TEODORO SILVA

RELATÓRIO DE ESTÁGIO - DESENVOLVIMENTO ANDROID NA EMPRESA IOASYS
INTERNSHIP REPORT - ANDROID DEVELOPMENT IN IOASYS

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para obtenção do título de Bacharel.

APROVADA em 02 de setembro de 2020

Prof. Dr. Heitor Augustus Xavier Costa UFLA

Bel. Diego Adenir Ferreira de Paula Carvalho IOASYS

Prof. Dr. Paulo Afonso Parreira Júnior

Orientador

LAVRAS - MG

2020

Dedico este trabalho aos meus pais, família e amigos que estiveram comigo durante o trajeto da graduação.

AGRADECIMENTOS

Agradeço aos meus pais que, desde o início, me apoiaram e me incentivaram a continuar no percurso da graduação.

Agradeço aos meus amigos que sempre estiveram comigo e tornaram a caminhada mais suportável e divertida.

Agradeço à Universidade Federal de Lavras por proporcionar os recursos necessários para que a graduação fosse possível, além de oferecer moradia estudantil e bolsa de extensão, o que foi indispensável para a minha permanência no curso.

RESUMO

Pretendeu-se, neste trabalho, apresentar as principais atividades realizadas no estágio desenvolvido na empresa ioasys, pontuando os métodos utilizados pela empresa e as tecnologias usadas para desenvolvimento. Para melhor compreensão do trabalho, definiu-se alguns conceitos importantes sobre a tecnologia Android e o framework Scrum, utilizados durante o estágio, apontando onde ou em quais situações estes conceitos foram aplicados. Constituíram também objetivos do trabalho, indicar os conhecimentos adquiridos, bem como as dificuldades encontradas em cada atividade realizada e exemplificar o resultado das atividades. Como resultado da experiência, concluiu-se que o estágio possui um grande valor, que contribuiu para a vida profissional e para a relação interpessoal do estagiário.

Palavras-chave: Desenvolvimento Android. Estágio.

ABSTRACT

In this work, it was intended to present the main activities carried out in the internship developed in the company ioasys, punctuating the methods used by the company and the technologies used for development. For a better understanding of the work, it was defined some important concepts about Android technology and the Scrum framework, used during the internship, pointing out where or in what situations these concepts were applied. The objectives of this work were also to indicate the knowledge acquired, as well as the difficulties encountered in each activity performed and exemplify the result of the activities. As a result of the experience it was concluded that the internship has great value, which contributed to the intern's professional life and interpersonal skills.

Keywords: Android development. Internship

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo, apresenta-se a contextualização do trabalho, uma breve descrição da empresa **ioasys** e uma introdução das atividades realizadas durante o estágio.

1.1 Contextualização

Com o crescimento de uma empresa ao longo do tempo, fica cada vez mais difícil manter a comunicação interna, o engajamento corporativo e a transmissão da cultura da empresa para seus colaboradores. Neste trabalho, apresenta-se as atividades realizadas durante o estágio na empresa **ioasys** (*Innovation Oasys* Desenvolvimento de Sistemas LTDA), mais especificamente em um projeto interno, denominado Noz, que visa minimizar os problemas citados anteriormente.

1.2 Sobre a ioasys e o projeto Noz

O estágio em uma empresa é um grande avanço para o graduando, pois, estando no ambiente de trabalho, o aluno pode aplicar os conhecimentos teóricos adquiridos em diversas disciplinas durante a graduação, preparando-se melhor para o mercado de trabalho. Na **ioasys**, cada estagiário possui um mentor técnico e um mentor cultural, os quais o auxiliam no seu desenvolvimento técnico e pessoal.

A **ioasys** é uma empresa belorizontina, criada em 2012, que oferece soluções digitais a diversas empresas. Possui mais de 100 colaboradores, com sede em Belo Horizonte e escritórios em Lavras, Aracajú, São Paulo e Londres. Seu diferencial é promover a adaptação de empresas tradicionais aos novos cenários de transformação digital, seguindo os princípios de metodologia ágil e abordagem de desenvolvimento centrado no usuário.

Na **ioasys,** há diversos projetos internos, como é o caso do projeto Noz. Noz é uma plataforma de engajamento corporativo, comunicação interna e cultura organizacional, que auxilia nos processos de Departamento Pessoal da empresa.

Um exemplo de funcionalidade desta plataforma é a possibilidade de criar publicações, tais como avisos, notícias, recomendações e enquetes, e compartilhá-las com seus colaboradores.

Além da ioasys, outras empresas são clientes do produto. O projeto conta com aplicativos nativos para Android e iOS e é customizável de acordo com o cliente. Cada cliente possui um “time” no projeto e, para cada time, são definidas cores personalizadas e aplicadas em componentes do aplicativo. O estagiário trabalhou, especificamente, na equipe responsável pelo desenvolvimento Android.

1.3 Objetivos

O projeto Noz passou por várias mudanças em seu aspecto visual, além de novas funções que foram implementadas. Neste estágio supervisionado, o objetivo foi o desenvolvimento de novas funções no aplicativo Android, bem como a manutenção de funções já existentes.

Dentre as funções implementadas, estão: (i) telas de aniversariantes; (ii) tela de detalhes da publicação, na qual é apresentado todo o conteúdo da publicação; e (iii) enquetes no mural de publicações. Dentre as manutenções realizadas no aplicativo, destacam-se: (i) adição de um controle na tela de configurações para permitir que vídeos iniciem automaticamente no mural; (ii) melhorias na visualização da tela de perfil do usuário; e (iii) adição de uma animação para recarregamento das publicações. Mais detalhes sobre essas e outras atividades encontram-se no Capítulo 3 deste trabalho.

1.4 Estrutura do trabalho

Este trabalho está organizado da seguinte forma: no Capítulo 2, são apresentados alguns conceitos básicos sobre as tecnologias e os métodos adotados durante o estágio; no Capítulo 3, são descritas as principais atividades realizadas pelo autor do presente relatório durante o estágio, bem como as dificuldades encontradas e os conhecimentos adquiridos; e, no Capítulo 4, são apresentadas as considerações finais deste trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo, apresenta-se, sucintamente, as principais tecnologias, metodologias e linguagens utilizadas durante a realização do estágio.

2.1 Scrum

Os conceitos apresentados são baseados no trabalho de Cruz (2018). Scrum é um *framework* para projetos ágeis, utilizado para o gerenciamento e o desenvolvimento de produtos. As equipes Scrum são compostas por três papéis básicos: *Scrum Master, Product Owner* e *Scrum Team*.

O papel do ***Scrum Master*** é garantir que o Scrum seja entendido e aplicado, fazendo com que o *Scrum Team* atente-se para as práticas e valores do Scrum. Quanto ao ***Product Owner***, este é responsável pelo entendimento do negócio do produto e pela entrega de valor ao cliente. Ele é a pessoa que gerencia e prioriza as tarefas realizadas pelo *Scrum Team*. O ***Scrum Team*,** comumente chamado de time de desenvolvimento, é responsável pelo desenvolvimento da funcionalidade do produto. Tendo as definições da funcionalidade do produto, o time se organiza e decide as melhores tecnologias, padrões e outras definições técnicas para desenvolver o produto de forma incremental e com qualidade.

As características e a funcionalidade do produto são descritas e disponibilizadas em um artefato, denominado ***Product Backlog***, que carrega o entendimento necessário para atender os requisitos.

O Scrum possui quatro eventos formais, descritos a seguir: planejamento do *Sprint,* reunião diária, revisão do *Sprint* e retrospectiva do *Sprint. Sprint* é um evento de duração fixa e pré-estabelecida. Idealmente, um *Sprint* deve durar um mês ou menos e, em um *Sprint*, deve-se ter metas pré-estabelecidas, com objetivos claros para, no final, gerar valor para o cliente. O Planejamento do *Sprint* é realizado para definir o que e como será feito durante o *Sprint*. A reunião diária, também chamada *daily meeting*, é uma reunião de, no máximo, 15 (quinze) minutos, que ocorre todos os dias no mesmo horário, na qual o time de desenvolvimento sincroniza suas atividades e planeja o próximo dia de trabalho.

Na revisão do *Sprint* (*Sprint Review*), o objetivo é a revisão do produto por parte do *Product Owner*, ou do cliente, em todos os itens concluídos pelo time durante o *Sprint.* Nessa revisão, é possível conferir e avaliar as atividades consideradas prontas, levando em consideração o que está sendo entregue e o que deveria ser entregue.

Na retrospectiva do *Sprint,* o objetivo é inspecionar como ocorreu o último *Sprint* em relação às pessoas (a relação entre elas) e aos métodos e ferramentas utilizados. Nessa retrospectiva, são analisados os itens que deram certo e devem permanecer no próximo *Sprint*, os itens que precisam ser melhorados e os itens que devem ser descartados.

O Scrum foi aplicado durante todo o estágio na **ioasys**. No projeto Noz, cada *Sprint* teve duração de 2 (duas) semanas. Como parte do time de desenvolvimento, o autor do presente relatório atuou como desenvolvedor Android.

2.2 Tecnologias utilizadas

Nesta seção, apresenta-se as tecnologias e as linguagens utilizadas no desenvolvimento do projeto Noz, durante o estágio na empresa **ioasys.**

2.2.1 Kotlin

Para o desenvolvimento Android, são utilizadas as linguagens Java e/ou Kotlin. De acordo com Lecheta (2017), Kotlin é uma linguagem de programação desenvolvida pela JetBrains, mesma empresa que criou o Android Studio[1]. Kotlin possui sintaxe simples e é compilada para executar na JVM (Java *Virtual Machine*), portanto tem total interoperabilidade com o ambiente de execução da plataforma Java.

Uma das vantagens do Kotlin é a sintaxe moderna e expressiva, pois, se comparada ao Java, pode-se escrever o mesmo código em menos linhas, como mostram os exemplos nas Listagens 1 e 2. Entretanto, por ser mais verboso que Kotlin, o Java torna-se mais legível.

[1] Android Studio é um ambiente de desenvolvimento integrado para desenvolvimento de aplicativos para a plataforma Android.

Listagem 1 - Trecho de código em Java

```
1  public class Person {
2      private String name;
3      private int age;
4
5      public Person(String name, int age) {
6          this.name = name;
7          this.age = age;
8      }
9
10     public String getName() {
11         return name;
12     }
13
14     public void setName(String name) {
15         this.name = name;
16     }
17
18     public int getAge() {
19         return age;
20     }
21
22     public void setAge(int age) {
23         this.age = age;
24     }
25 }
```

Fonte: Autor

Listagem 2 - Trecho de código em Kotlin

```
1 data class Person(var name: String, var age: Int)
```

Fonte: Autor

Na Listagem 1, apresenta-se uma classe feita em Java, a qual representa uma pessoa no mundo real. Na linha 1, a classe é declarada com o nome "*Person*". A classe possui dois atributos, declarados nas linhas 2 e 3, o atributo "*name*" do tipo *`String`* (uma sequência de caracteres), representando o nome da pessoa, e o atributo "*age*" do tipo *`int`* (um número inteiro), representando a idade da pessoa. Ambos os atributos são privados, devido ao uso do modificador de acesso *`private`*

no início da linha, ou seja, apenas a classe *Person* possui acesso a eles.

Na linha 5, é declarado o construtor da classe, o qual recebe dois parâmetros, o nome e a idade da pessoa. O modificador de acesso *public* indica que esse construtor é público e, assim, outras classes podem usá-lo para criar instâncias (objetos) da classe *Person.* As linhas 6 e 7 atribuem os valores passados por parâmetro às variáveis de instância da classe.

Nas linhas 10 e 18 são declarados métodos que retornam o nome e a idade da pessoa, respectivamente. Nas linhas 14 e 22, são declarados métodos para mudarem o valor do nome e da idade da pessoa, respectivamente.

Uma peculiaridade do kotlin é que existe uma forma de omitir os métodos acessores e os métodos modificadores para esse tipo de classe, cujo objetivo é reter dados. Para isso é necessário usar a anotação *data* na declaração da classe, conforme apresentado na Listagem 2. A declaração da classe possui a palavra-chave *data class*, seguida de seu identificador e de seus atributos. Esses atributos são os parâmetros do construtor da classe. Essa classe possui, implicitamente, todos os métodos mostrados na Listagem 1. Nota-se ainda que a declaração dos atributos é feita de uma forma diferente. Tem-se a palavra-chave *var*, que indica que é um atributo variável, cujo valor pode ser alterado (caso a variável seja imutável, usa-se a palavra-chave *val*).

Neste caso, para outra classe Kotlin ter acesso ao atributo de uma classe, supondo que um objeto *person* do tipo *Person* esteja instanciado, basta usar "`person.age`" para obter o valor da idade e "`person.age = age`" para alterar o seu valor. Em uma classe Java, para se obter o valor da idade, seria necessário usar "`person.getAge()`".

No projeto Noz, a maioria das classes antigas foram escritas em Java, portanto, a manutenção dessas classes é um dos motivos que exige o conhecimento da linguagem. As novas classes, entretanto, foram escritas em Kotlin, por isso, o estagiário precisou conhecer também esse novo tipo de tecnologia de programação.

2.2.2 Android SDK

De acordo com Cordeiro (2014), o Android SDK (*Software Development Kit* -

Kit de Desenvolvimento de *Software*) permite a criação de aplicativos para a plataforma Android, de forma nativa. Inclui-se no Android SDK: projetos de exemplo com código-fonte, ferramentas de desenvolvimento, emuladores e bibliotecas necessárias para a criação dos aplicativos. Embora as ferramentas do SDK possam ser usadas via linha de comando, o método mais comum de uso é por meio de um ambiente de desenvolvimento integrado (*Integrated Development Environment* - IDE), sendo o Android Studio o mais recomendado, por ser o ambiente oficial de desenvolvimento.

Uma das ferramentas presentes no Android SDK é o SDK *Tools*. O **SDK *Tools*** já é instalado com o pacote inicial do Android SDK e atualizado periodicamente. As ferramentas mais importantes incluem (CORDEIRO, 2014): o **Android SDK *Manager***, que permite gerenciar os pacotes do SDK, como as plataformas instaladas, por exemplo; o **AVD *Manager***, que fornece uma interface gráfica que permite criar e gerenciar dispositivos virtuais; o **Android *Emulator***, uma ferramenta de emulação de dispositivo Android; e o ***Dalvik Debug Monitor Server***, que permite depurar os aplicativos do Android de forma integrada ao Android Studio.

Por ser essencialmente necessário para o desenvolvimento Android, o Android SDK foi utilizado, juntamente com o Android Studio, no desenvolvimento das atividades do estágio em questão.

2.2.2.1 *Views* e *layout*

No desenvolvimento Android, tem-se como ênfase a criação de interfaces gráficas para dispositivos móveis, portanto, é importante entender alguns conceitos relacionados, tais como gerenciadores de *layout* e *widgets*. Os conceitos apresentados são baseados no trabalho de Lecheta (2017).

A classe `View` é a classe-mãe de todos os componentes visuais do Android SDK e suas diversas subclasses são utilizadas para criar elementos de interfaces gráficas do aplicativo. Existem dois tipos de *views*: os chamados *widgets* e os gerenciadores de *layout.* Um *widget* é um componente visual simples, que herda diretamente da classe `View`. Alguns exemplos de *widgets* são `Button, ImageView` e `TextView`, usados para criar botões, imagens e textos, respectivamente. Em geral,

esses componentes são utilizados (declarados) em arquivos XML (*EXtensible Markup Language*), conforme exemplo da Listagem 3.

Além dos *widgets* disponíveis pelo Android SDK, o desenvolvedor tem a possibilidade de criar os próprios *widgets*, criando uma classe que estende um *widget* ou gerenciador de *layout* existente e alterando seus atributos, conforme a necessidade. Esses *widgets* são comumente chamados de *custom view*, ou *view* customizada.

Um gerenciador de *layout* é um componente utilizado para organizar a disposição dos elementos gráficos na tela. Há diversos tipos de gerenciadores de *layout* no Android SDK. Alguns podem organizar os elementos na horizontal ou vertical, outros em forma de tabela. Todos os gerenciadores de *layout* são herdados da classe `ViewGroup`.

A linha 1 da Listagem 3 apresenta a tag `LinearLayout`, que corresponde a um gerenciador de *layout* que dispõe elementos na tela de forma linear, podendo ser de forma horizontal ou vertical, escolhidos por meio do atributo `orientation` (linha 5).

Listagem 3 - Trecho de código em XML

```
1     <LinearLayout
2        xmlns:android="http://schemas.android.com/apk/res/android"
3         android:layout_width="match_parent"
4         android:layout_height="wrap_content"
5         android:orientation="vertical">
6         <TextView
7             android:layout_width="wrap_content"
8             android:layout_height="wrap_content"
9             android:text="Texto 1" />
10
11        <TextView
12            android:layout_width="wrap_content"
13            android:layout_height="wrap_content"
14            android:text="Texto 2" />
15    </LinearLayout>
```

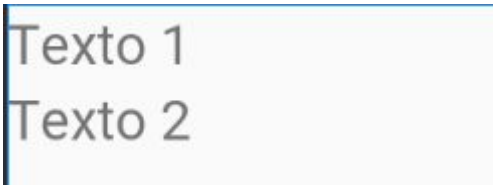
Fonte: Autor

Os atributos `layout_width` e `layout_height`, presentes nas linhas 3 e 4,

configuram a largura e altura do componente na tela, respectivamente. O valor *match_parent* indica que o componente deve ocupar toda a área disponível pelo componente superior a ele e o valor *wrap_content* indica que o componente deve ocupar o tamanho mínimo necessário.

Na linha 6, tem-se a tag *TextView*, um *widget* usado para criar texto na tela. Nota-se que a altura e largura ocupados na tela são “o mínimo necessário”. Na linha 9, é atribuído o valor “Texto 1” no atributo *text* (este atributo define o que será escrito na tela). O mesmo ocorre nas linhas 11 à 14, mudando apenas o valor do texto. Na linha 15, tem-se a *tag* de fechamento do *LinearLayout*. Como os dois textos estão dentro desse gerenciador de *layout*, eles estarão alinhados verticalmente, como é apresentado na Figura 1.

Figura 1 - Resultado visual do código da Listagem 3



Fonte : Autor

No desenvolvimento Android, é essencial o entendimento e uso correto desses componentes, bem como o conhecimento dos gerenciadores de *layouts* disponíveis.

2.2.2.2 Activity

No contexto de desenvolvimento de um aplicativo Android, um conceito importante é o de *activity*. *Activity*, segundo Lecheta (2017), é uma classe que herda da classe *android.app.Activity* ou de alguma de suas subclasses. Ela representa uma tela da aplicação e é responsável por tratar os eventos gerados pelo usuário.

A classe que representa uma *Activity* deve sobrescrever o método *onCreate(bundle: Bundle?)*, como mostra a linha 2 da Listagem 4. Esse método

é obrigatório e é responsável por realizar a inicialização necessária para executar a aplicação, por exemplo definir a interface de usuário. Esse método recebe como parâmetro um objeto `Bundle?` (a sintaxe da interrogação indica que o objeto pode ser nulo - esse é um recurso presente na linguagem Kotlin), que pode conter parâmetros enviados para a `activity`. Cada `activity` deve ser, obrigatoriamente, declarada no arquivo `AndroidManifest.xml`. Isso é feito por meio da *tag* `<activity>`, como apresentado na Listagem 5. O código da Listagem 5 indica que existe uma `activity`, cuja classe correspondente é `MainActivity`.

De acordo com Pereira (2009), cada `activity` possui um ciclo de vida específico. Esse ciclo de vida é representado por métodos chamados durante a existência da `activity`. São eles:

1. `onCreate`: chamado quando a *activity* é inicialmente criada;
2. `onStart`: chamado quando a *activity* torna-se visível;
3. `onResume`: chamado quando inicia-se a interação com o usuário;
4. `onPause`: chamado quando o sistema está perto de iniciar uma nova *activity*;
5. `onStop`: chamado quando a activity não estiver mais sendo utilizada pelo usuário e perdeu o foco para outra *activity*;
6. `onDestroy`: pode ser chamado quando a *activity* terminou ou quando o sistema finaliza para liberação de recursos;
7. `onRestart`: chamado quando a *activity* estiver interrompida e prestes a ser acionada novamente.

A classe `MainActivity` é representada na Listagem 4. Na linha 1, tem-se a declaração da classe que herda a classe `AppCompatActivity` (em kotlin, herança entre classes é indicada pelo caractere “dois pontos”), uma subclasse da classe `Activity`. A linha 3 chama o método `onCreate` da classe pai, por meio da palavra-chave `super`. Na linha 4, tem-se o método `setContentView`, que define qual interface será usada no contexto desta `activity`.

2.2.2.3 *Shared Preferences*

No desenvolvimento de aplicações Android, é comum salvar dados de configurações no aplicativo para definir preferências do usuário (GLAUBER, 2014).

Para isso, existe uma classe denominada *SharedPreferences*, que salva informações no formato “chave/valor”. Esses valores são armazenados em um arquivo XML, dentro de uma estrutura interna do aplicativo.

Listagem 4 - Trecho de código em uma classe filha da *Activity*

```
1 class MainActivity : AppCompatActivity() {
2     override fun onCreate(savedInstanceState: Bundle?) {
3         super.onCreate(savedInstanceState)
4         setContentView(R.layout.activity_main)
5     }
6 }
```

Fonte: Autor

Listagem 5 - Trecho de código no *AndroidManifest*

```
1  <activity android: name = ".MainActivity"/>
```

Fonte: Autor

Supondo que um aplicativo tenha dois temas de cores de interface gráfica, um escuro e um claro, e o usuário possa alternar entre os temas por meio de uma configuração dentro do aplicativo. Para que o usuário não tenha que selecionar o tema de sua preferência toda vez que abrir o aplicativo, é possível salvar a escolha do usuário com a classe *SharedPreferences*. Para obter uma instância dessa classe, basta chamar o método *getSharedPreferences*, passando como parâmetro o nome da chave e o modo de acesso à informação, conforme mostra a Listagem 6. O modo *MODE_PRIVATE* indica que somente o aplicativo em questão possui acesso aos valores armazenados.

Listagem 6 - Trecho de código que instancia uma *SharedPreferences*

```
1 SharedPreferences sharedPreferences =
2       getSharedPreferences("configuração", Context.MODE_PRIVATE);
```

Fonte: Autor

Para salvar uma preferência, usa-se um *Editor*, provido pela classe *SharedPreferences*, e o método *edit()*, como mostra a linha 1 da Listagem 7. Com o objeto da classe *Editor*, é possível salvar uma preferência, usando métodos correspondentes ao tipo de valor a ser armazenado. Se o valor a ser armazenado for um inteiro, usa-se o método *putInt*, caso seja uma *String*, usa-se *putString*. Em todos os casos, deve-se passar a chave e o valor como parâmetros para esses métodos. Na linha 2 da Listagem 7, é criada a chave "tema" e o valor "escuro" é atribuído a ela (caso a chave "tema" exista, o valor é atualizado). Na linha 3 da Listagem 7, a mudança é realmente aplicada.

Listagem 7 - Trecho de código que salva um valor pela *SharedPreferences*

```
1 SharedPreferences.Editor editor = sharedPreferences.edit();
2 editor.putString("tema", "escuro");
3 editor.apply();
```

Fonte: Autor

Por fim, para recuperar um valor armazenado com a classe *SharedPreference*, basta usar os métodos disponíveis, de acordo com o tipo do valor a ser lido. Como foi salva uma *String* relacionada com a chave "tema" na Listagem 7, usa-se o método *getString* para recuperar este valor, como mostra a Listagem 8. É preciso passar dois parâmetros para este método: uma chave, neste exemplo é usada a chave "tema", e um valor padrão para o caso em que não se encontre essa chave salva.

Listagem 8 - Trecho de código que recupera um valor da *SharedPreference*

```
1 String result = sharedPreferences.getString("tema", "");
```

Fonte: Autor

3 ATIVIDADES REALIZADAS

Neste capítulo, apresenta-se as atividades realizadas durante o período de estágio. São descritas as atividades desenvolvidas no projeto Noz, bem como as dificuldades encontradas e conhecimentos adquiridos no processo. Por motivos de sigilo, alguns trechos de código não puderam ser apresentados neste documento.

3.1 Período de Adaptação

Nesta seção, apresenta-se algumas das atividades realizadas antes da atuação direta no projeto Noz.

3.1.1 ioasys Camp

A empresa ioasys iniciou um projeto no segundo semestre de 2019, denominado **ioasys camp.** Tratava-se de uma etapa do programa de estágio, na qual os candidatos poderiam entrar em contato com os métodos de desenvolvimento e tecnologias utilizadas pela empresa. Os conteúdos eram transmitidos de forma teórica e a prática era realizada com um desafio baseado em problemas reais. Os membros que se destacavam neste processo poderiam ser chamados para trabalhar na empresa.

Nessa etapa, que antecedeu o estágio, foi iniciado um projeto para que os instrutores pudessem passar os conteúdos e colocar em prática alguns conceitos. Nas aulas sobre tecnologia Android, usou-se um projeto denominado “Empresas”, fazendo com que o aplicativo do projeto evoluísse a cada etapa do treinamento.

3.1.2 Início do Estágio

Após ser chamado para realizar o estágio, o estagiário continuou a desenvolver o aplicativo “Empresas”. Por meio desse aplicativo, foram exercitadas boas práticas de programação, seguindo princípios de orientação a objetos, além de colocar em prática os conhecimentos adquiridos por meio de cursos relacionados à tecnologia Android.

Os estudos foram relacionados a *views* customizadas, tratamento de erros, comunicação entre classes, retenção de dados no aplicativo, padrões arquiteturais, padrões de projetos, entre outros assuntos imprescindíveis para a inserção do estagiário no projeto Noz.

Após os estudos, o estagiário aprendeu os procedimentos para realizar suas atividades e encaminhá-las para o repositório de código do projeto. A empresa faz uso do Bitbucket[2] para hospedar seus repositórios e realizar o controle de versões dos projetos. Ao realizar uma atividade, era necessário disponibilizar o código para que outro desenvolvedor Android pudesse revisá-lo. Caso fosse aprovado, o estagiário atualizava o repositório com suas novas atividades prontas. Os desenvolvedores que analisam os códigos têm a liberdade de comentar em alguma parte do código que eles julgam ser necessário uma mudança ou que pode ser melhorado.

3.2 Regras de Negócio

Nesta seção, algumas regras de negócio relacionadas ao projeto Noz são apresentadas, para sua melhor compreensão: i) na tela de perfil do usuário, os campos de e-mail e telefone devem ser preenchidos com texto da cor do time; ii) no botão de interação de publicações, o número máximo de fotos de perfis deve ser 4; iii) as fotos devem ser dos primeiros usuários que curtiram a publicação; iv) o usuário recebe moedas ao interagir com o aplicativo ou eventos externos definidos pelos usuários com acesso de administrador; v) as moedas recebidas podem ser trocadas por produtos dentro do aplicativo; vi) os produtos são definidos e disponibilizados pelos usuários com acesso de administrador.

3.3 Atividade de ajustes e melhorias realizadas no Projeto Noz

Nesta seção, apresenta-se algumas atividades referentes às correções e às melhorias do aplicativo Noz, realizadas durante o estágio.

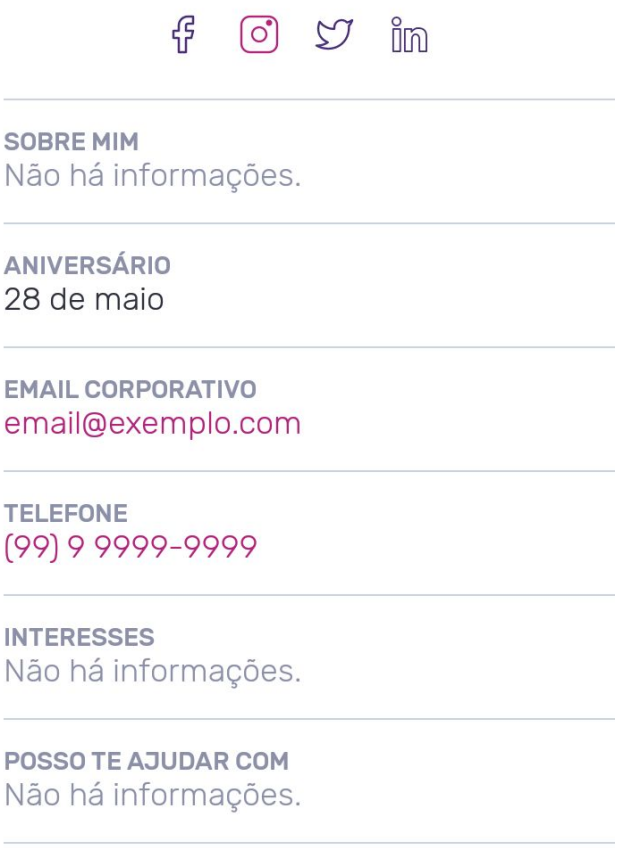
[2] Bitbucket é um serviço de hospedagem de projetos controlados com o Mercurial, um sistema de controle de versões distribuído.

3.3.1 Texto padrão em tela de perfil

A primeira atividade recebida para implementação no período do estágio foi colocar um texto de dica em campos vazios na tela de perfil do usuário. No aplicativo, o usuário possui alguns dados pessoais e dados de contato. Portanto, há uma tela de perfil, na qual são apresentados esses dados e há a possibilidade de editá-los.

Entretanto, é possível que alguns desses dados não estejam preenchidos. Para este cenário, foi decidido colocar o texto “Não há informações” nesses campos, com a cor do texto diferenciada, dando menos destaque a eles, conforme mostra a Figura 2.

Figura 2 - Tela de Perfil do usuário no aplicativo Noz



Fonte: **ioasys** (2020).

Nota-se que os campos “email corporativo” e “telefone” são preenchidos com uma cor diferente das demais; esta cor é a cor do time, de acordo com as customizações realizadas para o cliente em questão (conforme explicado na Seção 1.2, cada time tem uma cor personalizada, a pedido do cliente).

A Listagem 9 apresenta o método usado para definir a cor do texto. Para cada campo da tela apresentada na Figura 2 (exceto e-mail e telefone), é chamado esse método, no qual é verificado se o texto que deve ser apresentado na tela é vazio ou nulo (linha 4). Na linha 6, é selecionado o texto padrão para ser apresentado na tela.

Listagem 9 - Trecho de código que define a cor do texto na tela de perfil

```
1  private void handleTextAndColorTextView(String text,
2                          ObservableField<Integer> fieldColor,
3                          ObservableField<String> userTextField){
4      if (text == null || text.isEmpty()) {
5          fieldColor.set(R.color.blue_grey);
6          userTextField.set((getDefaultText()));
7      } else {
8          fieldColor.set(R.color.charcoal_grey_two);
9          userTextField.set(text);
10     }
11 }
```

Fonte: **ioasys** (2020).

Com essa atividade, foi possível fazer uso de boas práticas de programação, criando métodos que removeram códigos duplicados e facilitaram a manutenção e escalabilidade do código.

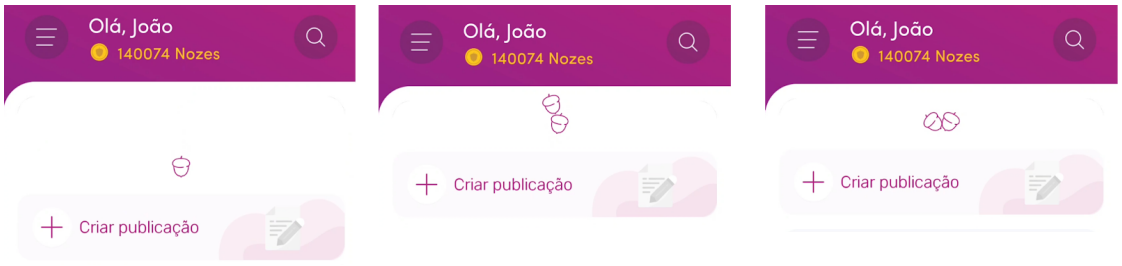
3.3.2 Ajustes no mural de publicações

Outra atividade realizada pelo estagiário foi a colocar uma animação de carregamento personalizada na tela principal do aplicativo, o Mural de Publicações. Esse mural é composto por publicações, um *menu* lateral e uma barra de ferramentas (*ToolBar*), que apresenta o nome do usuário, a quantidade de moedas que ele possui na plataforma e um botão de pesquisa. Quando o usuário está no topo do mural e faz o movimento de arraste para baixo na tela, tem-se o movimento

de atualização (*refresh*) da tela, realizando o recarregamento dos conteúdos do mural.

O *designer* do projeto desenvolveu uma animação para ser colocada acima das publicações, com o intuito de indicar que as publicações estão sendo carregadas. A animação apresenta duas nozes, uma das nozes aparece caindo sobre a outra e, ao se chocar contra a noz que estava embaixo, a de cima cai para o lado e começa a animação novamente. A Figura 3 representa a animação em três tempos diferentes: o lado esquerdo representa a animação no momento em que o usuário arrasta o mural para baixo, o meio representa o momento em que a noz, vinda de cima, choca-se com a outra noz, e o lado direito apresenta o final da animação, com as duas nozes caídas.

Figura 3 - Representação da animação de carregamento do mural de publicações



Fonte: **ioasys** (2020).

Além dessa animação, foi implementada uma função que identifica que o usuário chegou no final das publicações e faz uma nova requisição para buscar novas publicações. Cada requisição feita ao servidor onde os conteúdos ficam armazenados retornam 10 (dez) publicações. A maior dificuldade, nesse caso, foi identificar quando o usuário chegou na última publicação para fazer uma nova requisição. Com essa atividade, obteve-se mais maturidade com a manipulação de listas em aplicações Android.

3.3.3 *Autoplay* nos vídeos do mural de publicações

Uma das melhorias do aplicativo feitas durante o estágio foi a possibilidade de habilitar a iniciação automática dos vídeos no mural. Para isso, foi adicionada, na tela de configurações, uma chave para o usuário habilitar ou não essa função (Figura 4). Caso o usuário a desabilite, aparece um botão para iniciar o vídeo no mural. Se o usuário a deixar habilitada, o botão não aparece e os vídeos são executados automaticamente ao aparecerem na tela para o usuário. Uma vez que o vídeo iniciar, não é possível pausá-lo, a menos que o usuário saia da tela. Nesse caso, o vídeo para de ser executado e, ao voltar a ele, ele é reiniciado.

Figura 4 - Opção de Auto play na tela de configuração

Fonte: **ioasys** (2020).

Nesta tela de configurações, tem-se a lógica apresentada na Listagem 10. Há um *widget* (componente) chamado *switch,* representado pela variável `switchAutoplay`. Esse componente possui o método apresentado na linha 1 da Listagem 10, que tem como segundo parâmetro o valor da variável `isChecked`, o qual diz se o componente está ativado ou desativado. Esse método é chamado toda vez que o componente muda de estado (ligado/desligado). Portanto, ao ser chamado, esse método pode alterar o valor controlado pela `sharedPreferences` com método `setValue`, como mostra na linha 2 da Listagem 10. Nesse caso, a

chave é definida como "*autoplay_active*" e o valor atribuído a ela é o mesmo vindo do *switch*. O código da Listagem 10 foi alterado em relação ao original por motivos de sigilo e conveniência para melhor entendimento do leitor.

Na classe que define se o vídeo será ou não executado automaticamente, usa-se o método *getBoolean*, por se tratar de uma variável do tipo *boolean*, para definir o comportamento do vídeo, conforme mostra a Listagem 11. Na linha 1, é verificado o valor armazenado pela *sharedPrefences* com a chave "*autoplay_active"*. Se o valor for *true*, o botão para iniciar o vídeo, representado pela variável *playButton*, é escondido por meio do método *hide* (linha 2). Na linha 3, é indicado que o vídeo será executado assim que ele for carregado na tela. As linhas 5 e 6 fazem exatamente o oposto das linhas 2 e 3, respectivamente.

Listagem 10 - Trecho de código que altera o valor do auto play

```
1 switchAutoplay.setOnCheckedChangeListener((buttonView,
  isChecked) -> {
2     mySPreferencesManager.setValue("autoplay_active",isChecked);
3 });
```

Fonte: **ioasys** (2020)

Listagem 11 - Trecho de código que recupera o valor do auto play

```
1 if (mySPreferencesManager.getBoolean("autoplay_active")) {
2     playButton.hide();
3     player.setPlayWhenReady(true)
4 } else {
5     playButton.show();
6     player.setPlayWhenReady(false)
7 }
```

Fonte: **ioasys** (2020)

Com essa atividade, foi possível praticar o uso da classe *sharedPreferences*.

3.4 Descrição das atividades de novas telas e componentes

Nesta seção, algumas das atividades referentes às novas telas e funcionalidades do aplicativo Noz, implementadas durante o estágio, são apresentadas.

3.4.1 Tela de detalhes de publicação

Ao clicar em uma publicação no mural de publicações, o usuário é redirecionado para a tela de detalhes dessa publicação. Essa tela apresenta uma capa com o título, nome e foto do autor e, se o primeiro conteúdo da publicação for uma imagem, carrossel de imagens ou vídeo, este é apresentado também na capa. Abaixo da capa, são apresentados os conteúdos da publicação, seguido de um botão de interação, por meio do qual o usuário pode curtir/descurtir ou comentar. Abaixo do botão de interação, são apresentadas as *tags* da publicação. Essas *tags* são usadas para definir quais usuários poderão visualizar a publicação. Por exemplo, se somente a *tag* “RH” é selecionada para uma publicação, apenas as pessoas que possuem a *tag* “RH” poderão visualizar a publicação.

Uma versão dessa tela existia anteriormente, contudo, pelo fato de grande mudança no *layout* ser necessária, ela foi re-desenvolvida pelo autor deste relatório. Nos detalhes da publicação, pode haver conteúdos diferentes e, para cada tipo, foi desenvolvido um *layout* diferente. Os tipos de conteúdo implementados pelo autor foram: texto, imagem, carrossel de imagens, vídeo e *hyperlink*.

Na Listagem 12, apresenta-se a parte do código responsável por selecionar o *layout* a ser adicionado na listagem de conteúdos de acordo com o tipo de conteúdo. Na linha 2, tem-se a sobrescrita do método `onCreateViewHolder`, responsável por inflar o *layout* de um item de uma lista; este método é chamado para cada item da lista. Na linha 6, verifica-se qual é o tipo do item dessa lista. Da linha 7 até a 28 é selecionado o *layout* e a classe `ViewHolder`, de acordo com o tipo do item. Na classe `ViewHolder`, retém-se as informações da `View`. Na linha 29, retorna-se à classe `ViewHolder` selecionada.

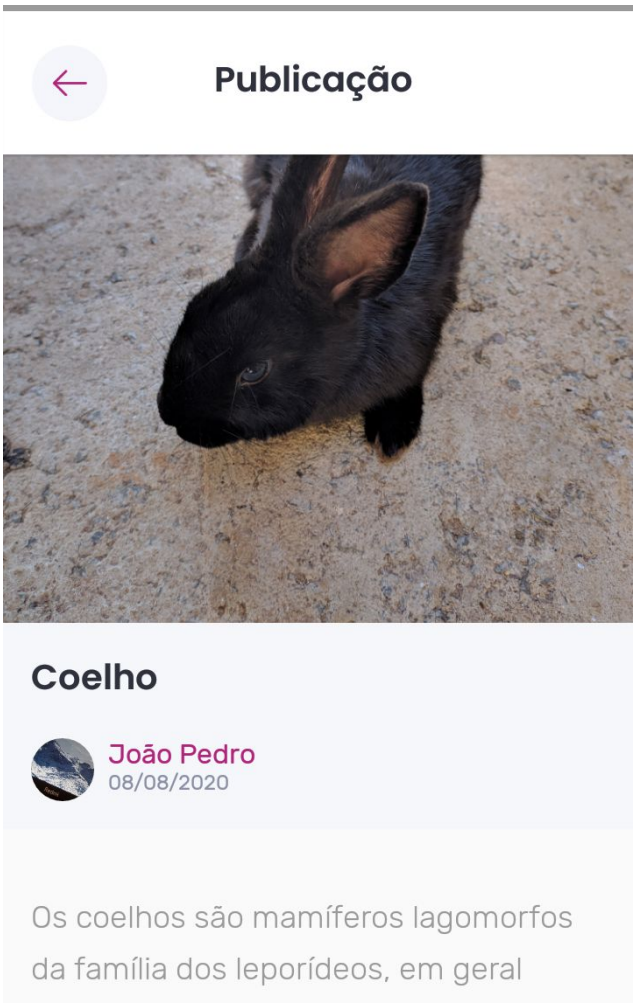
Na Figura 5, apresenta-se o topo da tela de detalhes de publicação, onde se tem uma imagem como capa e um conteúdo do tipo texto. Abaixo da imagem são apresentados o título da publicação, nome e imagem de perfil do autor da publicação e a data de publicação. Essa publicação foi feita pelo autor deste trabalho, exclusivamente para exemplo de *layout*, em ambiente de homologação.

Listagem 12 - Trecho de código que seleciona o *Layout* para conteúdo de publicação

```
1  @Override
2  public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup
3   parent, int viewType) {
4     RecyclerView.ViewHolder viewHolder;
5    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
6     PubType type = PubType.getByValue(viewType);
7    switch (type) {
8        case IMAGE:
9            View imageContent =
   inflater.inflate(R.layout.item_content_image, parent, false);
10           viewHolder = new ContentImageViewHolder(imageContent);
11           break;
12       case TEXT:
13           View textContent =
   inflater.inflate(R.layout.item_content_text, parent, false);
14           viewHolder = new ContentTextViewHolder(textContent);
15           break;
16       case LINK:
17           View linkContent =
    inflater.inflate(R.layout.item_content_link, parent, false);
18           viewHolder = new ContentLinkViewHolder(linkContent);
19           break;
20       case VIDEO:
21          View videoContent =
    inflater.inflate(R.layout.item_content_video, parent, false);
22           viewHolder = new ContentVideoViewHolder(videoContent);
23           break;
24       case CAROUSEL:
25           View carouselContent =
    inflater.inflate(R.layout.item_content_carousel, parent, false);
26           viewHolder = new ContentCarouselViewHolder(carouselContent,
    context);
27           break;
28    }
29   return viewHolder;
30 }
```

Fonte: **ioasys** (2020)

Figura 5 - Representação da tela de detalhes de publicação



Fonte: **ioasys** (2020)

Com o desenvolvimento dessa tela, a maior dificuldade foi a implementação de uma lista cujos itens possuem *layouts* diferentes. Devido a isso, houve um enriquecido conhecimento sobre a construção desse tipo de lista, em aplicativos Android. Foi possível, também, compreender melhor o funcionamento de uma `activity`, principalmente em relação ao seu ciclo de vida e aos eventos emitidos nas mudanças de seu estado.

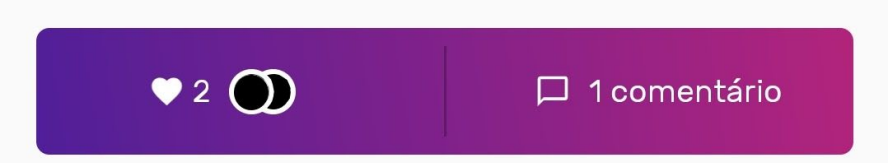
3.4.2 Botão de interação

Como foi dito na Seção 3.3.1, na tela de detalhes da publicação, há um botão de interação, por meio do qual o usuário pode curtir/descurtir e comentar. A criação deste botão foi dividida em 5 partes: criar o `layout` principal, criar a área da interface na qual o usuário clica para comentar, criar uma `View` customizada para apresentar as fotos de até quatro usuários que tenham curtido a publicação, criar os botões de curtir e descurtir e fazer as animações, de acordo com o estado em que o botão se encontra.

Dividiu-se o espaço do *layout* principal igualmente entre os botões de curtir e comentar, e o fundo desse *layout* é um degradê com as cores do time (personalizada de acordo com o pedido do cliente) - Figura 6. No botão de comentar, apresenta-se a quantidade de comentários e, ao clicar sobre ele, o usuário é redirecionado para a tela de comentários. A `view` customizada aparece quando o usuário curte a publicação, na qual podem aparecer de 1 a 4 fotos de perfil. Sempre terá, pelo menos, uma foto, pois, para aparecer essa `view`, o usuário precisa curtir a publicação. O botão de curtir possui um ícone de coração não-preenchido e a palavra “curtir”, quando o usuário ainda não curtiu a publicação; caso tenha curtido, mostra-se o coração preenchido de branco e o número de curtidas.

A animação do botão de curtir acontece da seguinte forma: ao clicar em curtir, o ícone é trocado, dando impressão de que o coração foi preenchido, o texto esmaece até desaparecer, o coração é arrastado para a esquerda e o número de curtidas e a `view` aparecem, gradualmente.

Figura 6 - Botão de interação após usuário curtir a publicação



Fonte: **ioasys** (2020).

A *view* customizada é, resumidamente, apresentada na Listagem 13. Na linha 1, é declarada a classe que estende um gerenciador de *layout*, denominado *ConstraintLayout*, este gerenciador de layout organiza os componentes de forma flexível, usando outros componentes da tela como referência. Das linhas 3 a 6, são declaradas as variáveis que representam as quatro imagens de perfil. Os construtores dessa classe chamam o método apresentado na linha 8. Esse método, por sua vez, infla o *layout* dessa *view* (linhas 9 e 10) e, entre as linhas 11 e 14, atribui os valores para as variáveis da classe, buscando os componentes presentes no *layout* inflado. O método apresentado na linha 18 é responsável por receber a URL de uma imagem e carregá-la na primeira imagem do componente. Para cada uma das quatro imagens, há um método semelhante a esse. A classe *GlideUtils*, apresentada na linha 19, faz o trabalho de carregar a imagem no componente *ImageView*. Essa classe é declarada estática no projeto e usa recursos do Android para carregar imagens por meio de URL, em componentes como *ImageView.*

Listagem 13 - Trecho de código em kotlin da *view* customizada

```
1 class ProfilesPicturesCustomView : ConstraintLayout {
2
3   private lateinit var pictureOne: ImageView
4   private lateinit var pictureTwo: ImageView
5   private lateinit var pictureThree: ImageView
6   private lateinit var pictureFour: ImageView
7   [...]
8   private fun initView(attrs: AttributeSet? = null) {
9       val view = View.inflate(context,
10                    R.layout.profiles_pictures_custom_view, this)
11
12       pictureOne = view.findViewById(R.id.pictureOneImageView)
13       pictureTwo = view.findViewById(R.id.pictureTwoImageView)
14       pictureThree = view.findViewById(R.id.pictThreeImageView)
15       pictureFour = view.findViewById(R.id.pictFourImageView)
16   }
17
18   fun setPicOne(url: String) {
19       GlideUtils.loadCircularImage(url, pictureOne)
20   }
[...]
```

Fonte: **ioasys** (2020)

Com essa atividade, o autor do presente relatório pôde aprender a implementar uma *view* customizada e a lidar com animações. A maior dificuldade foi desenvolver as animações, principalmente o arraste para a esquerda. A dificuldade estava em achar o ponto em que o ícone devia parar, pois o número de curtidas possui um tamanho variável.

3.4.3 Enquetes no mural de publicações

Com o objetivo de fazer rápidas pesquisas de opinião dentro da empresa, foram criadas publicações de enquete com respostas únicas ou múltiplas. Ao responder uma enquete, é mostrado um resultado parcial da pesquisa. Além disso, cada pesquisa possui uma data limite para ser respondida, definida pelo criador da pesquisa.

As opções de respostas dessas enquetes são construídas programaticamente (Listagem 14). Ao selecionar uma alternativa, aparece um botão para o usuário enviar sua resposta e, ao clicar sobre esse botão, uma caixa de diálogo de confirmação é apresentada ao usuário. Ao confirmar, a resposta é enviada e o resultado parcial é apresentado na tela.

Listagem 14 - Trecho de código que adiciona as opções de resposta programaticamente

```
1  public void addRadiosToRadioGroup(
2      RadioGroup radioGroup,
3      List<SurveyModel> surveys) {
4
5      for (int i = 0; i < surveys.size(); i++) {
6          RadioButton radioButton = new RadioButton(context);
7          radioButton.setButtonTintList(getColorStateList());
8          radioButton.setText(surveys.get(i).getTitle()));
9          radioButton.setId(surveys.get(i).getId());
10         radioGroup.addView(radioButton);
11     }
12 }
```

Fonte: **ioasys** (2020)

Para as enquetes de resposta única, é chamado o método apresentado na Listagem 14, que recebe como parâmetro o componente *RadioGroup*, no qual serão inseridos, como filhos, os componentes clicáveis e uma lista de objetos da classe *SurveyModel*, que contém os dados das opções de resposta. Na linha 5, é realizada a iteração nesta lista. Para cada item da lista, na linha 6, é criado um componente *RadioButton* e, nas linhas 8 e 9, são atribuídos os valores a esse componente, de acordo com os dados da opção de resposta. Na linha 7, é alterada a cor do componente, de acordo com o seu estado (selecionado/não selecionado). Por fim, na linha 10, o componente é adicionado ao *RadioGroup*.

Na Figura 7, apresenta-se uma enquete com múltiplas escolhas, ao lado esquerdo, e uma enquete de escolha única, ao lado direito. Nota-se que a única diferença visual é o componente usado para selecionar as opções.

Figura 7 - Representação de enquetes de múltiplas escolhas e de resposta única

Fonte: **ioasys** (2020).

O maior desafio dessa atividade foi ter o controle de quando o usuário confirma o envio para ser mostrado o resultado parcial. Por estar em um componente da lista de conteúdos, o item de enquete não tem acesso à confirmação de envio. Foi necessário salvar os dados da enquete no componente

em que ela se encontra e criar um método para alterar o *layout* das opções para o *layout* de respostas. Esse método é chamado quando o usuário confirma a resposta.

3.4.4 Telas de aniversários

Como uma empresa possui vários colaboradores, o aplicativo possui dois fluxos nos quais é possível visualizar o perfil de outros usuários, sendo eles um fluxo de aniversários e um de times (cada time representa um setor da empresa).

O autor deste relatório de estágio foi responsável por construir o fluxo de aniversários, o qual possui 3 (três) telas: (i) uma que lista os meses, mostrando a quantidade de pessoas que fazem aniversários naquele mês e a foto de três aniversariantes, como mostra a Figura 8 - lado esquerdo; (ii) uma tela que lista os aniversariantes do mês, ordenados pelo dia do aniversário - Figura 8, lado direito; e (iii) uma tela que mostra os aniversariantes do dia.

Figura 8 - Telas do fluxo de Aniversários

Fonte: **ioasys** (2020).

Por meio do *menu* do aplicativo, é possível acessar a tela de aniversários listadas por meses. Ao clicar em um mês, o usuário é redirecionado para a tela de aniversariantes do mês, na qual o usuário pode clicar em um perfil para visualizá-lo com mais detalhes. O mesmo acontece se o usuário clicar sobre o *card* de aniversariantes do dia. Esse *card* de aniversariantes do dia, apresentado na Figura 9, também é apresentado no topo do mural de publicações. Dessa forma, o acesso aos aniversariante do dia é mais rápido e prático, além de informar para o usuário que há aniversariantes no dia, sem que ele precise ir até o fluxo de aniversário.

Figura 9 - Card de aniversariantes do dia

Fonte: **ioasys** (2020).

Os aniversariantes retornados pelo servidor estão ordenados pela ordem de nascimento. Assim, foi preciso reordenar os usuários pelo dia do aniversário, desconsiderando o ano de nascimento.

Com essa atividade, foi possível aprender a utilizar a interface *Comparator* da linguagem Java. Com essa interface é possível definir um método a ser utilizado para comparar dois objetos em uma ordenação.

4 CONSIDERAÇÕES FINAIS

O estágio realizado na **ioasys** foi de grande valia para a vida profissional do estagiário. Ele trouxe para o estagiário um enriquecido conhecimento técnico na tecnologia trabalhada, além de desenvolver a relação interpessoal e colocar em prática os conhecimentos adquiridos durante a graduação.

Várias disciplinas realizadas durante o curso de Ciência da Computação foram importantes para a realização do estágio. Disciplinas como Introdução aos Algoritmos, Estrutura de Dados e Prática de Programação Orientada a Objetos estimularam a habilidade de raciocinar e procurar soluções para os problemas existentes, bem como trabalhar com estruturas de dados de forma eficiente, o que foi essencial para o desenvolvimento das atividades no estágio.

A disciplina Programação Paralela e Concorrente contribuiu para o desenvolvimento de tarefas nas quais partes do código eram executadas de forma assíncrona. A disciplina Arquitetura de Software foi de grande relevância para a escolha de padrões de projetos em determinadas situações e a identificação de más práticas de programação.

Durante o estágio, foi trabalhada, também, a comunicação com outros desenvolvedores, o que contribuiu para o desenvolvimento da relação interpessoal do estagiário. A empresa possui um canal de comunicação, por meio do qual é possível entrar em contato com qualquer pessoa que faz parte dela. Com o grupo de desenvolvedores Android, foi possível tirar dúvidas e aprender com outros profissionais mais experientes.

Em suma, concluiu-se que o estágio é uma grande oportunidade para o graduando desenvolver-se como profissional e preparar-se para o mercado de trabalho. No estágio, enfrentam-se problemas desafiadores que obrigam o estagiário a desenvolver-se e a crescer, fazendo com que ele tenha experiência ao entrar no mercado de trabalho.

REFERÊNCIAS

CORDEIRO, Fillipe. **Android Sdk: O que é? Para que serve? Como usar?.** 2014. Disponível em: <https://www.androidpro.com.br/blog/android-studio/android-sdk/>. Acesso em: 10 de agosto de 2020.

CRUZ, Fábio. **Scrum e Agile em Projetos:** guia completo**.** 2ª ed. Brasport, 2018.

GLAUBER, Nelson. **Dominando o Android**: do básico ao avançado. Novatec Editora, 2014.

LECHETA, Ricardo R. **Android Essencial com Kotlin**. Novatec Editora, 2017.

PEREIRA, Lucio Camilo Oliva; DA SILVA, Michel Lourenço. **Android para desenvolvedores**. Brasport, 2009.