UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESTUDO SOBRE A UTILIZAÇÃO DO ANDROID JETPACK NO DESENVOLVIMENTO DE APLICATIVOS ANDROID

TRABALHO DE GRADUAÇÃO

Aluno: Raquel Maria Santos de Oliveira (rmso@cin.ufpe.br)
Orientador: Leopoldo Motta Teixeira (lmt@cin.ufpe.br)
Área: Engenharia de Software

Recife
Dezembro de 2018

RAQUEL MARIA SANTOS DE OLIVEIRA

Estudo sobre a utilização do Android Jetpack no desenvolvimento de aplicativos Android

Trabalho de Conclusão de Curso apresentado ao curso de Ciência da Computação da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Leopoldo Motta Teixeira

Recife
Dezembro de 2018

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RAQUEL MARIA SANTOS DE OLIVEIRA

Estudo sobre a utilização do Android Jetpack no desenvolvimento de Aplicativos Android

Aprovada em ___ / ___ / ____

_____

Leopoldo Motta Teixeira

_____

Sergio Castelo Branco Soares

Aos meus pais, Laudicéa e Enilton. Sem vocês eu não chegaria até aqui.

AGRADECIMENTOS

Por várias vezes pensei em desistir, muitas vezes cai, outras eu pensei que não iria conseguir, mas hoje estou aqui apenas para agradecer por essa estrada que foi longa e de muito aprendizado.

Agradeço primeiramente à Deus por ter traçado esse caminho da minha vida. Muitas vezes achei muito pesada a carga que levei nessa trajetória, mas eu sempre lembrava que Ele não dá mais do que a gente suporta. Sempre carrego comigo o seguinte trecho na música que me faz ver o mundo de uma forma melhor: *Minha força e vitória tem um nome, é Jesus.* Por isso, sou grata eternamente por essa etapa de minha vida.

Quero agradecer imensamente aos meus pais (minha Nega e meu Tito), não sei o que eu seria hoje sem eles em minha vida. Eles são a minha base e minha maior motivação de querer seguir em frente. Agradeço pelos momentos de sacrifício para dar um futuro melhor aos seus filhos, agradeço pela paciência nos momentos em que eu me estressava com as coisas da faculdade, pelo silêncio dos momentos em que precisei estudar e agradeço principalmente a minha mãe que todos os dias estava sempre disposta a escutar minhas histórias ao chegar em casa depois de um longo dia, mesmo eu falando demais, ela não reclamava e isso me fazia muito bem para seguir adiante. Esses momentos fizeram muita falta nos meus últimos períodos de graduação.

A minha família Santos por sempre demonstrar um exemplo do que é uma família unida. Em especial ao meu irmão por estar sempre torcendo por mim e que me mata de orgulho, ao meu Tio/Padrinho por estar sempre orgulhoso da sua sobrinha preferida e representar um segundo pai em minha vida e ao meu querido e amado primo por ser um grande amigo/irmão em todos os momentos.

Ao meu esposo Rodolfo, por ter paciência em alguns momentos em que eu precisava me isolar para fazer projetos e por me ajudar a vigiar as análises deste trabalho durante várias madrugadas em que estava sendo executado. Agradeço pela sua compreensão nos momentos que precisei.

Ao meu querido Paulo (in memoriam), que sempre torcia pelo meu sucesso e se hoje estivesse presente nesse momento, seria um dos primeiros a sentir orgulho de mim e ficar feliz por todas as etapas que conquistei.

A minha amiga e tia de coração, Ana Paula. Agradeço por todos os momentos que torceu junto comigo por minhas vitórias. Desde o resultado do vestibular até a conclusão da graduação.

Ao longo desses anos de graduação passei por diversas fases de alegria, tristeza, desespero e vários outros sentimentos que quem é do CIn sabe como é. Nesses longos anos, eu sempre tive ao meu lado as Minhas Girls (Késsia e Bárbara), eu agradeço imensamente por todos esses momentos que passamos juntas. Momentos de felicidade em tirar nota boa, tristeza por não ter sido aprovada na cadeira, vontade de querer desistir do curso e no fim não desistir, desespero para entregar projetos, união por estudar e conseguir enfrentar alguns obstáculos. Agradeço por elas estarem presentes e sempre me ajudando de uma forma inexplicável. Muito obrigada por tudo e também por se disponibilizarem a me ajudar na correção desta documentação.

Aos amigos que compartilharam comigo momentos de desespero e felicidade. Lucas Artur, Thayonara, Eduardo Henrique, Carolina Carneiro, Egberto, Ricardo Robson, Deyvson, Thiago Aquino e todos que fizeram parte dessa longa trajetória.

Ao meu orientador Leopoldo que me deu total suporte para desenvolvimento deste trabalho e teve uma grande paciência comigo. No início eu estava perdida em relação ao que fazer e ele me guiou com toda a sua tranquilidade e paciência.

Aos alunos do CIn que eu não conhecia e me ajudaram com algumas dúvidas relacionadas a este trabalho.

A todos os professores que passaram por essa minha vida acadêmica e se não fosse eles, eu não teria adquirido tanto conhecimento.

Aos meus amores de quatro patas que sempre me recebiam com maior alegria no momento que eu chegava muito cansada em casa, aos meus bebês: Sniff, Princesa e Chupeta (in memoriam).

Por fim, a todos os meus amigos e colegas que de forma direta ou indiretamente fizeram parte dessa minha fase.

Mainha e Painho, essa conquista é nossa!

RESUMO

Durante o desenvolvimento de um aplicativo é importante manter boas práticas e ter uma aplicação de alta qualidade com menos código. Este trabalho tem como objetivo fazer uma mineração de repositórios que fazem uso do Android Jetpack para investigar como está sendo o proveito no desenvolvimento de aplicações para a plataforma Android. Android Jetpack é um conjunto de componentes que facilita a criação de aplicativos de alta qualidade e menos código. Adicionalmente, através da análise dos repositórios retirados do catálogo do F-Droid e de repositórios GitHub em geral, espera-se extrair informações dos 1606 repositórios sobre a quantidade de projetos que utilizam a ferramenta no desenvolvimento de aplicativos Android e qual componente do Android Jetpack está sendo mais utilizado.

Palavras-chave: aplicativos Android, Android Jetpack, desenvolvimento de aplicações, mineração de repositórios.

ABSTRACT

During an application development it is important to keep good practices and have a high quality application with less code. This work objectives doing a repositories mining that uses Android Jetpack to investigate how it has been used in the Android platform applications development. Android Jetpack is a set of components that facilitates a high quality applications creation and less code. In addition, using the repositories analysis taken from the F-Droid catalog and GitHub repositories in general, it is expected to extract informations about how many projects are using the tool in Android application development and which Android Jetpack component has been used the most from the 1606 repositories.

Key words: Android applications, Android Jetpack, application development, mining repositories

SUMÁRIO

1. INTRODUÇÃO

O mercado de dispositivos móveis está crescendo continuamente. Estatísticas de 2018 afirmam que 64% dos brasileiros usam aplicativos móveis [2]. Portanto, aplicativos de alta qualidade são esperados por usuários e podem influenciar diretamente no sucesso da aplicação a longo prazo.

Android Jetpack facilita a criação de aplicativos para Android com uma coleção de componentes, ferramentas e orientações. Os componentes ajudam a seguir práticas recomendadas e facilitam no desenvolvimento de aplicativos de alta qualidade com menos código [3]. O Android Jetpack teve o lançamento no início de 2018, por isso é algo recente, e possivelmente está sendo pouco adotado por enquanto. De forma a entender como está o cenário de adoção das bibliotecas, este trabalho irá utilizar uma ferramenta capaz de minerar dados de repositórios. Esta ferramenta serve para ajudar os desenvolvedores a analisar repositórios e facilitar a extração de informações a partir dos artefatos disponíveis. Para fazer essas análises de repositórios e extrair as informações desejadas é utilizado o PyDriller [4].

No desenvolvimento de aplicações Android com a utilização do Android JetPack é necessária a modificação de alguns arquivos que contém informações sobre vários componentes. Com essas alterações, é possível desenvolver análises mais detalhadas sobre a sua utilização em aplicações, buscar informações de como está sendo esta adoção e a porcentagem de quantos aplicativos móveis estão utilizando as práticas recomendadas.

1.1. Objetivos

Este trabalho tem como objetivo investigar o uso de Android Jetpack no desenvolvimento de aplicações Android de código aberto. Para apurar essas informações, realizamos um estudo desde a sua configuração até a forma de uso no desenvolvimento. Desse modo, realizamos uma análise sobre a utilização do Android Jetpack para entender a prática e a estimativa

das aplicações de alta qualidade, esse conhecimento será procedido através da pesquisa de repositórios que faz a utilização da ferramenta.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. Android Jetpack

Com um conjunto de componentes, ferramentas e orientações, o Android Jetpack [5] tem o objetivo de facilitar a criação de aplicativos. Conforme mostrado na Figura 1, os componentes do Android Jetpack reúnem a biblioteca de suporte que é um grupo de componentes que facilitam o uso dos novos recursos do Android e o *Architecture Components* que foi projetado para facilitar o gerenciamento de dados durante as alterações do ciclo de vida dos aplicativos. Os componentes são organizados em quatro categorias:

- Architecture;
- UI;
- Behavior;
- Foundation.

Os componentes com Android Jetpack foram desenvolvidos para funcionar juntos, mas não é obrigatório o uso de todos em conjunto. A integração pode ser feita apenas com as partes do Android Jetpack que resolvam os problemas do projeto e manter as partes do aplicativo que já funcionam bem.

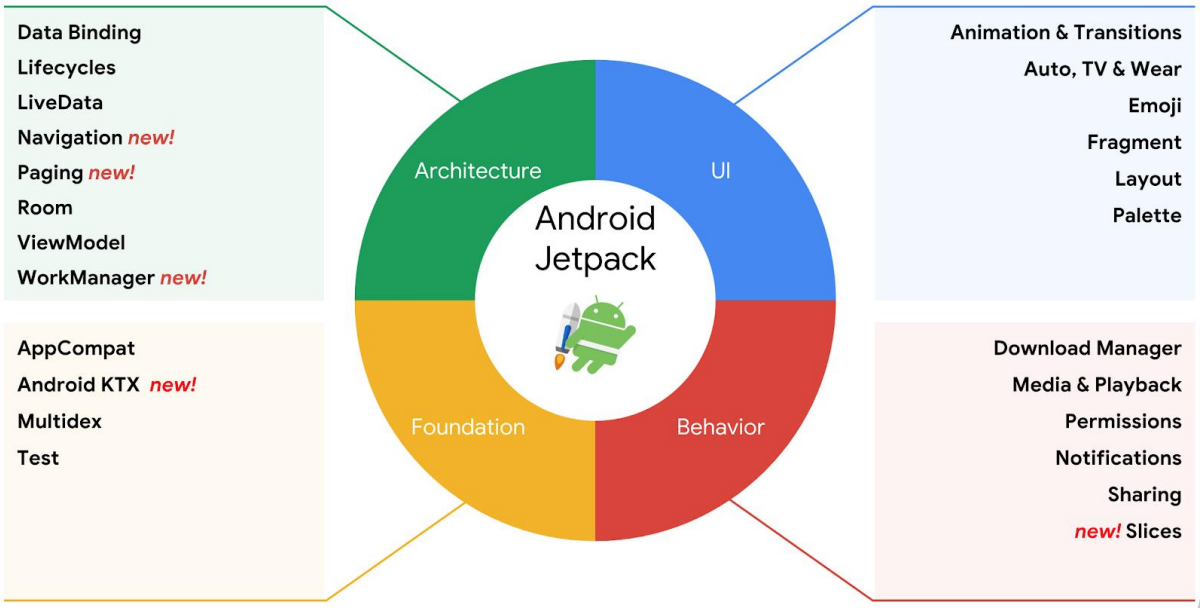

Figura 1 – Categorias do Android Jetpack

Fonte: Blog Google Developers

2.1.1. *Architecture Components*

É uma coleção de bibliotecas que ajudam a desenvolver aplicações robustas, testáveis e de fácil manutenção. Esta biblioteca ajuda a gerenciar o ciclo de vida dos componentes da UI e controlar melhor a persistência de dados mesmo com alterações do ciclo de vida da aplicação [3].

Nos próximos tópicos apresentamos algumas bibliotecas que fazem parte da *Architecture Components*, esses componentes foram a base da análise deste trabalho.

2.1.1.1. *Lifecycle*

É uma classe que contém as informações sobre o ciclo de vida de uma **`Activity`** ou **`Fragment`** nos termos de estados e eventos e permite que outros objetos observem esse estado [3]. Tem o objetivo de ajudar o gerenciamento dos componentes, sendo capaz de facilitar automaticamente a associação de comportamento com base no estado atual do seu ciclo de vida.

Quando acontece alguma alteração no ciclo de vida, seus eventos são chamados. Os eventos que podem acontecer são:

- **`on_create`** - quando o **`LifecycleOwner`** é criado;

- **`on_destroy`** - quando o **`LifecycleOwner`** é destruído;
- **`on_pause`** - quando o **`LifecycleOwner`** é pausado;
- **`on_resume`** - quando o **`LifecycleOwner`** é resumido;
- **`on_start`** - quando o **`LifecycleOwner`** é iniciado;
- **`on_stop`** - quando o **`LifecycleOwner`** é parado;
- **`on_any`** - chamado quando ocorre qualquer tipo de evento no ciclo de vida do **`LifecycleOwner`** [6].

Os estados ajudam a indicar onde se encontra o **`LifecycleOwner`** no seu ciclo de vida. Sempre que ocorre um evento existe também uma mudança de estado como mostra na Figura 2.

`LifecycleOwner` é uma interface que possui um método único no qual indica a existência de um **`Lifecycle`** na classe. Esse método é o **`getLifecycle()`** que deve ser implementado pela classe. Ela abstrai o uso de um **`Lifecycle`**, permitindo escrever componentes que funcione com eles [7]. Qualquer classe de uma aplicação personalizada pode implementar a **`LifecycleOwner`** interface.

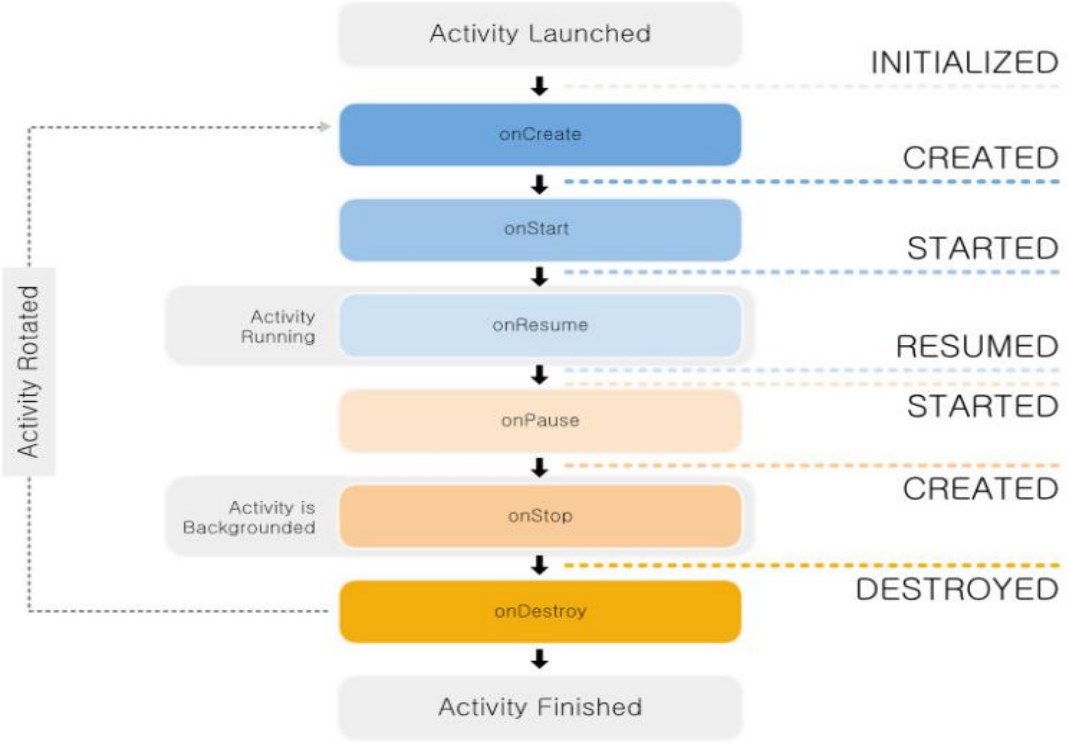

Figura 2 – Estados e eventos do Lifecycle

Fonte: Componentes da Arquitetura Android: Lifecycle e LiveModel

2.1.1.2. `LiveData`

É uma classe capaz de conter dados que são passados a ela para serem observados [9]. O **`LiveData`** conhece e obedece ao ciclo de vida dos componentes da aplicação fazendo com que atualize apenas os observadores de componentes que estão em um estado ativo do seu ciclo de vida. Dessa forma, o **`LiveData`** se torna diferente dos observadores comuns.

Como mostra a Figura 3, o **`LiveData`** faz a comunicação com o controlador da interface do usuário (UI) para atualizar a interface sempre que houver uma alteração de estado, em vez de ocorrer atualização toda vez que os dados da aplicação são modificados [10].

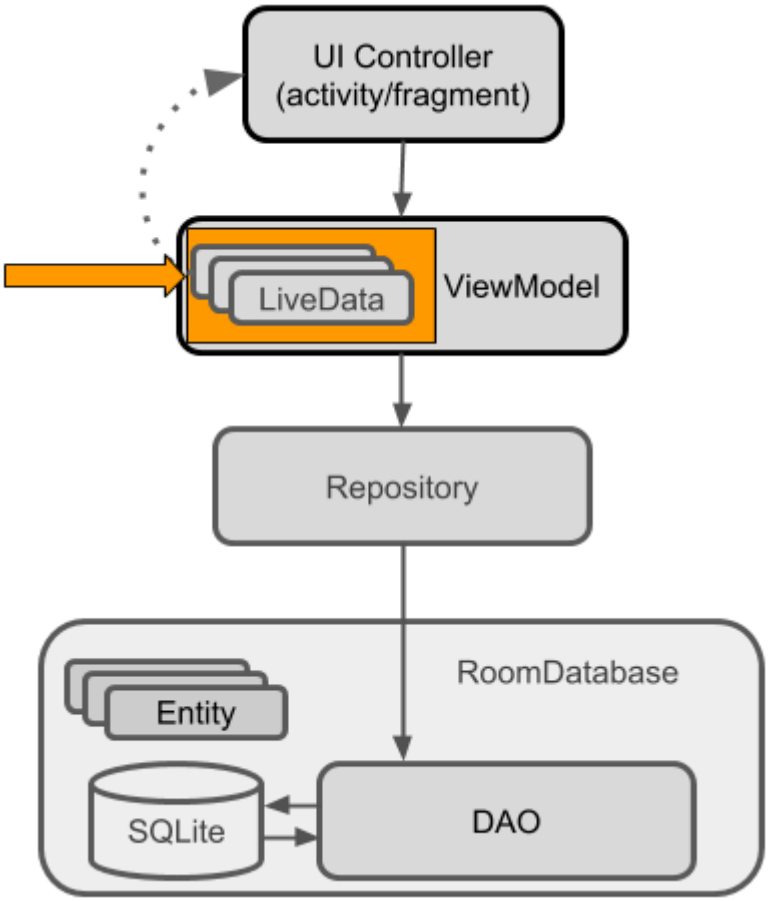

Figura 3 – Relação do LiveData com os componentes

Fonte: Google Developer Training – Architecture Components

Ao observar as alterações nos dados de vários componentes, o **LiveData** não cria caminhos de dependência rígido e explícito entre eles [11] e notifica apenas observadores ativos sobre atualizações. Observadores inativos que estão registrados para observar o **LiveData** não são notificados sobre as alterações que foram realizadas.

As vantagens de usar o **LiveData** incluem [3]:

- Garantir que sua interface do usuário corresponda ao seu estado de dados;
- Não há vazamento de memória;
- Nenhuma falha devido a atividades interrompidas;
- Não há mais manipulação manual do ciclo de vida;
- Dados sempre atualizados;
- Mudanças de configuração apropriada;
- Compartilhamento de recursos.

2.1.1.3. **Navigation**

É uma biblioteca que estrutura a UI dos aplicativos e tem o objetivo de criar uma navegação do aplicativo de forma declarativa e visual. O **Navigation** simplifica a implementação do fluxo da sua aplicação relacionados aos destinos.

Um destino é qualquer lugar que pode ser navegado no aplicativo. Um conjunto de destino faz a composição do gráfico de navegação de um aplicativo. São chamados de ações quando um gráfico de navegação tem conexões entre os destinos [3]. O Navigation é composto por um conjunto de ferramenta, APIs e recursos, que facilita na construção de uma navegação do aplicativo de forma mais positiva.

Navigation permite declarar transações que podem ser gerenciadas, cria automaticamente o comportamento correto para as ações de subir e voltar, oferece suporte completo a links diretos e auxiliares para conectar o **Navigation** aos widgets [5].

A Figura 4 mostra uma representação visual de um gráfico de navegação para um aplicativo de amostra contendo 6 destinos conectados por 5 ações [3].

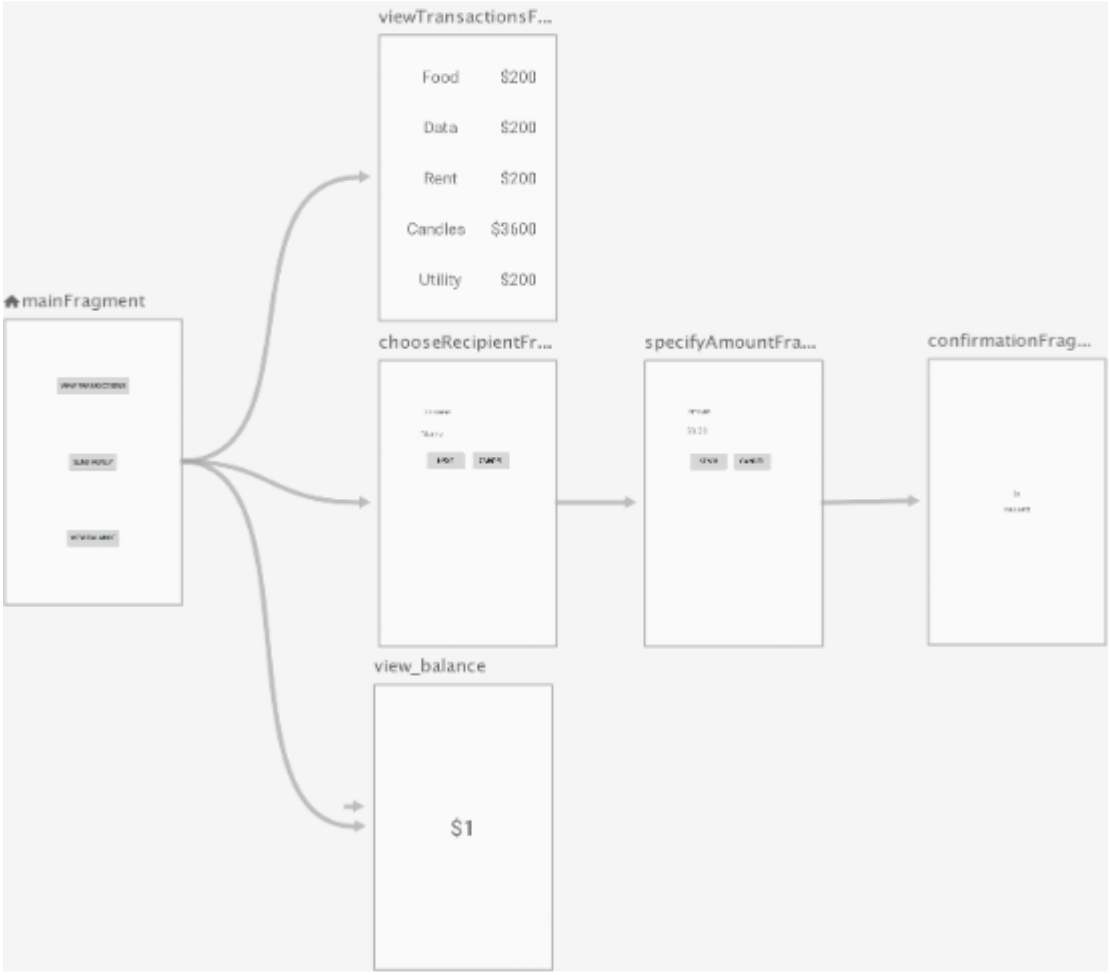

Figura 4 – Gráfico de navegação
Fonte: Android Developers

2.1.1.4. **`Paging`**

É uma biblioteca que facilita o carregamento e a exibição de um conjunto de dados no **`RecyclerView`**. O **`RecyclerView`** tem a funcionalidade de exibir dados de forma mais eficiente e evita procurar os componentes da interface do usuário toda vez que mostrar um item da lista. Vários aplicativos utilizam dados que possuem um grande número de itens, mas exibem uma pequena parte de cada vez, como uma lista de informações que é chamada para ser exibida através do **`RecyclerView`**.

O **Paging** ajuda a observar e exibir um subconjunto razoável desses dados e essa funcionalidade tem algumas vantagens que serão citadas abaixo [3]. Ele também facilita o carregamento e a apresentação de conjuntos de dados grandes com a rolagem rápida e infinita em uma **RecyclerView**.

Vantagens:

- Consumo de menos largura de banda da rede e menos recursos do sistema. Usuários que possuem um baixo plano de dados conseguem olhar as informações dos dados no aplicativo;
- O aplicativo continua respondendo rapidamente a entrada do usuário mesmo com atualizações de dados.

2.1.1.5. Room

É uma biblioteca de banco de dados que facilita o uso do banco de dados SQLite e evita o trabalho de escrever códigos longos e clichês de criação de banco. O **Room** fornece uma camada de abstração sobre o SQLite que é possível permitir acesso ao banco de dados mais robusto e aproveitar todo o poder do SQLite. Conforme mostra a Figura 5, **Room** é uma camada de banco de dados sobre um banco de dados SQLite e cuida de algumas tarefas que o SQLiteOpenHelper era responsável.

É possível executar consultas utilizando apenas anotações na classe do modelo e não é preciso utilizar as classes **Cursor** e **Loader** [12]. Ajuda a criar cache de dados em um dispositivo que está rodando a aplicação onde é executado como uma fonte única do aplicativo e permite que os usuários visualizem uma cópia consistente das informações, mesmo não possuindo uma conexão com a internet.

Existem algumas vantagens da utilização desta biblioteca:

- Verifica consultas SQL em tempo de compilação – cada anotação **@Query** e **@Entity** é verificada no momento da compilação, por isso há menor risco de erro de execução que possa travar o aplicativo;

- Elimina uma grande quantidade de código clichê *(boilerplate)*;
- Possui integração total com outros componentes da arquitetura [13].

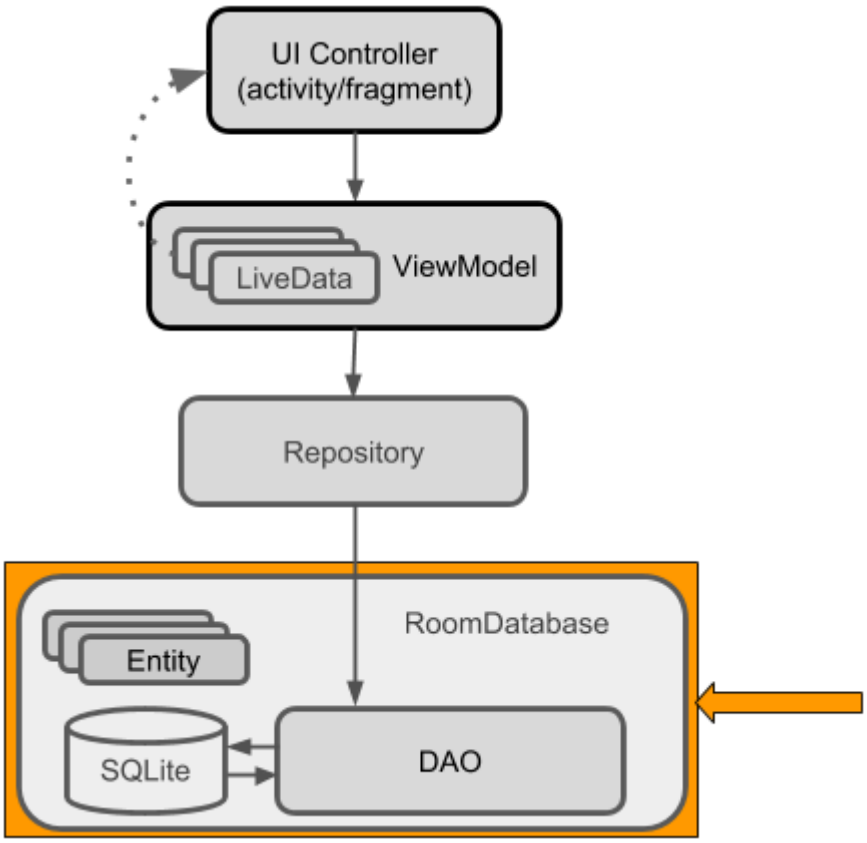

Figura 5 – Relação do Room com os componentes

Fonte: Google Developer Training – Architecture Components

2.1.1.6. `ViewModel`

É uma classe que permite que os dados sobrevivam com as alterações de configuração, como rotacionar a tela do dispositivo. **`ViewModel`** gerencia e armazena dados relacionados à interface do usuário além de se comunicar com os componentes, como mostra a Figura 6, que ilustra sua ligação com o controlador da UI. A **`ViewModel`** atua como um centro de comunicação entre o repositório e a UI [10].

A forma de armazenamento dos dados da interface do usuário é realizada de maneira consistente que sobrevive às mudanças na configuração, como rotações de tela. Separar os dados de

visualização com a lógica do controlador da UI, conforme mostra a Figura 7, permite seguir o princípio de responsabilidade única deixando o código mais fácil de compreender e potencialmente mais eficiente. As ***activities*** e ***fragments*** são responsáveis por desenhar os dados na tela e o **ViewModel** tem a responsabilidade de manter e verificar os dados necessários para a interface do usuário.

A Figura 8 explica os ciclos de vida de uma **Activity** desde a sua criação com surgimento de uma rotação e, em seguida, a sua conclusão. Também é possível visualizar a vida útil do **ViewModel** referente ao ciclo de vida da **Activity** associada [3]. O **ViewModel** pode ser solicitado na primeira vez que o sistema chama o método **onCreate()** da **Activity** até que a **Activity** seja concluída e finalizada.

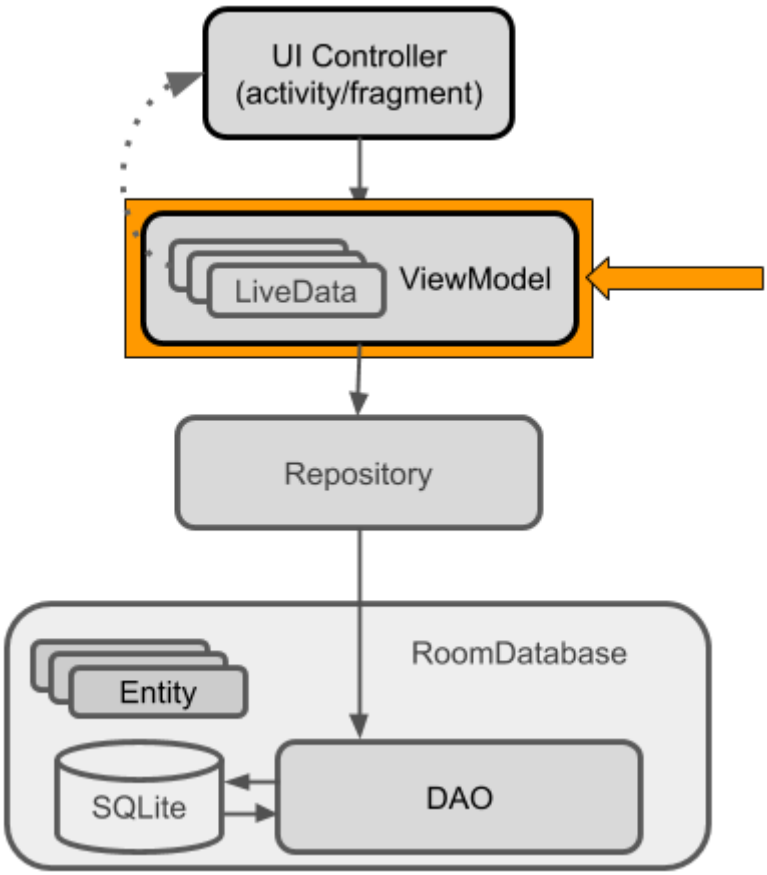

Figura 6 –Relação do ViewModel com os componentes

Fonte: Google Developer Training – Architecture Components

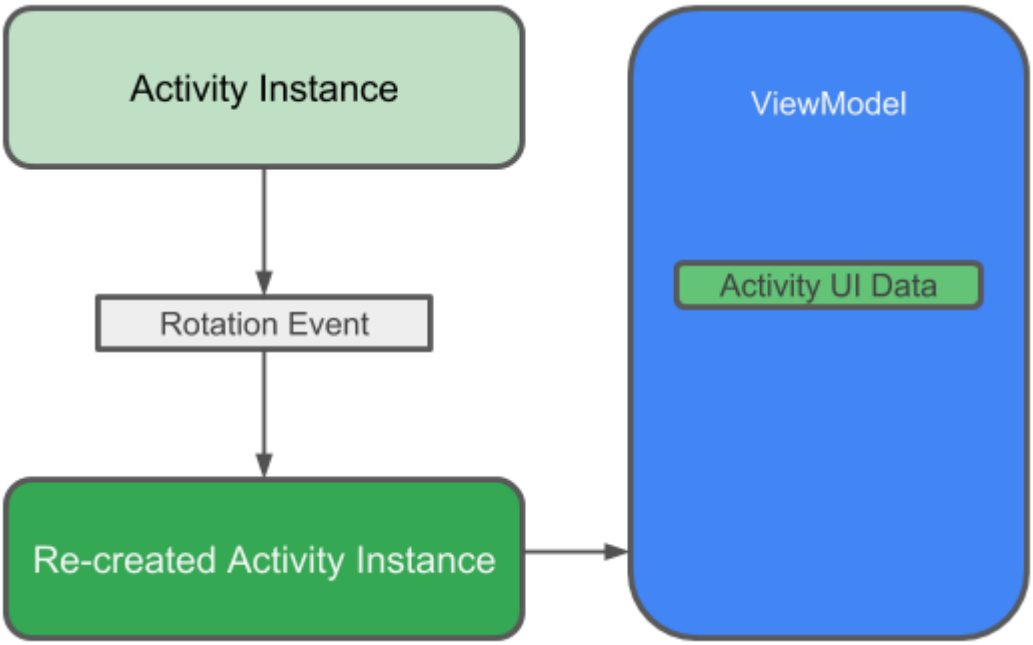

Figura 7 –Separação dos dados de visualização

Fonte: Google Developer Training – Architecture Components

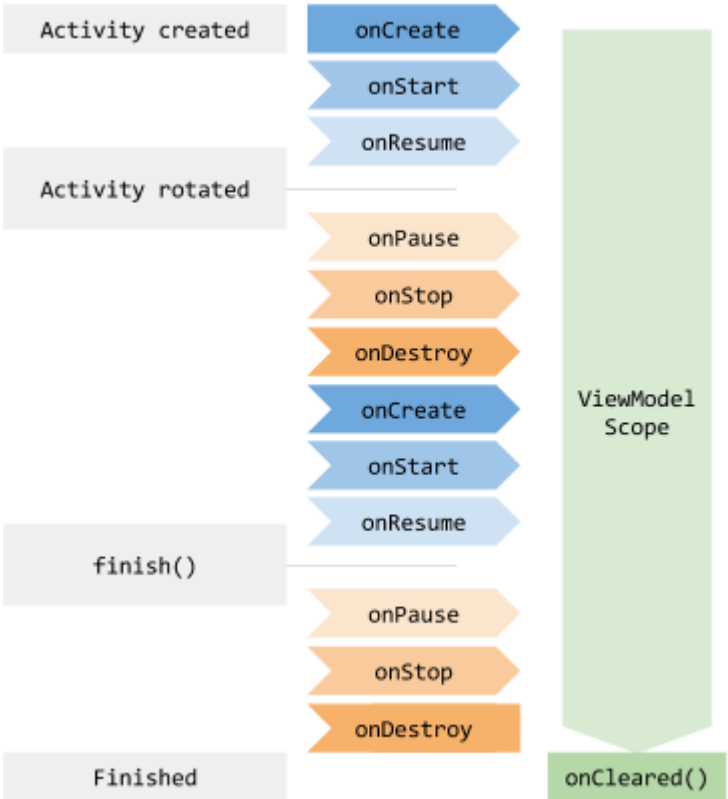

Figura 8 – Estados do ciclo de vida

Fonte: Android Developers

2.1.1.7. **WorkManager**

É uma biblioteca que oferece soluções completas para trabalhos de segundo plano baseados em restrição que necessita de execução garantida e elimina o uso de **Jobs** ou **SyncAdapters** [5]. O **WorkManager** dispõe de uma API moderna e simplificada que facilita a especificação de tarefas assíncronas e concedidas quando devem ser executadas. Quando o trabalho em segundo plano for oportunista, o **WorkManager** executará suas atividades em segundo plano o mais rápido possível. Essa biblioteca também cuida da lógica para iniciar seu trabalho diante de várias situações mesmo o usuário saindo da aplicação, essa forma de realização é chamada de execução garantida.

O **WorkManager** tem seu próprio banco de dados para manter as tarefas. A Figura 9 mostra todo o fluxo realizado por um **WorkManager** [14]. Essa biblioteca consegue escolher a melhor maneira de executar as tarefas baseadas no estado do aplicativo e no nível da API utilizada. Para executar uma de suas tarefas, caso a aplicação não esteja em execução, ele vai escolher a melhor forma de agendar a tarefa em segundo plano e se o aplicativo estiver em execução ele é capaz de executar em uma nova *thread* no processo da aplicação.

O **WorkManager** é uma biblioteca simples, flexível e com algumas vantagens que são [15]:

- Suporte para tarefas assíncronas e periódicas assíncronas;
- Suporte para algumas restrições como armazenamento, condições de rede e status de carregamento;
- Compatibilidade do nível de API;
- Saída de uma solicitação de trabalho usada como entrada para a próxima;

- Suporte ao **`LiveData`** para exibir o estado da solicitação de trabalho na interface do usuário de maneira fácil.

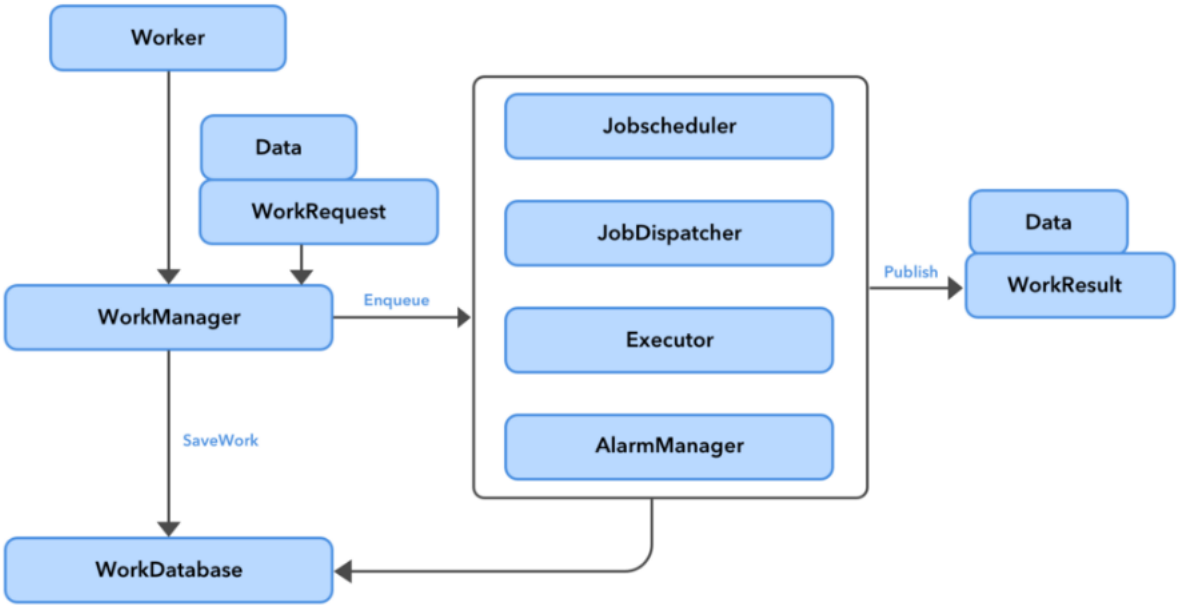

Figura 9 – Fluxo do WorkManager

Fonte: Android Jetpack – Work Manager

2.1.2. Android KTX

É um conjunto de extensões da linguagem Kotlin que facilita a escrita do código de forma mais concisa, idiomática e agradável de usar sem ter a necessidade de adicionar novos recursos às APIs que existem no Android. Essa facilidade na escrita diminui a necessidade de código clichê (*boilerplate*) no desenvolvimento do projeto.

O Android KTX é bastante útil pela interação realizada com APIs do Android usando alguns recursos da linguagem de Kotlin como as funções de extensão, lambdas, parâmetros nomeados e padrão de parâmetro. Ele é organizado em módulos e cada módulo contém um ou mais pacotes, conforme mostra a Figura 10.

Módulo (artefato)	Versão	Pacote
androidx.core:core-ktx	1.0.0-alpha1	Veja todos os pacotes principais abaixo.
androidx.fragment:fragment-ktx	1.0.0-alpha1	androidx.fragment.app
androidx.palette:palette-ktx	1.0.0-alpha1	androidx.palette.graphics
androidx.sqlite:sqlite-ktx	1.0.0-alpha1	androidx.sqlite.db
androidx.collection:collection-ktx	1.0.0-alpha1	androidx.collection
androidx.lifecycle:lifecycle-viewmodel-ktx	2.0.0-alpha1	androidx.lifecycle
androidx.lifecycle:lifecycle-reactivestreams-ktx	2.0.0-alpha1	androidx.lifecycle
android.arch.navigation:navigation-common-ktx	1.0.0-alpha01	androidx.navigation
android.arch.navigation:navigation-fragment-ktx	1.0.0-alpha01	androidx.navigation.fragment
android.arch.navigation:navigation-runtime-ktx	1.0.0-alpha01	androidx.navigation
android.arch.navigation:navigation-testing-ktx	1.0.0-alpha01	androidx.navigation.testing
android.arch.navigation:navigation-ui-ktx	1.0.0-alpha01	androidx.navigation.ui
android.arch.work:work-runtime-ktx	1.0.0-alpha01	androidx.work.ktx

Figura 10 – Módulos do Android KTX

Fonte: Android Developers

2.2. Mineração de Repositórios

A mineração de repositórios é um campo da engenharia de software onde desenvolvedores profissionais e pesquisadores usam técnicas de mineração de dados para fazer a análise e extrair informações úteis para o processo de desenvolvimento [17].

A próxima seção apresenta a ferramenta de mineração de repositórios que foi utilizada como base para o desenvolvimento do projeto.

2.2.1. PyDriller

É um framework em Python que ajuda desenvolvedores a analisar repositórios Git [4]. É possível extrair facilmente alguns dados do *commit* como por exemplo, a mensagem e o autor, mas também podemos retirar informações que estão contidas nos arquivos do repositório e exportar esses dados para arquivo CSV.

Para desenvolver um programa que faça a mineração dos repositórios desejados, é necessário criar o ***`RepositoryMining`*** que é uma classe que

recebe como entrada uma lista de repositórios e retorna um gerador que faz a iteração sobre os *commits*. O PyDriller clona em uma pasta temporária todos os repositórios que foram passados como entrada e em seguida é apagado.

A configuração da análise do projeto é passada como parâmetro na classe ***`RepositoryMining`***. É possível informar qual o tipo de arquivo que deseja fazer a análise e após verificar os *commits* que possuem o determinado arquivo escolhido, é possível informar qual informação deseja analisar.

3. METODOLOGIA

Neste trabalho, desenvolvemos análises em Python para clonar repositórios Git com a ferramenta PyDriller e percorrer os *commits* dos repositórios. As implementações são capazes de extrair dados sobre os componentes do Android Jetpack e gerar as informações em arquivos CSV. Para a visualização dos dados foi necessário gerar os gráficos a partir dos arquivos que foram exportados.

As próximas seções detalham como foi realizada a implementação e como foram gerados os resultados.

3.1. Implementação

A implementação foi dividida em duas categorias, com algumas condições para a configuração da análise desejada:

- Análise do arquivo *build.gradle*;
- Análise dos arquivos *.java* e/ou *.kt*.

3.1.1. Análise do arquivo build.gradle

Foi desenvolvido um *script* em Python, com o auxílio da ferramenta PyDriller, para verificar os repositórios que foram selecionados para a análise. O ***`RepositoryMining`*** é uma classe que recebe como entrada o caminho do repositório e retorna um gerador que itera sobre os *commits*. Um dos parâmetros de entrada do ***`RepositoryMining`*** que utilizamos foi o tipo

de arquivo *build.gradle* para poder realizar verificação de mudanças neste arquivo nos *commits* explorados de cada repositório. Após analisar se ocorreu alguma mudança do arquivo passado como entrada, foi implementado algumas condições da existência de dependência das bibliotecas do *Architecture Components* e Android KTX.

Esses projetos são analisados para identificar quais contém os tipos de implementação no arquivo *build.gradle* com indicações da importação dos componentes do Android Jetpack.

A Figura 11 mostra uma parte da implementação para a análise dos repositórios que possuem modificações no arquivo *build.gradle*. O trecho ilustrado verifica se na alteração do arquivo existem as palavras que indicam as importações das bibliotecas.

```
for commit in RepositoryMining(path_to_repo=url,
    only_modifications_with_file_types=['build.gradle']).traverse_commits():
    for modification in commit.modifications:
        if 'androidx.room:room-runtime' in modification.diff:
            if 'build.gradle' in modification.filename:
                output_file_room_androidx.write("Project {}\n".format(commit.project_name))
                print(commit.project_name)
```

Figura 11 – Análise do arquivo *build.gradle* para verificar se existe as expressões que estão nas condições do *script*.

A Tabela 1 exibe as expressões que são utilizadas para a utilização das bibliotecas da *Architecture Components* e a Tabela 2 mostra as palavras-chave utilizadas na implementação do Android KTX. Quando é encontrado algum repositório com essas modificações, o nome do projeto referente ao repositório encontrado é adicionado ao arquivo CSV.

Architecture Components	
Lifecycle	android.arch.lifecycle:extensions
	androidx.lifecycle:lifecycle-extensions
LiveData	android.arch.lifecycle:livedata
	androidx.lifecycle:lifecycle-livedata
Navigation	android.arch.navigation:navigation
Paging	android.arch.paging:runtime
	androidx.paging:paging-runtime
Room	androidx.room:room-runtime
	android.arch.persistence.room:runtime
ViewModel	android.arch.lifecycle:viewmodel
	androidx.lifecycle:lifecycle-viewmodel
WorkManager	android.arch.work:work-runtime

Tabela 1 – Expressões para utilização das bibliotecas da Architecture Components no arquivo *build.gradle*

Android KTX
androidx.palette:palette-ktx
androidx.sqlite:sqlite-ktx
androidx.collection:collection-ktx
androidx.lifecycle:lifecycle-viewmodel-ktx
androidx.lifecycle:lifecycle-reactivestreams-ktx
android.arch.navigation:navigation-common-ktx
android.arch.navigation:navigation-fragment-ktx
android.arch.navigation:navigation-runtime-ktx
android.arch.navigation:navigation-testing-ktx
android.arch.navigation:navigation-ui-ktx
android.arch.work:work-runtime-ktx

Tabela 2 – Expressões para utilização das bibliotecas o Android KTX no arquivo *build.gradle*

3.1.2. Análise dos arquivos .java e/ou .kt

Um outro *script* em Python foi implementado para dar continuidade à primeira categoria de análise e verificar quantos repositórios utilizam de fato os componentes do Android Jetpack nas classes. Foi passado como entrada do ***RepositoryMining*** o tipo *.java* e *.kt* para saber quais repositórios analisados contém esses arquivos. Através dessa verificação de existência dos arquivos, foram implementadas algumas condições para saber se as importações em relação ao Android Jetpack nas classes estavam sendo realizadas.

Essa segunda categoria foi implementada para verificar quantos repositórios e projetos em Android utilizam as bibliotecas do Android Jetpack nas classes, além de colocar a dependência no arquivo *build.gradle*.

A implementação analisa os arquivos *.java* e *.kt* e verifica se existe a importação das expressões das bibliotecas em estudo, como mostra a Figura 12.

```
for commit in RepositoryMining(path_to_repo=url,
    only_modifications_with_file_types=['.java', '.kt']).traverse_commits():
    for modification in commit.modifications:
        if 'import android.arch.persistence.room' in modification.diff:
            if '.java' in modification.filename:
                output_file_room_java_kt.write("Project {}\n".format(commit.project_name))
                print(commit.project_name)
            if '.kt' in modification.filename:
                output_file_room_java_kt.write("Project {}\n".format(commit.project_name))
                print(commit.project_name)
```

Figura 12 – Análise do arquivo *.java e .kt* para verificar se existe as expressões que estão nas condições do script

A Tabela 3 mostra as expressões que foram pesquisadas nas alterações dos arquivos para verificar a utilização da *Architecture Components*. A Tabela 4 indica as palavras que são utilizadas nas classes para o uso do Android KTX.

Architecture Components	
Lifecycle	import android.arch.lifecycle.Observer
	import androidx.lifecycle.Observer
LiveData	import android.arch.lifecycle.LiveData
	import androidx.lifecycle.LiveData
Navigation	import androidx.navigation
Paging	import android.arch.paging
	import androidx.paging
Room	import android.arch.persistence.room
	import androidx.room
ViewModel	import android.arch.lifecycle.AndroidViewModel
	import android.arch.lifecycle.ViewModel
	import androidx.lifecycle.AndroidViewModel
	import androidx.lifecycle.ViewModel
WorkManager	import androidx.work

Tabela 3 – Expressões para importação das bibliotecas da Architecture Components nas classes

Android KTX
import androidx.fragment.app
import androidx.palette.graphics
import androidx.sqlite.db
import androidx.collection
import androidx.lifecycle
import androidx.navigation
import androidx.navigation.fragment
import androidx.work.ktx
import androidx.navigation.testing
import androidx.navigation.ui

Tabela 4 – Expressões para importação das bibliotecas do Android KTX nas classes

3.2. Repositórios de Teste

Para gerar um cenário pequeno e realizar alguns testes da ferramenta PyDriller, utilizamos alguns repositórios públicos do GitHub que usam bibliotecas do Android Jetpack. Os repositórios podem ser encontrados nas seguintes URLs:

- https://github.com/RaquelMSantos/Cooking;
- https://github.com/RaquelMSantos/PopularMovies;
- https://github.com/RaquelMSantos/PlayMusic.

Esses repositórios foram escolhidos para validar a implementação e verificar se os *scripts* estavam extraindo corretamente os dados do repositório e suas análises de *commits*. No início, foi extraído para o arquivo CSV a linha de código que foi alterada do arquivo *build.gradle* e esse procedimento foi realizado para obter verificação de forma mais precisa. Dessa forma, foi possível avaliar os resultados obtidos diretamente com a extração de informações e seguir para um ambiente exploratório com os repositórios válidos para a análise.

3.3. Experimento

Para realizar o teste com mais informações e gerar uma análise de dados mais complexa, foram extraídos do catálogo *F-Droid* [1] o endereço dos repositórios a serem avaliados. Foram retirados 1584 repositórios do catálogo e armazenados em um arquivo. A partir desse arquivo, foram removidos repositórios duplicados, não existentes e não acessíveis por necessitar de credenciais para a clonagem. Após essa remoção de repositórios, restaram 1406 repositórios válidos.

Além dos repositórios do catálogo *F-Droid*, foi utilizada a API do GitHub [18] para extrair repositórios de aplicativos desenvolvidos na linguagem Java e Kotlin. Foram retirados 200 repositórios que foram ordenados pelo número de estrelas que possuíam. Ao total, 1606 repositórios fizeram parte da análise final.

3.4. Resultados

Foi realizada a análise dos 1606 repositórios e gerado um arquivo CSV para cada biblioteca após a mineração sucedida dos repositórios investigando o uso do Android Jetpack. Na Tabela 5 temos os dados de repositórios que utilizam as dependências do Android Jetpack e 1504 repositórios não fazem dependência de nenhuma das bibliotecas.

Biblioteca	Quantidade de repositórios
Android KTX	11
Lifecycle	74
LiveData	7
Navigation	6
Paging	8
Room	60
ViewModel	21
WorkManager	14

Tabela 5 – Dados coletados sobre o uso das dependências do Android Jetpack em repositórios

Na Figura 13 temos os dados da Tabela 5 em forma de gráfico. Podemos observar que o **`Lifecycle`** seguido do **`Room`** são as bibliotecas que tiveram mais uso das suas dependências nos projetos analisados. Com uma quantidade de 74 repositórios que possuem a biblioteca mais utilizada, pode-se dizer que a adoção de Android Jetpack ainda está baixa. Com um grande crescimento da criação de aplicativos móveis para a plataforma Android, é de grande importância garantir que as aplicações sejam otimizadas com o uso do Android Jetpack para ter alta qualidade para a usabilidade dos usuários.

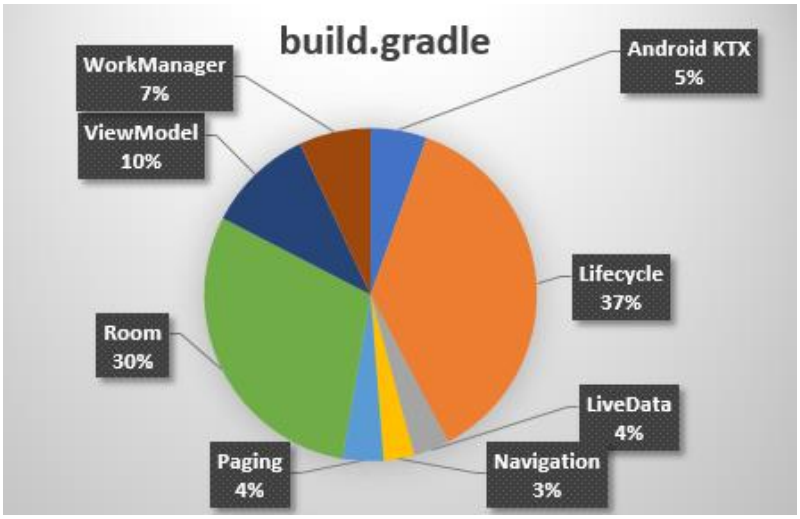

Figura 13 – Gráfico de utilização das dependências do Android Jetpack em repositórios

Na Tabela 6 podemos analisar a quantidade de repositórios que incluíram a importação das bibliotecas nas classes *.java* ou *.kt*. Verificamos que os números de repositórios em relação a Tabela 5 foram diferentes. Esse caso aconteceu pelo fato de existir outras formas de adicionar dependências nos projetos no qual não tratamos, um exemplo seria declarar a dependência da biblioteca em outro arquivo e com expressões diferentes das quais utilizamos para as análises. Como podemos verificar, o número de repositórios que utilizam a importação das bibliotecas teve um aumento, isso aconteceu para a maioria das bibliotecas. Então podemos concluir que vários projetos além de utilizar as dependências, eles realmente estão fazendo a importação nas classes para a possibilidade de seu uso e a quantidade de repositórios que não faz nenhuma importação das bibliotecas é de 1463.

Biblioteca	Quantidade de repositórios
Android KTX	77
Lifecycle	65
LiveData	60
Navigation	5
Paging	9
Room	67
ViewModel	74
WorkManager	15

Tabela 6 – Dados coletados sobre o uso das importações do Android Jetpack nas classes dos repositórios

A Figura 14 apresenta em forma de gráfico os dados que foram coletados de acordo com as importações das bibliotecas realizadas nos repositórios analisados. O gráfico está com as porcentagens para representar qual biblioteca foi mais utilizada comparada com as demais. Como observado no gráfico, o Android KTX está sendo o mais utilizado e é um dos componentes mais recentes do Android Jetpack. Podemos considerar que a sua adoção está caminhando bem.

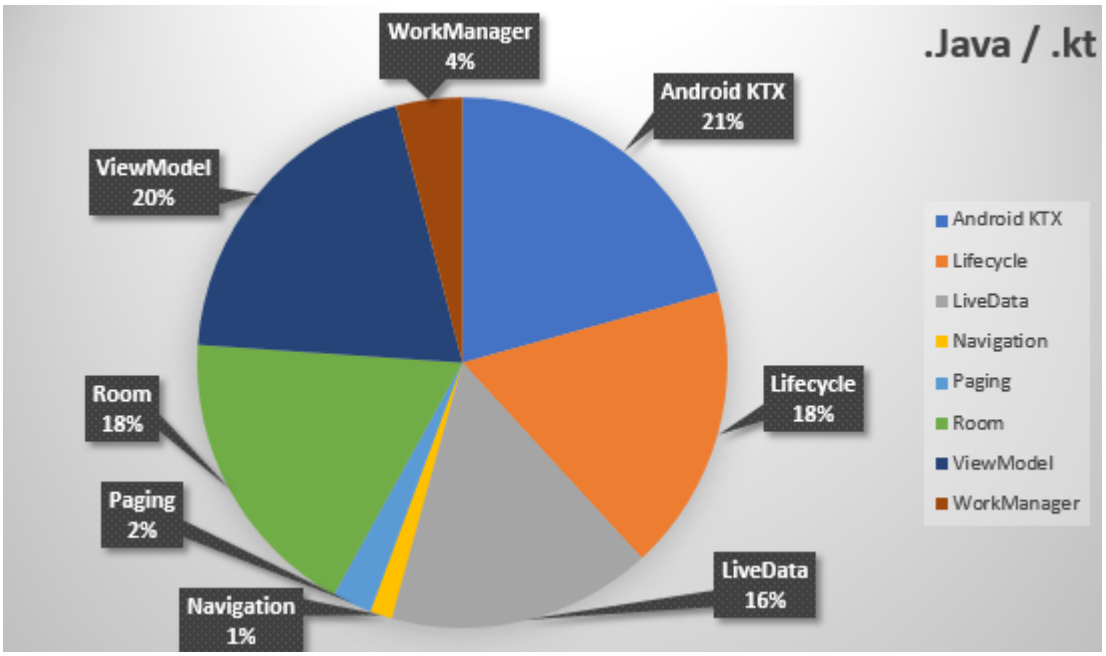

Figura 14 – Gráfico da importação do Android Jetpack em repositórios

4. CONSIDERAÇÕES FINAIS

Neste trabalho, foi realizada inicialmente uma revisão da literatura onde parte desses materiais foram usados como referências bibliográficas. Após o estudo sobre o Android Jetpack, foi possível identificar as suas quatro categorias. Para a escolha do *Architecture Componentes* e Android KTX, foi levado em consideração que elas são mais recentes que as outras bibliotecas que fazem parte do Android Jetpack, desta forma decidimos analisar quantos projetos já estão se adaptando com essas novidades.

Em seguida, utilizamos a ferramenta PyDriller para realizar a mineração de repositórios e começamos a fazer alguns testes com projetos que já possuíam alguma biblioteca do Android Jetpack. Após a realização dos testes utilizando as expressões de dependências, ficamos mais seguros para fazer a análise para o experimento. Após a etapa de testes, selecionamos os 1606 repositórios foram ser analisados para a coleta dos dados final.

A partir dos dados obtidos, percebemos que o número de repositórios que utilizam o Android Jetpack ainda é um pouco baixo mesmo com uma importância em melhorar o desempenho da aplicação. Podemos concluir que as bibliotecas analisadas não estão apenas sendo declaradas no arquivo *build.gradle* como também estão sendo utilizadas nas classes.

Como trabalhos futuros, podemos analisar todas as quatro categorias do Android Jetpack e verificar se o número de repositórios que utilizam as bibliotecas continua baixo mesmo não tratando apenas de bibliotecas recentes. Podemos também verificar a data dos últimos *commits* para investigar se os repositórios continuam ativos.

5. REFERÊNCIAS

[1] F-Droid, disponível em https://f-droid.org, acessado em 26 de agosto de 2018.

[2] Estatísticas de uso de aplicativos no Brasil, disponível em https://www.dubsolucoes.com/single-post/estatisticas-de-uso-de-aplicativos-no-Brasil, acessado em 25 de agosto de 2018.

[3] Android Developers, disponível em https://developer.android.com, acessado em 26 de agosto de 2018.

[4] PyDriller, disponível em https://github.com/ishepard/pydriller, acessado em 26 de agosto de 2018.

[5] Blog Google Developers, disponível em https://developers-br.googleblog.com/2018/05/use-o-android-jetpack-para-acelerar-o.html, acessado em 20 de outubro de 2018.

[6] Android Architecture Components - Lifecycle, disponível em https://medium.com/android-dev-moz/aac2-a7c59e1a3cf7, acessado em 20 de outubro de 2018.

[7] Android Architecture Components - LifecycleOwner, disponível em https://medium.com/android-dev-moz/aac3-1df6cd39b4e0, acessado em 20 de outubro de 2018.

[8] Componentes da Arquitetura Android: Lifecycle e LiveModel, disponível em https://code.tutsplus.com/pt/tutorials/android-architecture-components-lifecycle-and-livemodel--cms-29275, acessado em 25 de outubro de 2018.

[9] Android Architecture Components – LiveData, disponível em https://medium.com/android-dev-moz/aac5-9c56e6b4cffc, acessado em 25 de outrubro de 2018.

[10] Google Developer Training – Architecture Components, disponível em https://google-developer-training.gitbooks.io/android-developer-advanced-course-concepts/content/unit-6-working-with-architecture-components/lesson-14-architecture-components/14-1-c-architecture-components/14-1-c-architecture-components.html#lifecycle, acessado em 25 de outubro de 2018.

[11] Codelabs – Android lifecycle-aware componentes, disponível em https://codelabs.developers.google.com/codelabs/android-lifecycles, acessado em 25 de outubro de 2018.

[12] Android Architecture Components – Room, disponível em https://medium.com/android-dev-moz/aac6-b46de8513df8, acessado em 1 de novembro de 2018.

[13] Android Architecture Components: Room – Introduction, disponível em https://android.jlelse.eu/android-architecture-components-room-introduction-4774dd72a1ae, acessado em 1 de novembro de 2018.

[14] Android Jetpack – Work Manager, disponível em https://medium.com/@balakrishnan.750/android-jetpack-work-manager-de6909eb684d, acessado em 1 de novembro de 2018.

[15] Codelabs – Background Work with Manager, disponível em https://codelabs.developers.google.com/codelabs/android-workmanager, acessado em 1 de novembro de 2018.

[16] PyDriller documentation, disponível em https://pydriller.readthedocs.io/en/latest, acessado em 2 de novembro de 2018.

[17] Mining Software Repositories – MSR, disponível em https://www.webopedia.com/TERM/M/mining-software-repositories-msr.html, acessado em 6 de novembro de 2018.

[18] GitHub Developer, disponível em https://developer.github.com/v3/, acessado em 2 de novembro de 2018.