UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE COMPARATIVA ENTRE PADRÕES ARQUITETURAIS NA PLATAFORMA ANDROID

ALEXANDRE FREITAS CESAR

JOÃO PESSOA

2019

UNIVERSIDADE FEDERAL DA PARAÍBA

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE COMPARATIVA ENTRE PADRÕES ARQUITETURAIS NA PLATAFORMA ANDROID

ALEXANDRE FREITAS CESAR

JOÃO PESSOA

CENTRO DE INFORMÁTICA

2019

Alexandre Freitas Cesar

ANÁLISE COMPARATIVA ENTRE PADRÕES ARQUITETURAIS NA PLATAFORMA ANDROID

Monografia apresentada ao curso Ciência da Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em ciência da computação.

Orientador: Raoni Kulesza

Setembro de 2019

Ficha catalográfica: elaborada pela biblioteca do CI.

Será impressa no verso da folha de rosto e não deverá ser contada.
Se não houver biblioteca, deixar em branco.

ANÁLISE COMPARATIVA ENTRE PADRÕES ARQUITETURAIS NA PLATAFORMA ANDROID

ALEXANDRE FREITAS CESAR

Trabalho de Conclusão de Curso de Ciência da Computação intitulado Análise comparativa entre padrões arquiteturais na plataforma Android de autoria de Alexandre Freitas Cesar, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Raoni Kulesza

Prof.

Prof.

Coordenador(a) do Departamento CI/UFPB

João Pessoa, 14 de setembro de 2019

Centro de Informática, Universidade Federal da Paraíba

Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600

Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

AGRADECIMENTOS

Agradeço ao professor Raoni Kulesza por ter me ensinado os conhecimentos do mercado de trabalho, assim como pelo seu apoio e suporte na pesquisa do meu trabalho de conclusão.

Agradeço ao meu amigo Amaro por ter feito os trabalhos comigo e acompanhado em momentos difíceis.

Agradeço às professoras Daniela Coelho, Thais Gaudencio e Danielle Rousy pelo conhecimento e suas personalidades inspiradoras.

Agradeço aos professores Clauirton de Albuquerque, Ronei marcos, Christian Azambuja, José Antônio Gomes pelo conhecimento e suas personalidades fortes e inspiradoras.

"A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original".

Albert Einstein

RESUMO

O desenvolvimento de aplicações Android têm sido de interesse para pesquisas desde o lançamento do primeiro smartphone Android. O *architecture tradeoff analysis method* (ATAM) vai ser utilizado nesta pesquisa para ajudar a escolher uma arquitetura adequada para o desenvolvimento de software. Existe um padrão de arquitetura chamado *Model-View-Presenter* (MVP) que é o mais utilizado para aplicações Android. O objetivo da pesquisa é comparar o MVP e o *Model-View-ViewModel* (MVVM) com foco em desempenho. Os resultados mostraram que o MVVM apresenta uma melhor qualidade de desempenho para CPU e memória quando comparado com o MVP.

Palavras-chave: Android, mvp, mvvm, dispositivos móveis, desenvolvimento móvel, arquitetura de software.

ABSTRACT

Android application development has been of interest to research since the launch of the first Android smartphone. The architecture tradeoff analysis method (ATAM) will be used in this research to help choose a suitable architecture for software development. There is an architecture standard called Model-View-Presenter (MVP) that is most commonly used for Android applications. The goal of the research is to compare MVP and Model-View-ViewModel (MVVM) with a focus on performance. The results showed that MVVM has a better performance quality for CPU and memory when compared to MVP.

Key-words: Android, mvp, mvvm, mobile devices, mobile development, software architecture.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	*Associação Brasileira de Normas Técnicas*
GUI	*Graphical User Interface*
MVC	*MODEL-VIEW-CONTROLLER*
MVP	*MODEL-VIEW-PRESENTER*
MVVM	*MODEL-VIEW-VIEW-MODEL*
ATAM	*Architecture tradeoff analysis method*
MB	*Megabyte*

SUMÁRIO

1 INTRODUÇÃO

O Android foi apresentado ao mundo em 2005, e ao longo desses anos de existência, a plataforma alcançou sucesso, tornando-se o sistema operacional móvel mais instalado[16]. Durante esse tempo, foram lançadas várias versões diferentes do sistema operacional, com o Android sempre se tornando mais desenvolvido. Durante a evolução do desenvolvimento Android para aplicações móveis, o Google não forneceu um padrão de arquitetura para aplicar no desenvolvimento de aplicações[1]. Nesse quesito, a plataforma oferece aos desenvolvedores uma liberdade muito grande para desenvolver aplicações, no entanto essa liberdade pode gerar problemas para testar, desempenho inferior e aumento de código quando o objetivo é desenvolver uma aplicação dentro de uma empresa[1].

1.1 Definição do Problema

As aplicações atuais são construidas usando arquiteturas específicas, no qual diferentes arquiteturas oferecem outras vantagens que vêm com diferentes incapacidades. A aplicação de uma arquitetura depende muitas vezes dos requisitos específicos do projeto e propriedades do seu Framework. Os padrões de arquitetura geralmente tem propriedades diferentes para atender outros tipos de necessidade, algumas arquiteturas mostram melhor desempenho dependendo da sua atividade, o que pode deixar as funcionalidades mais fluidas[10].

O padrão de arquitetura é um campo de conhecimento que foi desenvolvido durante os anos para melhorar a qualidade dos sistemas e para encontrar soluções[3]. Este conhecimento vai ser importante no futuro, pois novos sistemas podem exigir qualidade de desempenho. A arquitetura quase sempre é representada por um conjunto de propriedades e uma comparação pode ser feita em cima dessas propriedades, no entanto não basta apenas comparar suas propriedades, é necessario analisar os resultados, no qual esta análise pode ser usada para tomar decisões sobre qual arquitetura usar em um projeto[10].

1.2 Objetivo geral

A intenção deste trabalho é comparar os principais padrões arquiteturais usados pelos desenvolvedores para construir aplicações na plataforma Android. O presente trabalho tem como principais objetivos específicos:

1.2.1 Objetivo específico

- Utilizar a método ATAM para comparar arquiteturas.

- Conceituar e identificar os padrões de arquitetura mais conhecidos segundo autores.
- Diferenças e semelhanças entre *Model-View-Presenter*(MVP) e *Model-View-ViewModel*(MVVM).
- Responder qual é arquitetura que deve ser usada para desenvolver uma aplicação considerando o seu desempenho, MVP ou MVVM ?.

1.3 Motivação

O Android é o sistema operacional móvel mais utilizado do mundo, em uma pesquisa feita em 2013 foi mostrado que 71% dos programadores para aplicações moveis desenvolviam para Android[16]. Deste modo, uma investigaçao nesta área de desenvolvimento de software será favorável para os desenvolvedores de aplicações móveis. É possível notar que novas aplicações Android estão constatemente sendo lançadas no mercado com mais desempenho. A arquitetura mais tradicional utilizada em aplicações Android é o MVP, mas existe outras arquiteturas como o MVVM[3].

Uma motivação importante para esta pesquisa é que os dados sobre o desempenho das arquiteturas MVP e MVVM não foram feitos com a API atual e ferramentas que mostrem o desempenho em tempo real. Uma análise de desempenho com está ferramenta pode mostrar resultados favoráveis para os desenvolvedores de aplicações móveis e outros pesquisadores. A pesquisa tem como objetivo descobrir se existe um desempenho melhor do MVVM quando comparado com o MVP.

1.4 Grupos Alvo

Esta seção consiste de apresentar quem são os stakeholders da pesquisa. O grupo alvo são os desenvolvedores de aplicações Android que buscam um melhor desempenho da sua aplicação. Conhecendo os padrões arquiteturais baseado nas métricas importantes, os desenvolvedores poderão ter o conhecimento sobre qual arquitetura deve escolher para solucionar problemas de desempenho presentes na sua aplicação. A decisão da escolha da arquitetura poderá afetar o desempenho da aplicação desenvolvida. Os pequisadores também são um grupo alvo desta pesquisa.

1.5 Organização do trabalho

O Capítulo 2 descreve a identificação dos padrões arquitetuturais existentes e conceitos importantes para entender o funcionamento do Android.

O Capítulo 3 descreve o método utilizado na pesquisa para fazer a análise dos padrões arquiteturais.

O Capítulo 4 descreve os cenários de uso utilizados na pesquisa para escolher as ações que são necessárias para testes.

O Capítulo 5 descreve as aplicações utilizadas para testes de desempenho de CPU, memória e casos de teste.

O Capítulo 6 descreve a apresentação dos resultados.

O Capítulo 7 descreve o resumo dos resultados

O Capítulo 8 descreve qual o padrão se apresentou melhor diante dos cenários analisados em tempo de execução e memória.

2 CONCEITOS GERAIS

Neste capítulo é descrito os principais conceitos do Android, no qual os padrões arquiteturais tradicionais são identificados,conceituados e diferenciados.

2.1 Constituição de uma aplicação Android

O sistema operacional Android é contruído sobre alguns dos principais componentes e camadas. Os componentes tem o objetivo de ajudar a desenvolver aplicações robustas, testavéis e sustentáveis. O Google publicou esses componentes com o intuito de auxiliar no desenvolvimento de aplicativos para o Android. Nesta seção será descrito três conceitos importantes neste TCC: *Activity, Fragment* e Recursos.

2.1.1 Activity

É a maneira de interagir com o usuário, ou seja, uma *Activity* é basicamente uma interface de uma aplicação, para cada nova tela que é criada é necessário uma *Activity*[17]. As *Activitys* quando são abertas utilizam o conceito de pilha, exemplo: quando se tem uma tela e é acessado uma nova tela, a tela nova fica sobre a tela antiga[17]. Dentro da activity é carregado um arquivo xml que compõe a interface.

2.1.1.1 Ciclo de vida

A maioria dos componentes de apps Android têm ciclos de vida ligados a eles, que são gerenciados diretamente pelo próprio sistema.

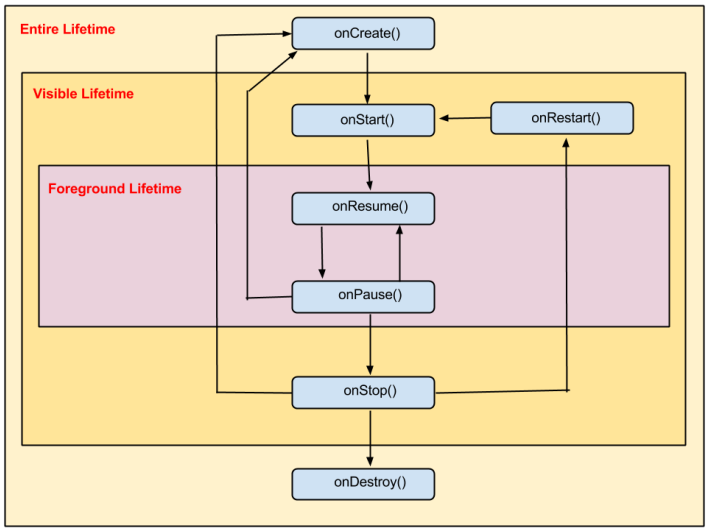

Figura 1 – Ciclo de vida de uma Activity

O ciclo de vida é dividido em 3 níveis: *Entire lifetime, Visible lifetime e Foreground lifetime*. O *Entire lifetime* aborda todo o ciclo do *created* até o *destroyed*. O *Visible lifetime* aborda só a parte que o usuário pode ver a sua atividade que é do *start* até o *stop*; O *Foreground lifetime* aborda todo contexto que permeti interagir com a atividade. Quando o usuário executa a aplicação, a atividade começa no *ONCREATE* que é chamado apenas uma única vez, o usuário ainda não pode ver nada na tela, logo depois a atividade vai para o *ONSTART* no qual inicia a apresentação da tela, mas não pode interagir com a tela, no *ONRESUME* o usuário já está em execução e é possível interagir[18].

2.1.2 Fragment

É uma mini activity que possui o seu próprio ciclo de vida e estão interligados a uma activity, mas não é um componente do sistema, pois não tem assinatura no AndroidManifest[7]. O funcionamento é parecido como as *Activity's* funcionam. A diferença entre ambos é que a Activity é um componente do Android e tem uma conexão com o processo da aplicação, já o *Fragment* fica sobre o encargo direto da *Activity*[7].

2.1.3 Recursos

Os recursos são os arquivos adicionais e conteúdo estático que o código usa. Existe vários tipos de recursos como: *Layout, bitmaps*, dimensão e etc.

2.2 Padrões de arquitetura

No decorrer de toda a história do desenvolvimento Android, o Google não entregou sua posição sobre qual o padrão de arquitetura de software que deveria utilizar para desenvolver aplicativos Android[1]. Quando lançaram oficialmente as primeiras versões do Android, o objetivo era usar o MVC para o desenvolvimento das aplicações, mas rapidamente surgiram problemas nesse padrão arquitetural, então foi preciso criar novas abordagens como os padrões MVP, MVVM, MVC.

2.2.1 Arquitetura multicamadas

O sistema multicamadas não tem uma definição clara de quais são as camadas que certamente devem ter em um aplicativo Android, nem mesmo o número mínimo e máximo. Este sistema faz uso de objetos distribuídos(permite a operação com objetos remotos) associado à utilização de interfaces para executar seus procedimentos, podendo esta aplicação Android ser dividida em várias partes, cada uma bem definida, com suas características e responsável por determinadas funções[13]. Em um aplicativo nesta forma,

pelo menos três camadas são necessárias: apresentação, regras de negócios e banco de dados.

2.2.2 Model-View-Controller

O MVC foi criado na década de 70 como forma de separar a lógica de apresentação da lógica de negócios, reduzindo a ligação entre classes. No ano de 2009, quando saiu as primeiras versões do Android, a ideia era usar o padrão arquitetural MVC para o desenvolvimento dos projetos. O incentivo para usar o MVC foi os desenvolvedores conseguirem paralelizar o desenvolvimento e conseguir fazer o que chamamos de reuso de código[2]. Esta abordagem separa sua aplicação em um nível de composição com 3 conjuntos de funções. Sendo eles:

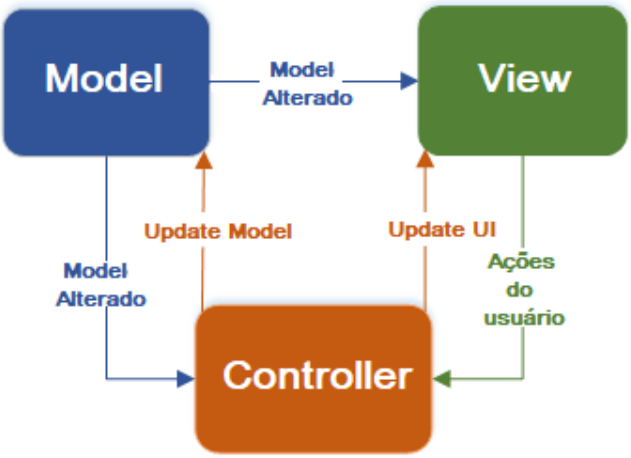

Figura 2 – Arquitetura MVC

A Camada *View* é responsável por apresentar informações de forma gráfica para o usuário e faz a comunicação com o controller. A Camada *Controller* é responsável por fazer a ligação entre a interface gráfica e a lógica de negócios, alterando atributos e mostrando as modificações na interface gráfica. A Camada *Model* é onde a lógica de negócio é definida, incluindo a persistência de dados[2].

O MVC consegue separar o *Model* e a *View*. Deste jeito o *Model* pode ser testado, pois não está vinculado a nada e a View não tem muito para testar em um nível de teste unitário. No entanto, o *Controller* não vai ficar disponível para testes, pois várias partes que deveriam estar dentro do *Controller* foram passadas para o *Activity*[2].

2.2.3 Model-View-Presenter

O MVP surgiu na década de 90, no qual a união da APPLE, HP e IBM fez com que esse padrão arquitetural surgisse para complementar necessidades que o padrão MVC

não cobria[20].

O MVP possibilita uma divisão melhor para camadas quando o objetivo é separar camadas superiores de camadas inferiores, ou seja, possibilita que a camada diretamente relacionada com a interface do usuário somente se comunique com a camada diretamente abaixo dela[2]. Esta abordagem separa sua aplicação em um nível de composição com 3 conjuntos de funções. Sendo eles:

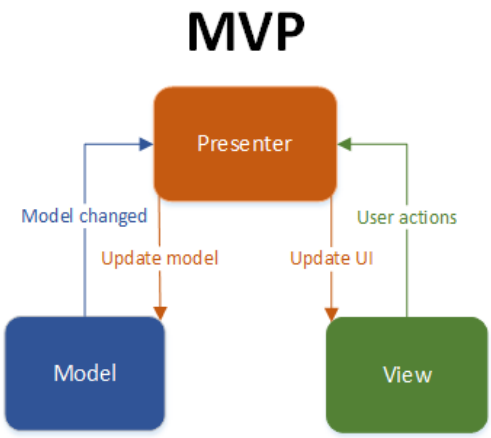

Figura 3 – Arquitetura MVP

A camada *View* é responsável por responder a saída ou entrada de dados, no qual a saída vem da camada Presenter, e a entrada vem normalmente do usuário[2]. A camada *Presenter* é responsável por responder as invocações da camada de visualização e invocações da camada de modelo e pode também invocar ambas as camadas[2]. O *Presenter* pode incluir lógica de negócio e pode ser responsável pela formatação dos dados[2]. A camada *Model*: é responsável por fornecer os dados e pode conter lógica de negócio para domínio do problema[2].

A grande vantagem do padrão arquitetural MVP é que ele descreve três níveis de abstração, o que torna mais fácil para depurar a aplicação Android[2]. A separação da lógica de negócios e lógica de acesso a dados também permite o teste que vai ser implementado. A camada *Presenter* faz referência ao *Controller* do MVC, exceto por ele não estar vinculado ao *View*, apenas a uma interface, com isto ele não mais gerencia o tráfego de solicitações recebidas, como é feito no *Controller*[2].

2.2.4 Model-View-ViewModel

O MVVM é um padrão que foi criado em 2005, por John Gossman. O MVVM assemelha-se em alguns aspectos do MVC e MVP[21]. Neste modelo o *ViewModel* não está ciente do que ocorre no *View*, mas este está informado do que ocorre no *ViewModel*.

No caso do *Model* ambos estão cientes do que ocorre em cada um. O nome se dá porque ele adiciona propriedades e operações ao *Model* para atender as necessidades do *View*, portanto ele cria um novo modelo para a visualização.

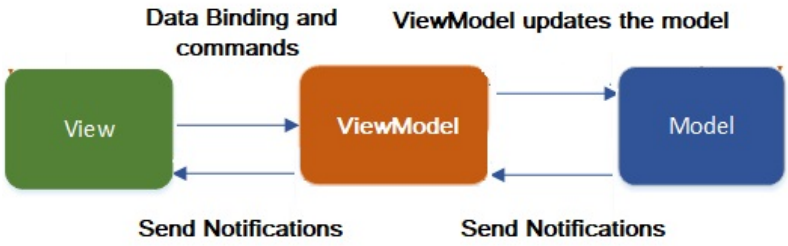

Figura 4 – Arquitetura MVVM

A camada *View* liga-se a variáveis *Observable* e ações expostas pelo *ViewModel* de forma flexível[2]. A responsabilidade da *ViewModel* no contexto do MVVM, é disponibilizar para a *View* uma lógica de apresentação[2]. A *View Model* não tem nenhum conhecimento específico sobre a *View*, ou como ela implementada, nem o seu tipo[2]. A *ViewModel* implementa propriedades e comandos, para que a View possa preencher seus controles e notifica a mesma, caso haja alteração de estado, seja através de eventos ou notificação de alteração[2]. A *ViewModel* é peça fundamental no MVVM, por que é ela quem vai coordenar as iterações da *View* com o *Model*, considerando que ambos não terem conhecimento um do outro[2]. E além de tudo isto, a *ViewModel*, também pode implementar a lógica de validação, para garantir a consistência dos dados[2]. A camada *Model* é responsável por fornecer os dados e pode conter lógica de negócio para domínio do problema.

2.2.5 FLUX

A arquitetura do Flux é usada pelo Facebook para construir seus aplicativos da Web do lado do cliente e também pode ser utilizado em aplicações moveis[22]. Ele é dividido em 3 partes: *Dispatcher, View e Store.*

A camada *View* é a interface do aplicativo, cria ações em resposta às interações do usuário[22]. A camada *Dispatcher* é um hub central através do qual passam todas as ações e cuja responsabilidade é fazer com que elas cheguem a todos os stores[22]. A camada *Store* é responsável por manter o estado de um determinado domínio de aplicativo, eles respondem a ações de acordo com o estado atual, executam a lógica de negócios e emitem um evento de mudança quando são concluídas[22]. Esse evento é usado pela visão para atualizar sua interface.

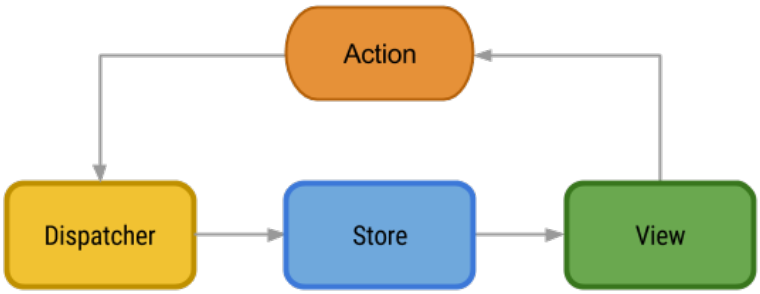

Figura 5 – Arquitetura Flux

2.3 Diferenças e semelhanças entre MVP e MVVM

Os padrões arquiteturais MVP e MVVM apresentam poucas diferenças em estrutura e responsabilidades dos componentes, mas que podem resultar em diferenças de desempenho. Os seus componentes são similares, mas com responsabilidades diferentes.

1. As diferenças entre MVP e MVVM:
    - A camada *Presenter* e a camada *View* estão em um relacionamento de 1 para 1(significa dizer que um Presenter pode mapear apenas 1 View) [15].
    - O *View* e o *ViewModel* estão em um relacionamento de 1 para muitos(significa dizer que uma *ViewModel* pode mapear muitas *Views*)[15].
    - A camada *ViewModel* substitui o *Presenter* na camada intermediária[15].
    - A camada *Presenter* mantém referências à camada *View*, já a camada *ViewModel* não[15].
    - A camada *ViewModel* envia fluxos de dados[15].
    - O *ViewModel* não sabe que o *View* está escutando[15].
2. As semelhanças entre MVP e MVVM:
    - A camada *Model* é semelhante para MVP e MVVM[15].

3 METODOLOGIA

O trabalho apresenta uma metodologia descritiva, no qual o objetivo é esclarecer ao máximo um assunto já conhecido. Neste caso foi feito uma forte revisão teórica envolvendo o objeto de estudo que é comparar e analisar as informações.

O método ATAM foi adotado nesta pesquisa afim de analisar comparativamente a qualidade do código e os detalhes na estrutura de implementações da mesma aplicação utilizando padrões arquiteturais diferentes. O método ATAM analisa quão bem a arquitetura de software satifaz objetivos particulares de qualidade. Este método é baseado no método de análise de arquitetura de software(SAAM) e é considerado uma evolução deste método. Foi utilizado uma versão adaptada para pesquisa feita pelo pesquisador T. Lou[3]. O método é formado por 4 fases e 9 passos, sendo eles:

3.1 Fase de apresentação

- Apresentação do ATAM para os stakeholders.
- Descrever os objetivos de négocio que estão motivando o desenvolvimento e quais serão os requisitos arquiteturalmente relevantes.
- Apresentar a arquitetura proposta.

3.2 Fase de investigação e análise

- Identificar, descrever e diferenciar as abordagens arquiteturais tradicionais.
- Coletar cenários que são populares e identificar as principais funções básicas.
- Coletar os requisitos, neste passo o requisito utilizado é o desempenho.

3.3 Fase de teste

- Brainstorm e priorização de cenários. Neste passo a aplicação TODO e Finder foram escolhidas para análise.
- As aplicações TODO e finder serão testadas para análise de arquitetura.

3.4 Fase de relátorio

- Apresentar resultados, neste passo é mostrado um relátorio com base nas informações recolhidas como por exemplo: estilos e cenários.

4 CENÁRIOS DE USO

Um passo necessário do método ATAM é o levantamento dos cenários de uso, ele é um conjunto de ações básicas que é necessário para completar um caso de uso, são as ações são descritas do ponto de vista da aplicação. As ações básicas que vão ser realizadas durantes os testes são as seguintes: clicar no botão, acessar armazenamento de dados e memória, entrar em uma nova tela, selecionar os itens da lista e etc. As aplicações mais conhecidas do Android geralmente são formadas por cenários de caso de uso pequenos que consistem em até cinco ações básicas.

4.1 Aplicativo Youtube Music

É uma das aplicações criadas pelo Google. Esta aplicação permite que o usuários procurem e escutem músicas. Este cenário de uso é o mais utilizado no Youtube Music. As ações básicas:

- Tela principal aberta
- Carregar músicas recentemente escutadas
- Clicar no ícone "lupa"para pesquisar uma música
- Clicar no ícone da música
- Abrir nova tela para transmitir a música

4.2 Aplicativo Tarefas

É uma das aplicações criadas pelo Google. Esta aplicação permite que o usuários registrem tarefas. As ações básicas:

- Tela principal aberta
- Clicar no botão "+"para registrar uma tarefa
- Escrever uma nova tarefa
- Clicar no botão para registrar detalhes
- Clicar no botão salvar
- Abrir nova tela para listar tarefas salvas

Outro cenário de caso de uso é editar a sua tarefa. As ações básicas:

- Tela principal aberta
- Clicar na tarefa
- Abrir nova tela para editar tarefa
- Editar nome ou descrição da tarefa
- Confirmar edição
- Abrir nova tela para listar tarefas salvas

Outro cenário de caso de uso é ver o feed do usuário. As ações básicas:

- Tela principal aberta
- Tarefas registradas na tela principal
- Rolar para ver tarefas

Outro cenário de caso de uso é deletar as tarefas. As ações básicas:

- Tela principal aberta
- Clicar na tarefa
- Abrir nova tela com descrição e nome da tarefa
- clicar no ícone da lixeira para deletar
- Voltar para tela principal

4.3 Aplicativo Medium

Esta aplicação permite que o usuário pesquise para encontrar artigos explicando vários assuntos. O cenário de uso mais popular é a pesquisa. Este cenário é composto pelas seguintes ações básicas:

- Tela principal aberta
- Digitar o texto da pesquisa
- Clicar na lupa para pesquisar
- Abrir nova tela para listar sua pesquisa

- Clicar no ícone da imagem da pesquisa para ver o texto
- Abrir nova tela para exibir a pesquisa

É importante notar que os cenários de uso mais populares consistem de ações básicas, ou seja, a maioria dos cenários de caso de uso tem como finalidade permiti que o usuário alcance os seus objetivos mais rapidamente. Isto significa dizer que os cenários de caso de uso pequenos são de interesse para pesquisadores e desenvolvedores.

5 APLICAÇÕES ANDROID PARA LEVANTAMENTO DE AMOSTRAS

Os casos de uso são implementados na forma de aplicativos Android. Estas aplicações requerem nível mínimo API Android 14, que corresponde ao Android 4.0.1 - 4.0.2 *Ice Cream Sandwich*, em torno dos 99% está o mercado que funciona com a versão 4.0 até a 9.0 que é do Android mais atual. O dispositivo utilizado para medir desempenho foi um motorola G6 play com Android 9.0. Os testes foram realizados utilizando uma ferramenta do Android studio chamada *Profile Performance* que é capaz de medir o tempo de uso dos métodos da aplicação em execução, ela também pode medir o consumo de memória e energia.

5.1 Aplicativo TODO

Este aplicativo foi desenvolvido pelo Google para mostrar diferentes abordagens arquitetônicas para o desenvolvimento de aplicativos no sistema operacional Android. Esta aplicação está disponível nos padrões MVP e MVVM.

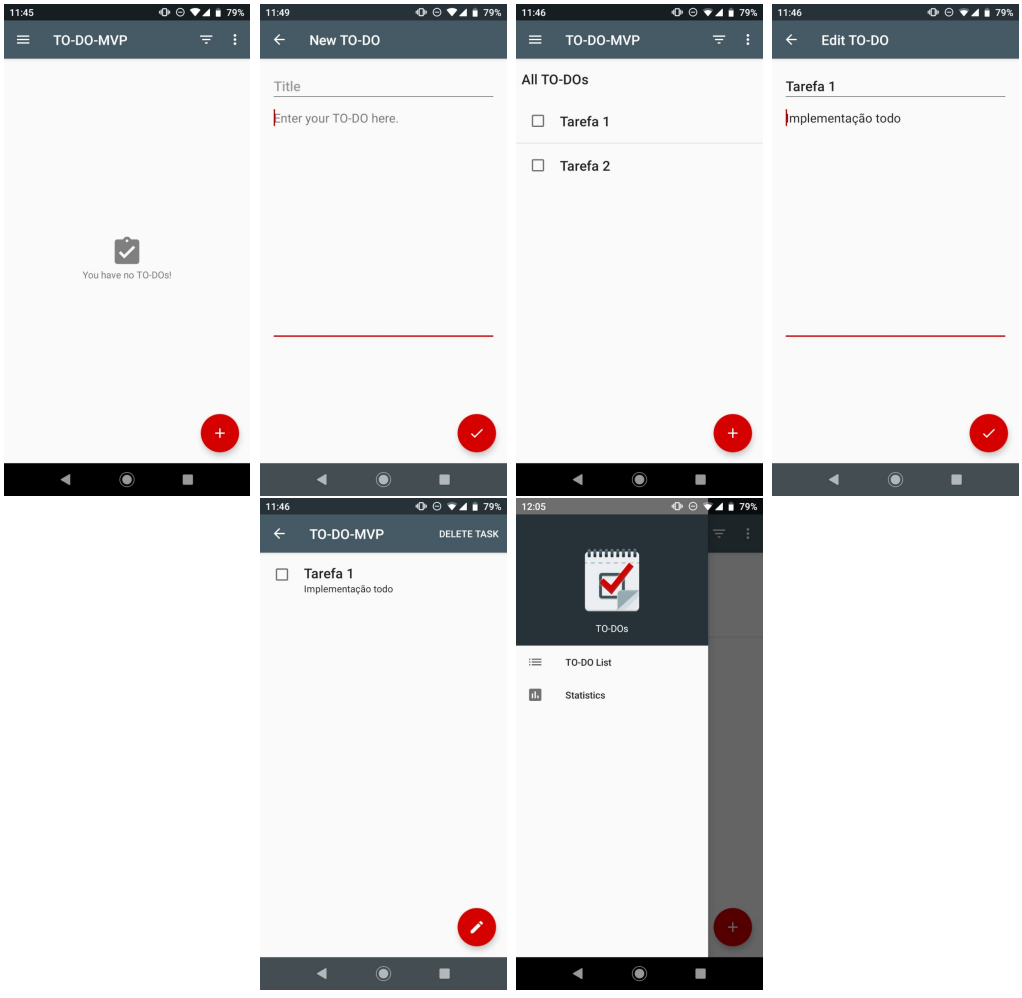

Figura 6 – Aplicativo TODO GOOGLE

A figura 6 mostra a interface do usuário das telas em caso de uso. A interface de usuário utiliza uma abordagem simples que é para focar no principal que é a arquitetura. A navegação é feita utilizando menu latetal e botões de cadastramento e visualização. O botão vermelho é utilizado para chamar atenção do usuário sobre sua funcionalidade principal que é cadastrar e editar. O menu é utilizado para mostrar estatísticas e voltar para o menu principal com a lista de tarefas.

5.2 Aplicativo Movie Finder

Este aplicativo foi desenvolvido pelo DigiGene para comparar diferentes abordagens arquitetônicas para o desenvolvimento de aplicativos no sistema operacional Android. Esta aplicação está disponível nos padrões MVC, MVP, MVVM e sem padrão.

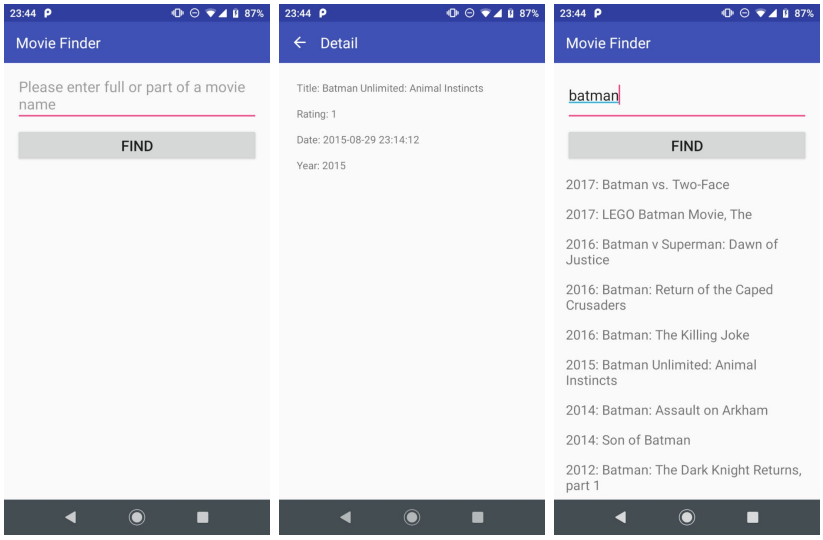

Figura 7 – Aplicativo Movie Finder

A Figura 7 mostra a interface do usuário das telas em caso de uso. A interface de usuário utiliza uma abordagem simples que é para focar no principal que é a arquitetura. A navegação é feita utilizando botões. O botão *finder* é utilizado para encontrar filmes. Clicar no nome do filme que está na lista é mostrado uma descrição.

6 DESCRIÇÃO DO EXPERIMENTO E RESULTADOS DA AVALIAÇÃO

Neste capítulo é mostrado uma descrição com base nas informações recolhidas como: estilos, cenários, questões específicas dos atributos, a árvore de utilidades, pontos de sensibilidade. A implementação do relátorio consistiu de um conjunto de casos de uso, que executa operações básicas nas aplicações Android levantadas. Os casos de uso levantados vão servir para representar os cenários de uso que foram levantados no capítulo 4.

Estes cenários tem um conjunto de ações simples como: adicionar, remover, deletar, procurar e listar. As ações que são apenas cálculo de hardware devem ter o acesso rápido, pois não depende do requisito da internet. Neste capítulo também será mostrado tabelas com medições, obtidos nos testes. Os padrões arquiteturais vão ser avaliados no contexto das medidas e analisadas para responder objetivos da pesquisa e seguir os passos do ATAM. Os padrões de arquiteturas serão avaliados de acordo com a métrica desempenho.

6.1 Inspecionar atividades de CPU para comparar desempenho

O desempenho é uma métrica muito importante para avaliar os padrões de arquitetura. Uma arquitetura com um desempenho comprometido pode levar o usuário a ter um experiência de uso ruim devido a problemas de congelamento e lentidão quando o aplicativo abre novas telas, então o desempenho deve ser considerado uma métrica principal para uma análise de arquitetura.

Profile é uma ferramenta utilizada para medir desempenho e uso de memória. A ferramenta é capaz de otimizar e mostrar problemas de desempenho presentes, animações que nunca terminanam, problemas de congelamento, alto consumo de energia[5]. Uma otimização do uso da CPU em um aplicativo Android pode deixar a aplicação mais rápida e suave[5]. O uso da ferramenta Profile do Android studio vai servir para inspecionar o uso de CPU das atividades em tempo real, com está ferramenta o tempo de execução das principais funções do aplicativo vão ser capturadas em microsegundos e milisegundos e estes dados vão ser utilizados para comparação dos padrões arquiteturais MVC, MVP e MVVM.

Os resultados de desempenho foram coletados por meio de uma estrutura de testes automatizados para aplicações Android. Os cliques e digitações foram realizados em uma velocidade constante pela ferramenta Appium, 45 medições foram feitas para calcular a média e mediana, a mediana foi realizada para evitar que valores individuais pudessem influenciar nos resultados. Os resultados, amostras, tutorial e código estão disponíveis na referência [19].

6.1.1 Descrever o experimento com aplicação TODO da Google

O experimento tem como objetivo comparar arquiteturas pelo desempenho dos casos de uso construídos com padrões arquiteturais que foram levantados. O desempenho é medido em tempo real de execução e memória total utilizada.

Os casos de uso do Google Samples vão ser utilizados para avaliar os padrões arquiteturais MVP e MVVM. As avaliações vão ser em cima das funções: criar uma tarefa, editar uma tarefa, excluir uma tarefa e listar as tarefas.

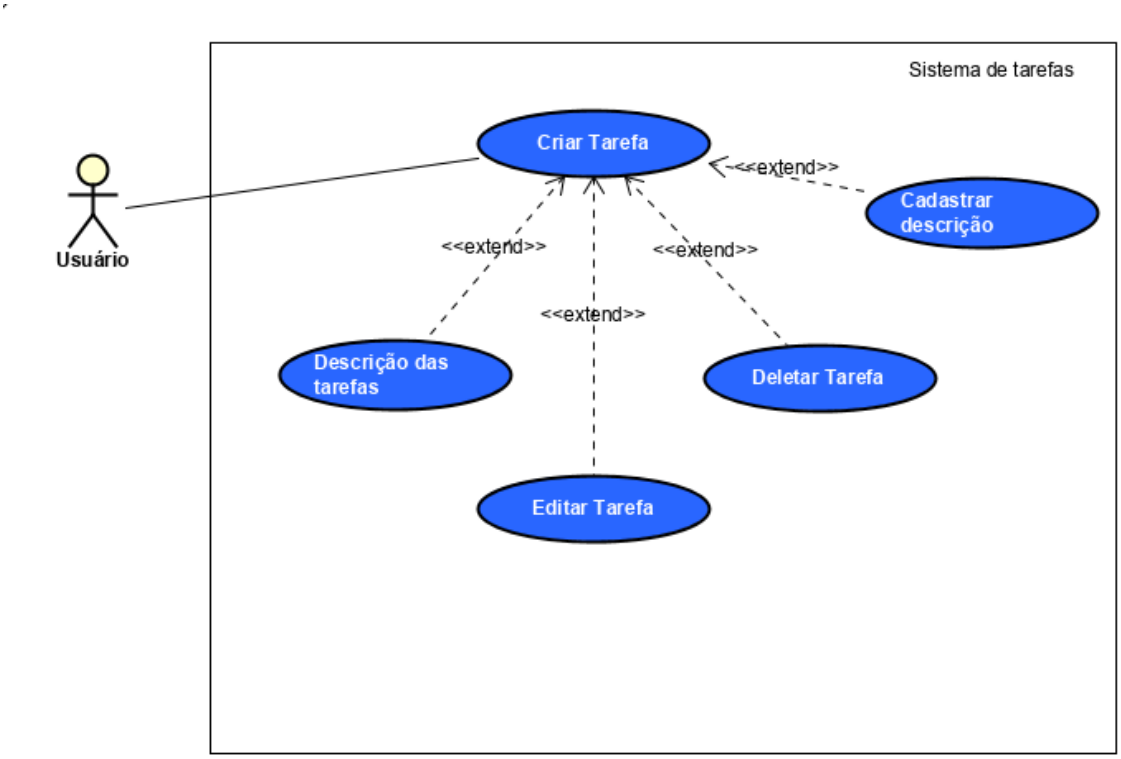

Figura 8 – Diagrama de caso de uso do TODO

6.1.1.1 Criar Tarefa

Esta seção mostra o desempenho do uso de CPU e consumo de memória em tempo real para adicionar as tarefas.

Tabela 1 – Resultado do uso de CPU para adicionar tarefa com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	14,246	14.269	14.137	14.045	5,415
MVVM	14.504	12.107	13.433	12.271	6,502

A Tabela 1 mostra a média, mediana e desvio da execução em tempo real da função criar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM.

A média e mediana foi calculada para 25 e 45 testes. Observando a média, o padrão arquitetural MVVM mostrou uma redução de 9,58% no tempo real de CPU em comparação com o MVP. Verificando a mediana a arquitetura MVVM mostrou uma redução de 12,63% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

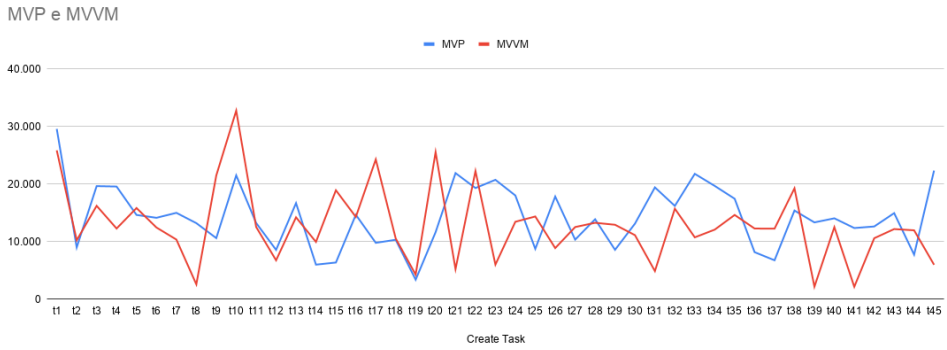

Figura 9 – Desempenho do criar tarefa em microsegundos

A Figura 9 mostra o desempenho de CPU para criação da tarefa em tempo real para os padrões arquiteturais MVP e MVVM utilizando a ferramenta de testes Appium, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android studio. O intervalo de 95% de confiança para o desempenho de CPU em criar tarefa no MVP varia entre [12,42; 15,58]. O intervalo de 95% de confiança para o desempenho de CPU em criar tarefa no MVVM varia entre [10,10; 13,90]

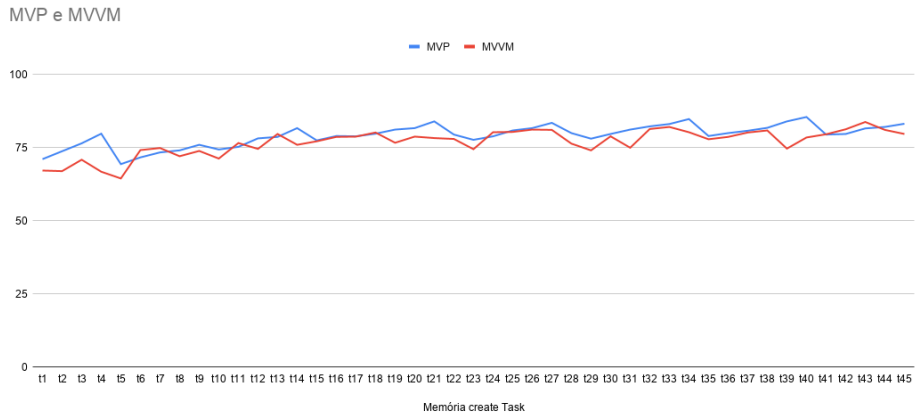

Figura 10 – Desempenho do uso de memória para criar tarefa em mb

A Figura 10 mostra o desempenho da memória para criação da tarefa em tempo real para os padrões arquiteturais MVP e MVVM que foi medido utilizando a ferramenta profile do Android studio. O intervalo de 95% de confiança para o desempenho de memória

do criar tarefa no MVP varia entre [77,92; 80,08].O intervalo de 95% de confiança para o desempenho de memória do criar tarefa no MVVM varia entre [74,71; 77,29].

Tabela 2 – Resultado do uso de memória para adicionar tarefa com o tempo medido em MB

Padrões Arquiteturais - MB	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	77.2	79,1	78.1	79.6	3.70
MVVM	74.7	76.7	75.9	78.2	4.41

Observando a média presente na tabela 2, o padrão arquitetural MVVM mostrou um redução de 2,98% no tempo real do uso de memória em comparação com o MVP. Observando a mediana, o padrão arquitetural MVVM mostrou uma redução de 1,76% no uso de memória em comparação com o MVP.Do ponto de vista de uso de memória a mudança na quantidade de MB não foi muito significativa então a qualidade do desempenho foi MVP = MVVM.

6.1.1.2 Deletar tarefa

Esta seção mostra o desempenho do uso de CPU em tempo real para deletar as tarefas.

Tabela 3 – Resultado do uso de CPU para deletar tarefa com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	10.363	10.736	11.178	10.302	4,736
MVVM	7.116	6.831	7.317	7.061	1.317

A Tabela 3 mostra a média, mediana e desvio da execução em tempo real da função deletar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes. Observando a média presente na tabela 3, o padrão arquitetural MVVM mostrou uma redução de 36,36% no tempo real de CPU em comparação com o MVP. Verificando a mediana a arquitetura MVVM mostrou uma redução de 31,43% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

A Figura 11 mostra o desempenho de CPU para deletar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM utilizando a ferramenta de testes Appium, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android studio. O intervalo de 95% de confiança para o desempenho do uso de CPU em deletar tarefa no MVP varia entre [8,62; 11,38]. O intervalo de 95% de confiança para o desempenho do uso de CPU em deletar tarefa no MVVM varia entre [5,63; 6,38].

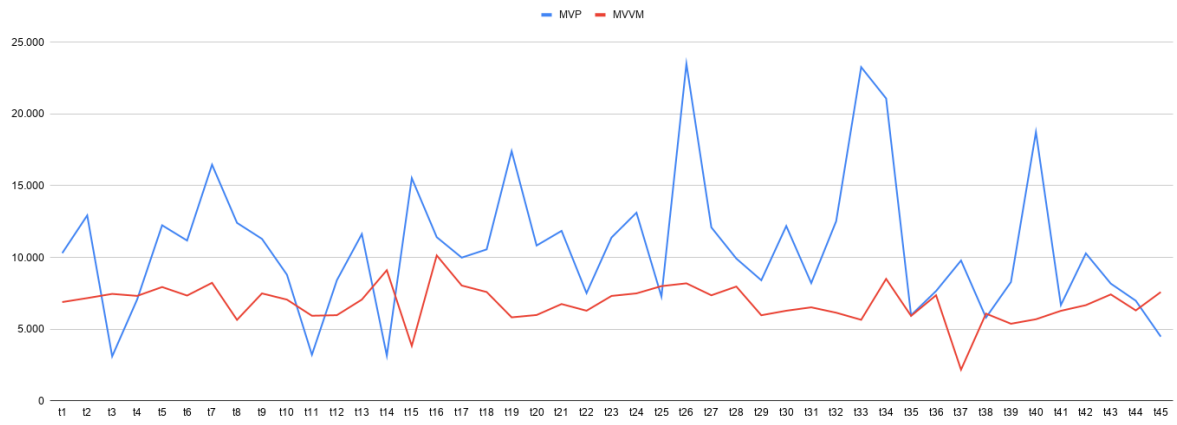

Figura 11 – Desempenho do deletar tarefa em microsegundos

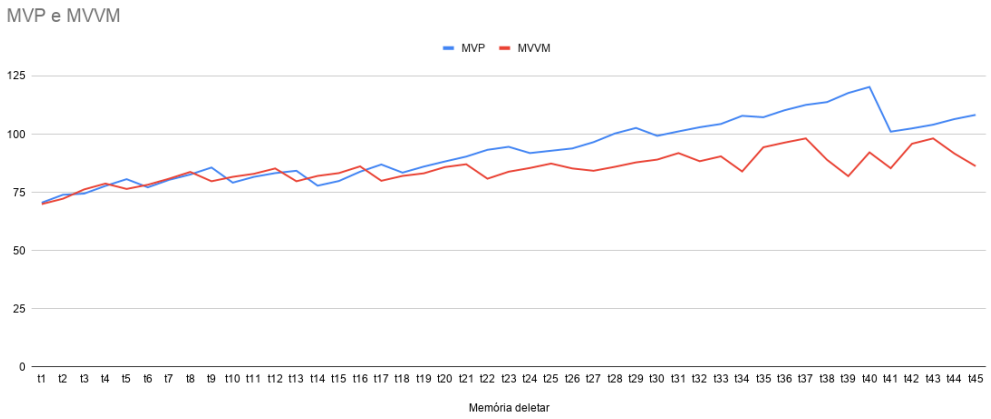

Figura 12 – Desempenho do uso de memória para deletar tarefa em mb

A Figura 12 mostra o uso de memória para deletar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM utilizando a ferramenta de testes Appium, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android studio. O intervalo de 95% de confiança no desempenho do uso de memória para deletar tarefa no MVP varia entre [89,23; 96,77]. O intervalo de 95% de confiança no desempenho do uso de memória para deletar tarefa no MVVM varia entre [83,18; 86,82].

Tabela 4 – Resultado do uso de memória para deletar tarefa com o tempo medido em MB

Padrões Arquiteturais - MB	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	83.2	93.2	83.3	92.9	12.9
MVVM	81.3	85.1	82.1	85.3	6.23

Observando a média presente na tabela 4, o padrão arquitetural MVVM mostrou um redução de 8,68% no tempo real do uso de memória em comparação com o MVP. Verificando a mediana, o padrão arquitetural MVVM mostrou uma redução de 8,18% no

uso de memória em comparação com o MVP. Do ponto de vista de uso de memória a qualidade do desempenho foi MVP < MVVM.

6.1.1.3 Listar tarefa

Esta seção mostra o desempenho do uso de CPU em tempo real para listar as tarefas.

Tabela 5 – Resultado do uso de CPU para listar tarefa com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	10.009	10.302	8.931	9.848	3,583
MVVM	11.172	9.968	9,937	9.158	5,645

A Tabela 5 mostra a média, mediana e desvio da execução em tempo real da função listar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes. Observando a média presente na tabela 5, o padrão arquitetural MVVM mostrou uma redução de 3,24% no tempo real de CPU em comparação com o MVP. Verificando a mediana a arquitetura MVVM mostrou uma redução de 7,53% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP = MVVM.

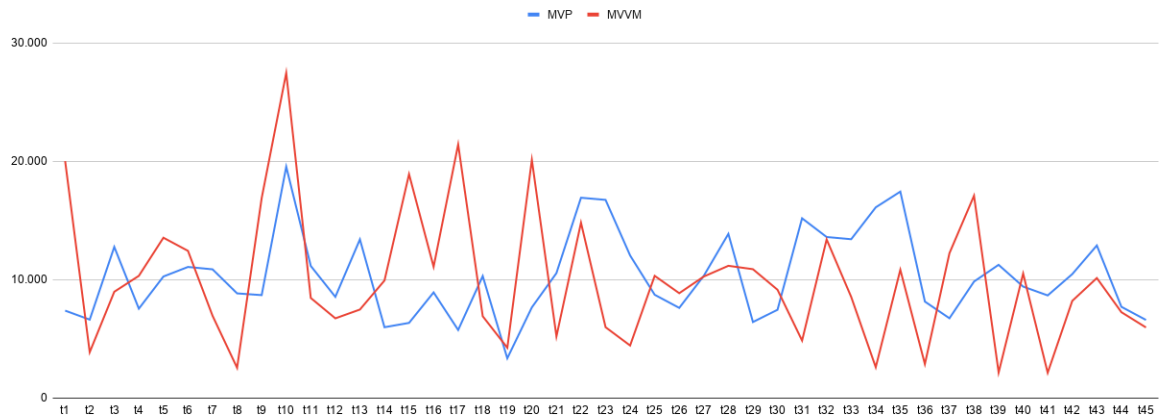

Figura 13 – Desempenho do listar tarefas em microsegundos

A Figura 13 mostra o desempenho de CPU para listar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM utilizando a ferramenta de testes Appium, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android studio. O intervalo de 95% de confiança para o desempenho do uso de CPU em listar tarefa no MVP varia entre [8,95; 11,05]. O intervalo de 95% de confiança para o desempenho do uso de CPU em listar tarefa no MVVM varia entre [7,35; 10,65].

6.1.1.4 Editar tarefa

Esta seção mostra o desempenho do uso de CPU em tempo real para editar as tarefas.

Tabela 6 – Resultado do uso de CPU para editar tarefa com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	15.830	16.544	15.957	16.713	2.351
MVVM	13.343	13.579	14.525	14.377	4.346

A Tabela 6 mostra a média, mediana e desvio da execução em tempo real da função editar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes. Observando a média presente na tabela 6, o padrão arquitetural MVVM mostrou uma redução de 17,92% no tempo real de CPU em comparação com o MVP. Verificando a mediana a arquitetura MVVM mostrou uma redução de 13,97% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

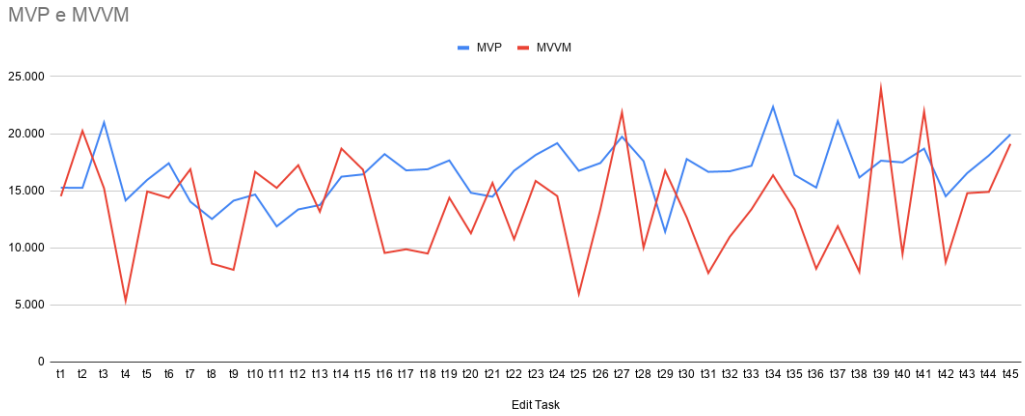

Figura 14 – Desempenho do editar tarefa em microsegundos

A Figura 14 mostra o desempenho de CPU para editar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM utilizando a ferramenta de testes Appium, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android studio. O intervalo de 95% de confiança para o desempenho do uso de CPU em editar tarefa no MVP varia entre [15,31; 16,69]. O intervalo de 95% de confiança para o desempenho do uso de CPU em editar tarefa no MVVM varia entre [11,73; 14,27].

A Figura 15 mostra o uso de memória para editar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM utilizando a ferramenta de testes Appium, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do

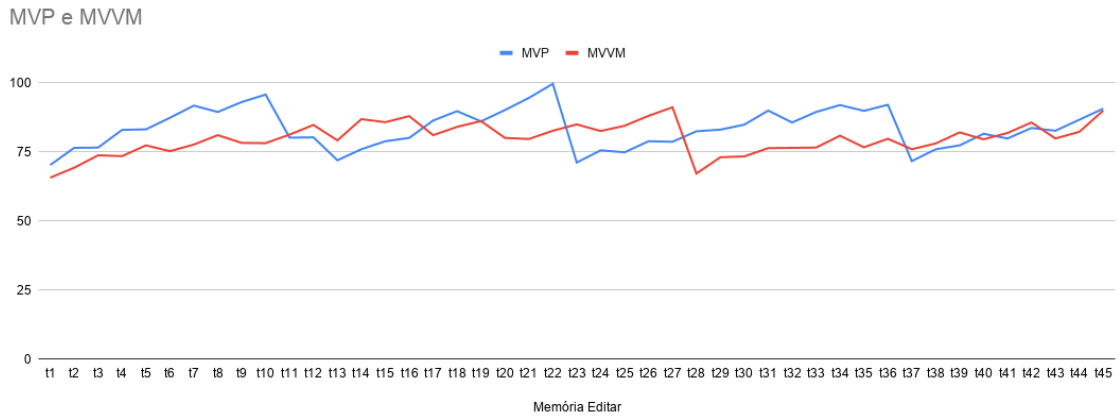

Figura 15 – Desempenho do uso de memória para editar tarefa em mb

Android studio. O intervalo de 95% de confiança para o desempenho do uso de memória em editar tarefa no MVP varia entre [80,90; 85,10]. O intervalo de 95% de confiança para o desempenho do uso de memória em editar tarefa no MVVM varia entre [77,36; 80,64].

Tabela 7 – Resultado do uso de memória para editar tarefa com o tempo medido em MB

Padrões Arquiteturais - MB	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	83.3	83.5	83.0	83.1	7.19
MVVM	80.0	79.95	81.1	79.9	5.61

Observando a média presente na tabela 7, o padrão arquitetural MVVM mostrou um redução de 4,86% no tempo real do uso de memória em comparação com o MVP. Verificando a mediana, o padrão arquitetural MVVM mostrou uma redução de 3,85% no uso de memória em comparação com o MVP. Do ponto de vista de uso de memória a mudança na quantidade de MB não foi muito significativa então a qualidade do desempenho foi MVP = MVVM.

6.1.2 Descrever o experimento com aplicação Finder

O aplicativo Movie Finder vai ser utilizado para avaliar os padrões arquiteturais MVP e MVVM. As avaliações vão ser em cima das função pesquisar.

6.1.2.1 Pesquisar filmes

A Tabela 8 mostra a média, mediana e desvio da execução em tempo real da função editar tarefa que foi implementada para os ambos padrões arquiteturais MVP e MVVM. A média e mediana foi calculada para 25 e 45 testes. Observando a média, o padrão arquitetural MVVM mostrou uma redução de 30,10% no tempo real de CPU em comparação com o MVP. Observando a mediana, a arquitetura MVVM mostrou uma

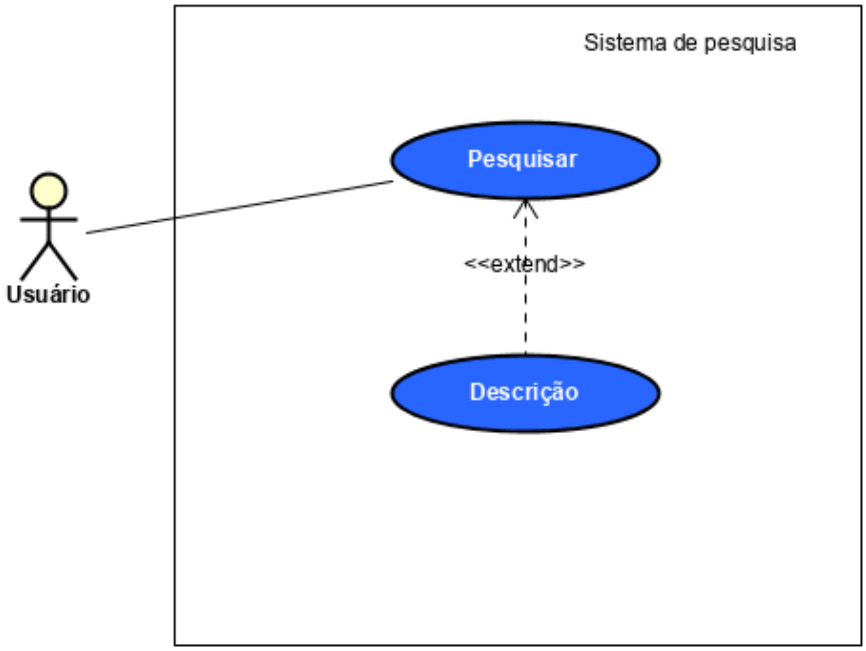

Figura 16 – Diagrama de caso de uso do Movie Finder

redução de 25,67% no tempo real de CPU em comparação com o MVP. Então a qualidade de desempenho é MVP < MVVM.

Tabela 8 – Resultado do uso de CPU para pesquisar filmes com o tempo medido em μs

Padrões Arquiteturais - μs	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	9.751	8.768	9.485	7.707	4.03
MVVM	6.599	6.128	5.919	5.728	2.73

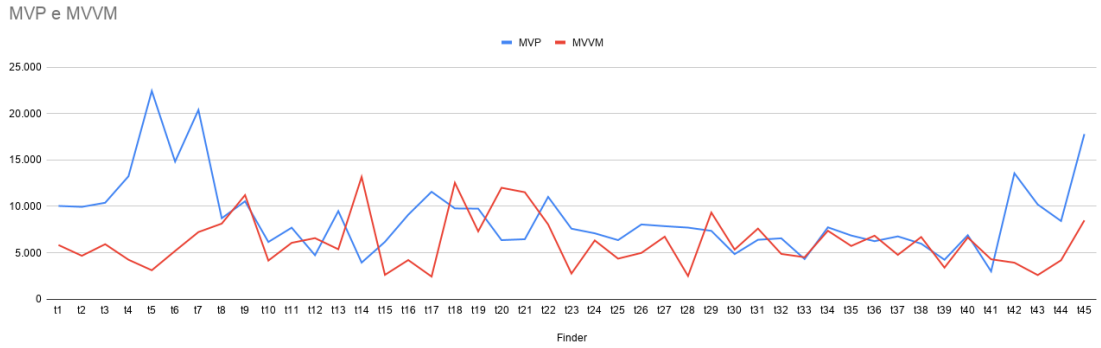

Figura 17 – Desempenho de CPU da função Pesquisar em microsegundos

A Figura 17 mostra o desempenho de CPU para pesquisar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM utilizando a ferramenta de testes Appium,

no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android studio. O intervalo de 95% de confiança para o desempenho do uso de CPU em pesquisar tarefa no MVP varia entre [6,82; 9,18]. O intervalo de 95% de confiança para o desempenho do uso de CPU em pesquisar tarefa no MVVM varia entre [5,20; 6,80].

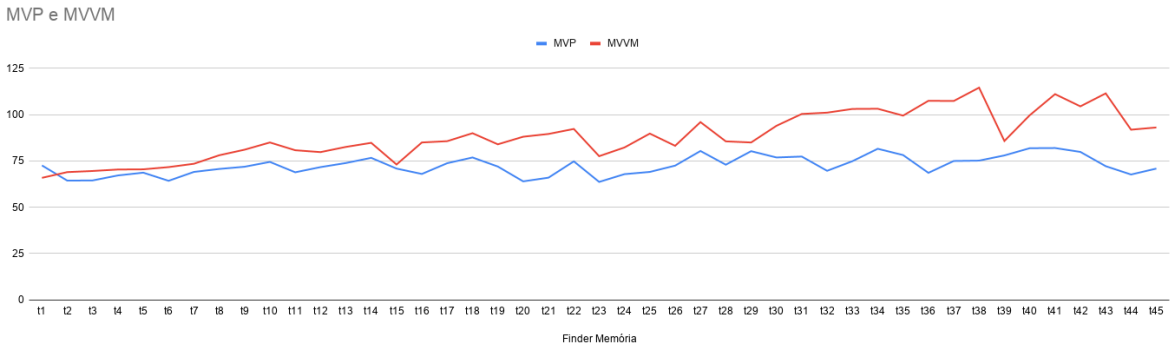

Figura 18 – Desempenho do uso de memória em MB

A Figura 18 mostra o uso de memória para pesquisar a tarefa em tempo real para os padrões arquiteturais MVP e MVVM utilizando a ferramenta de testes Appium, no qual foi automatizado as funções e foi medido o desempenho utilizando a ferramenta profile do Android studio. O intervalo de 95% de confiança para o desempenho do uso de memória em pesquisar tarefa no MVP varia entre [70,49; 73,51]. O intervalo de 95% de confiança para o desempenho do uso de memória em pesquisar tarefa no MVVM varia entre [84,32; 91,68].

Tabela 9 – Resultado do uso de memória para pesquisar filmes com o tempo medido em MB

Padrões Arquiteturais - MB	Média		Mediana		Desvio
	25 testes	45 testes	25 testes	45 testes	-
MVP	69.8	72.5	69.1	72.2	5.17
MVVM	80.0	88.4	81.1	85.7	12.6

Observando a média presente na tabela 9, o padrão arquitetural MVVM mostrou um aumento de 21,43% no tempo real do uso de memória em comparação com o MVP. Verificando a mediana, o padrão arquitetural MVVM mostrou um aumento de 18,69% no uso de memória em comparação com o MVP. Do ponto de vista de uso de memória a mudança na quantidade de MB não foi muito significativa então a qualidade do desempenho foi MVP > MVVM.

6.1.3 Qual padrão arquitetural Android leva a um melhor desempenho, MVVM ou MVP?

O desempenho é avaliado pela medição de tempo, foi análisado o tempo de execução real das funções até o seu termino. O MVVM apresentou um melhor resultado para a

métrica de desempenho dos testes em tempo real de execução, mas em algumas funções como listar e criar a diferença não foi tão significativa, do ponto de vista de memória na função pesquisar o MVVM teve um aumento de consumo e na função de deletar o MVVM teve uma redução de consumo, nos outros casos a memória se manteve constante. O ambiente de teste foi um dispositivo Android com um hardware da atualidade e ultima versão da API. Os resultados podem variar se forem aplicados em dispositivos diferentes.

7 RESUMO DOS RESULTADOS

Esta seção é responsável por apresentar um resumo dos resultados. O padrão arquitetural que apresentar uma execução no tempo real em menor tempo, ou um menor uso de memória, então deve indicar um melhor desempenho.

7.1 Desempenho do TODO

1. Criação da tarefa - Aplicação TODO
    - CPU: MVP < MVVM
    - Memória consumida: MVP = MVVM
    - Intervalo de confiança para CPU no MVP: [12,42; 15,58]
    - Intervalo de confiança para CPU no MVVM: [10,10; 13,90]
    - Intervalo de confiança para memória no MVP: [77,92; 80,08]
    - Intervalo de confiança para memória no MVVM: [74,71; 77,29]
2. Deletar a tarefas - Aplicação TODO
    - CPU: MVP < MVVM
    - Memória consumida: MVP < MVVM
    - Intervalo de confiança para CPU no MVP: [8,62; 11,38]
    - Intervalo de confiança para CPU no MVVM: [5,63; 6,38]
    - Intervalo de confiança para memória no MVP: [89,23; 96,77]
    - Intervalo de confiança para memória no MVVM: [83,18; 86,82]
3. Listar Tarefas - Aplicação TODO
    - CPU: MVP = MVVM
    - Memória consumida: MVP = MVVM
    - Intervalo de confiança para CPU no MVP: [8,95; 11,05]
    - Intervalo de confiança para CPU no MVVM: [7,35; 10,65]
4. Editar tarefa - Aplicação TODO
    - CPU: MVP < MVVM
    - Memória consumida: MVP = MVVM
    - Intervalo de confiança para CPU no MVP: [15,31; 16,69]

- Intervalo de confiança para CPU no MVVM: [11,73; 14,27]
- Intervalo de confiança para memória no MVP: [80,90; 85,10]
- Intervalo de confiança para memória no MVVM: [77,36; 80,64]

7.2 Desempenho do Finder

1. Pesquisar tarefa - Aplicação *Finder*
    - CPU: MVP < MVVM
    - Memória consumida: MVP > MVVM
    - Intervalo de confiança para CPU no MVP: [6,82; 9,18]
    - Intervalo de confiança para CPU no MVVM: [5,20; 6,80]
    - Intervalo de confiança para memória no MVP: [70,49; 73,51]
    - Intervalo de confiança para memória no MVVM: [84,32; 91,68]

8 CONCLUSÃO E TRABALHOS

O TCC tem como objetivo definir se o padrão arquitetural MVVM com os cenários levantados pode ser melhor que o MVP que é mais conservador no mercado. Para encontrar os resultados foi preciso estudar conceitos básicos do padrão arquitetural, sua estrutura e definir semelhanças e diferenças.

O tempo necessário para executar suas funções e o uso de memória são duas medidas para mostrar que existe um desempenho melhor. Executando as aplicações testadas com automação para garantir que os fatores com inserção de textos e cliques sejam os mesmose, em seguida, foi utilizado uma ferramenta de perfomance do Android studio para medir o tempo de execução real de cada uma das funções presentes. O MVVM apresentou os melhores resultados em termos de CPU, do ponto de vista de memória, apenas na pesquisa apresentou um consumo mais alto de memória. Os casos de uso do padrão arquitetural MVP é necessário mais tempo para ser executado e em alguns casos necessita de mais memória.

Qual é o padrão arquitetural que pode ter o melhor desempenho, MVVM ou MVP? O padrão arquitetural MVVM teve os melhores resultados nas métricas de CPU e memória, ou seja, o MVVM leva um melhor desempenho do que o padrão MVP, levando em consideração os cenários e o aparelho que foi usado.

8.1 Trabalhos futuros

Durante o desenvolvimento do TCC foi percebido pontos que podem ser melhorados e novos pontos para acrescentar na pesquisa que são de interesse na comunidade do Android, são eles:

- Aumentar o número de amostras para obter um intervalo de confiança melhor.
- Levantar mais cenários poderia dar melhores resultados e ter um noção maior sobre qual padrão arquitetural pode abranger mais cenários que simulam os aplicativos nos dias de hoje.
- Obter aplicações mais complexas implementadas com diferentes arquiteturas, mas com mesma função.
- Utilizar diferentes padrões GOF para analisar o comportamento do desempenho e a facilidade para produzir o código com diferente padrões arquiteturais.
- O TCC poderia incluir mais métricas para comparar.

- Utilizar diferentes dispositivos Android para saber o desempenho deles diante dos padrões arquiteturais.
- Investigar outros padrões arquiteturais e poder encontrar melhores resultados. Novas arquiteturas estão surgindo ao longo dos anos e comparar elas com arquiteturas tradicionais pode criar uma pesquisa forte com grandes resultados na comunidade.

REFERÊNCIAS

[1] LINHARES. Paulo. Android Architecture Components (Componentes de Arquitetura Android). 2018. Disponível em:< https://medium.com/iterative/android-architecture-components-componentes-de-arquitetura-android-6c6be539f47e>. Acesso em: 19 ago. 2019.

[2] NUNES, Felipe. Android MVC x MVP x MVVM qual Pattern utilizar — Parte 1. 2017. Disponível em:<https://medium.com/@FilipeFNunes/android-mvc-x-mvp-x-mvvm-qual-pattern-utilizar-parte-1-3defc5c89afd>. Acesso em: 19 ago. 2019.

[3] LOU, T. A comparison of Android Native App Architecture MVC, MVP and MVVM. 2016. Disponível em:<https://pure.tue.nl/ws/portalfiles/portal/48628529/Lou_2016.pdf>. Acesso em: 19 ago. 2019.

[4] Android Developers, 2019. Disponível em:<https://developer.android.com/docs>. Acesso em: 19 ago. 2019.

[5] Profile your app performance | Android Developers. 2019. Disponível em: <https://developer.android.com/studio/profile/>. Acesso em: 19 ago. 2019.

[6] Testar o aplicativo | Android Developers. 2019. Disponível em: <https://developer.android.com/studio/test>. Acesso em: 19 ago. 2019.

[7] Fragmentos | Android developers. 2019. Disponível em: <https://developer.android.com/guide/components/fragments?hl=pt_br>. Acesso em: 19 ago. 2019.

[8] android.app | Android developers. 2019. Disponível em: <https://developer.android.com/reference/android/app/package-summary>. Acesso em: 19 ago. 2019.

[9] Fragmentos | Android developers. 2019. Disponível em: <https://developer.android.com/guide/components/fragments?hl=pt_br>. Acesso em: 19 ago. 2019.

[10] HUMENIUK, Vladyslav. Android Architecture Comparison: MVP vs. VIPER. 2019. Disponível em:<http://lnu.diva-portal.org/smash/get/diva2:1291671/FULLTEXT01.pdf>. Acesso em: 19 ago. 2019.

[11] CAVALCANTE, Henri. Flux — A arquitetura do Facebook para desenvolvimento FrontEnd. 2015. Disponível em:<https://medium.com/@henricavalcante/flux-a-arquitetura-do-facebook-para-desenvolvimento-frontend-2bf7192c8f77>. Acesso em: 19 ago. 2019.

[12] CARVALHO, Bruno. Appium: automação para apps mobile - assert(QA) - Medium. 2018. Disponível em:<https://medium.com/assertqualityassurance/appium-automação-para-apps-mobile-a256db64a27d>. Acesso em: 19 ago. 2019.

[13] MVP Android. 2017. Disponível em: <https://www.thiengo. com.br/mvp-android>. Acesso em: 19 ago. 2019.

[14] SOUTO, Thiago. Arquiteturas em Android : MVVM + Kotlin + Android Architecture Components (Databinding, Lifecycle, LiveData, Room). 2019. Disponível em:<https://medium.com/android-dev-br/arquiteturas-em-android-mvvm-kotlin-android-architecture-components-databinding-lifecycle-d5e7a9023cf3>. Acesso em: 19 ago. 2019.

[15] CHUGH, Anupam. Android MVVM Design Pattern - JournalDev. 2018. Disponível em:<https://www.journaldev.com/20292/android-mvvm-design-pattern>. Acesso em: 19 ago. 2019

[16] Android - Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Android.>. Acesso em: 19 ago. 2019.

[17] Activity | Android developers. 2019. Disponível em: <https://developer.android.com/reference/android/app/Activity>. Acesso em: 19 ago. 2019.

[18] CICLO DE VIDA DE UN ACTIVITY - ANDROID STUDIO. Disponível em: <https://www.youtube.com/watch?v=nv_J-VGnQVo>. Acesso em: 19 ago. 2019.

[19] Tutorial para execução dos testes, resultados e amostras. Disponível em: <https://github.com/alexandrefcesar/tcc-codigo-amostras>. Acesso em: 19 ago. 2019.

[20] Model-View-Presenter - Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Model-view-presenter>. Acesso em: 19 ago. 2019.

[21] Cadu. Entendendo o pattern Model View ViewModel MVVM. 2010. Disponível em: <https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>. Acesso em: 19 ago. 2019.

[22] Flux Architecture on Android. 2015. Disponível em: <https://github.com/lgvalle/android-flux-todo-app>. Acesso em: 19 ago. 2019.