Universidade Tecnológica Federal do Paraná

Departamento Acadêmico de Informática

Curso de Bacharelado em Sistemas de Informação

Caio Luiz Salgado
Emanuel Atanázio de Souza

Análise de consumo de energia em aplicações móveis, um estudo de caso.

Trabalho de Conclusão de Curso

CURITIBA
2019

Caio Luiz Salgado
Emanuel Atanázio de Souza

Análise de consumo de energia em aplicações móveis, um estudo de caso.

Proposta para o desenvolvimento do Trabalho de Conclusão do Curso de Bacharelado em Sistemas de Informação, apresentado à UTFPR como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Orientador:
Prof. Doutor Paulo Roberto Bueno

CURITIBA
2019

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Coordenação do Curso de Bacharelado em Sistemas de Informação

TERMO DE APROVAÇÃO

“ANÁLISE DE CONSUMO DE ENERGIA EM APLICAÇÕES MÓVEIS, UM ESTUDO DE CASO”

por

“CAIO LUIZ SALGADO e EMANUEL ATANÁZIO DE SOUZA”

Este Trabalho de Conclusão de Curso foi apresentado no dia **11** de **julho** de **2019** como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação na Universidade Tecnológica Federal do Paraná - UTFPR - Câmpus Curitiba. O(a)(s) aluno(a)(s) foi(ram) arguido(a)(s) pelos membros da Banca de Avaliação abaixo assinados. Após deliberação a Banca de Avaliação considerou o trabalho _____.

_____ **<Prof. Paulo Roberto Bueno>** (Presidente - UTFPR/Curitiba)	_____ **<Prof. Marcelo Mikosz Goncalves>** (Avaliador 1 - <Instituição>)
_____ <**Prof. Eteocles da Silva Cavalcanti>** (Avaliador 2 - Instituição)	_____ **Profa. Leyza Baldo Dorini** (Professor Responsável pelo TCC – UTFPR/Curitiba)
_____ **Prof. Marcelo Mikosz Goncalves** (Coordenador(a) do curso de Bacharelado em Sistemas de Informação – UTFPR/Curitiba)	

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso.”

RESUMO

Neste trabalho analisamos a eficiência energética em duas abordagens de aplicações móveis utilizadas no mercado de desenvolvimento de sistemas e aplicações móveis. A primeira construída sob a arquitetura cliente servidor em que o aplicativo faz requisições a um servidor remoto e o mesmo processa as respostas e devolve para o cliente. Já na segunda, deixamos sob responsabilidade da aplicação cliente se conectar ao banco de dados e processar as regras de negócio. Para cada uma das técnicas implementadas, são efetuados conjuntos de testes de maior à menor carga para obter dados factíveis, podendo então utilizar esses dados de consumo de bateria como objeto de análise e comparação.

Palavras chaves: Computação móvel. Consumo de energia. Aplicações Móveis.

ABSTRACT

In this research we analyze the energy efficiency in two mobile applications widely used in the mobile systems and applications development market. The first one built under the client server architecture where the application makes requests to a remote server and it processes the responses and returns them to the client. The second approach is the application responsible for connecting to the database and processing business rules. For each of the techniques implemented, forms of testing was provided, since larger sets of tests until lower charges, and then was used these battery consumption data as analysis and comparison object.

Keywords: Mobile computing. Mobile application. Power consumption.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HTTP – HyperText Transfer Protocol

SQL – Structured Query Language.

REST – Representational State Transfer

URL – Uniform Resource Locator

NA – Número da amostra

ME – Média das amostras

MA – Mediana das amostras

4G – Quarta geração

ADB – Android Debug Bridge

SOAP – Simple Object Access Protocol

RPC – Remote Procedure Call

MAH – Miliamperes

CPU – Central Processing Unit

URI – Uniform Resource Identifier

PDA – Personal Digital Assistant

SUMÁRIO

1 INTRODUÇÃO

Os *smartphones* e tablets permitem que as pessoas tenham mais poder computacional em suas mãos do que com os *desktops* alguns anos atrás. A computação móvel encontra diversas limitações e apesar de o processamento, armazenamento e capacidade de comunicação desses dispositivos estarem aumentando ao longo dos anos a durabilidade das baterias desses dispositivos não acompanham o mesmo progresso (LIU, 2008). A fonte de energia ainda continua sendo um recurso escasso e a eficiência no seu consumo por aplicações móveis é um fator importante na satisfação do usuário de *smartphone*.

Existem pesquisas a respeito de reduzir o consumo de energia em dispositivos móveis das mais variadas formas, seja por meio de otimização do *hardware*, sistema operacional, *firmware* ou *software*. Vários estudos têm sido realizados para identificar de que forma os aplicativos móveis poderiam ser construídos para que o consumo de energia fosse menor, tanto por questões ambientais, tanto para satisfazer os usuários que desejam recarregar com menos frequência seus celulares.

Neste trabalho analisamos o consumo de energia em dispositivos móveis, especialmente *smartphones*, comparando duas abordagens comumente utilizadas no mercado de desenvolvimento sistemas e aplicações móveis. Implementamos duas aplicações móveis e fizemos a análise do consumo de energia de cada uma. Ambas têm as mesmas funcionalidades e foram desenvolvidas em linguagem Java para dispositivos com sistema operacional *Android*, como sendo um dos mais populares sistemas operacionais (NAGATA, 2012). Considerando diferentes abordagens de arquiteturas disponíveis no mercado, selecionamos duas, a primeira construída sob a arquitetura cliente servidor em que o aplicativo faz requisições a um servidor remoto e o mesmo processa as respostas e devolve ao cliente. Já na segunda, deixamos sob responsabilidade da aplicação cliente se conectar ao banco de dados e processar as regras de negócio. Para cada uma das técnicas utilizadas, são fornecidas três conjuntos de teste, uma enviando cem requisições, outra enviando quinhentas e uma última enviando mil, para efetuar testes de maior carga e assim obter dados experimentais, podendo então utilizar esses dados de consumo de bateria como objeto de análise e comparação.

Nosso objetivo é analisar se o custo energético de executar as tarefas localmente é superior ao custo de executá-las remotamente, pois neste caso, existe um custo extra de comunicação com o *web service* que pode acrescer seu consumo e ser superior ao custo de execução local. E também e contribuir com a comunidade científica e de desenvolvimento de *softwares* móveis na escolha de uma abordagem mais eficientes em termos energéticos para a sua aplicação.

1.1 ESTRUTURA DO DOCUMENTO

O próximo capítulo contém a metodologia a ser empregada, assim como a classificação da mesma, detalhamento das etapas envolvidas e quais tecnologias foram adotadas. A terceira seção, abordará a revisão bibliográfica e o referencial teórico, assim como os trabalhos relacionados. A quarta seção apresenta os resultados, considerações finais e discussões. O quinto contém a conclusão e projetos futuros. Findando com a seção de referências bibliográficas utilizadas no projeto e os anexos contendo o código fonte e diagramas das aplicações desenvolvidas.

2 METODOLOGIA

Esse capítulo compreende o detalhamento do estudo realizado, a motivação, as etapas realizadas em ordem cronológica, procedimentos, materiais, métodos e projetos futuros.

2.2 MOTIVAÇÃO

O tema e área escolhida para este trabalho foi a computação móvel e ubíqua, que ao longo dos últimos anos tem expressado marcos de grande popularidade[1]. Segundo Coulouris (COULOURIS, 2013), computação móvel e ubíqua é a comunicação entre dispositivos portáteis por meio de sistemas distribuídos. Podendo ser eles *notebooks*, *smartphones*, *pagers*, assistentes digitais pessoais (PDAs), *smartwatches* e até mesmo máquinas de lavar, carros, geladeiras e etc.

Maior parte do acesso à internet hoje é realizado a partir de dispositivos móveis. A Google revelou (DISCHLER, 2015) que em 2015 a maioria das buscas e pesquisas feitas eram via dispositivos móveis. Segundo o portal de estatísticas Statista[1] o número de usuários de celulares no mundo passará a marca de cinco bilhões até 2019.

Esses dados demonstram grande relevância e importância da computação móvel no cotidiano de cada ser humano e indica uma linha de tendência para o mercado de *software*. Por outro lado, há uma extrema necessidade de cada vez mais fazer com que os dispositivos móveis tenham suas baterias duráveis por mais tempo (LIU, 2008).

Definido isto, a principal motivação deste trabalho é responder algumas perguntas que não encontramos as respostas na literatura científica relacionadas ao desenvolvimento de aplicativos móveis e seu consumo de energia e bateria.

Considerando diferentes abordagens disponíveis no mercado, selecionamos duas, a primeira construída sob a arquitetura cliente servidor em que o aplicativo faz requisições ao servidor e o mesmo processa as respostas. E na segunda abordagem, deixamos sob responsabilidade da aplicação cliente se conectar ao banco de dados, tratá-los e processar as regras de negócio.

---

1 “Number of mobile phone users worldwide from 2013 to 2019 (in billions)”. . Acessado em: 04/11/2017.

As perguntas que motivaram essa pesquisa são:

1. O que traz mais economia energética no contexto móvel, executar tarefas remotamente via web service ou executá-las localmente sem acesso aos protocolos de acesso à internet?
2. Como o tráfego de dados e o tempo de resposta se comportam aplicados a essas duas técnicas e implicam no consumo de bateria?
3. Qual é a diferença energética entre as duas abordagens quando se efetuam grandes quantidades de chamadas encadeadas, por exemplo, cem, quinhentas e mil chamadas?
4. Em qual cenário houve maior diferença de economia energética entre as duas abordagens?
5. Qual abordagem é mais eficiente em termos de consumo de energia?

Então, elaboramos a seguinte hipótese:

Parte-se da hipótese de o custo energético de executar as tarefas localmente é inferior ao custo de executá-las remotamente, pois existe um custo extra de comunicação com o *web service* que acresce ao seu consumo, sendo assim superior ao custo de execução local.

2.3 CLASSIFICAÇÃO DA PESQUISA

O presente trabalho, trata-se de uma pesquisa **aplicada**, **quantitativa** e **exploratória.**

A seguir será apresentada detalhadamente a classificação do objeto de estudo deste trabalho, conforme os autores Gil e Antonio Carlos (GIL, 1991) e Silva e Menezes (SILVA, 2001).

Do ponto de vista da natureza da pesquisa, consideramos uma pesquisa **aplicada**, pois envolve a aplicação prática de conhecimentos resultantes desta e outras pesquisas posteriores, como a eletrônica, computação móvel, dentre outras.

A respeito da forma de abordagem do problema, ela é **quantitativa,** pois fazemos uso de técnicas estatísticas no experimento científico, por meio da coleta de dados, amostras e análise da população de dados de maneira objetiva.

É **exploratória** pois envolve um estudo aprofundado de diversas técnicas para a construção dos artefatos e também por meio de referencial teórico e estudo bibliográfico para a compreensão do problema e do estudo de caso.

2.4 ETAPAS

Para atingir os objetivos propostos e testar as hipóteses experimentais formuladas, realizou-se um estudo em quatro fases:

A fase inicial foi de definição do tema e desenvolvimento do modelo de projeto. Onde foi delimitado o escopo, objeto de estudo, os objetivos e o referencial teórico. Ao fim dessa fase, realizou-se a apresentação da proposta e com aprovação, houve algumas adequações feitas no mesmo.

Como segunda fase, houve pesquisa de tecnologias disponíveis e viáveis que fossem de encontro com a temática e objetivos do projeto.

Já na terceira fase, foi aplicado os conhecimentos adquiridos e desenvolvido duas aplicações móveis. Para auxiliar na coleta dos dados, foi automatizado feito automação dos testes para tornar o dado estatístico consistente.

E por fim, análise dos experimentos e avaliações dos resultados.

2.5 MATERIAIS E MÉTODOS

A seguir será descrito os procedimentos feitos no trabalho, o manejo utilizado, o controle das condições experimentais, as técnicas de coleta de dados, as avaliações realizadas e as variáveis analisadas.

2.5.1 Materiais

Para o desenvolvimento desse experimento, foram necessárias a implementação de dois sistemas de *software*. Um deles trata-se de um aplicativo para dispositivos móveis

com sistema operacional Android. Esse *app* tem objetivo de auxiliar na interação e facilitar a coleta de dados por meio das telas e interface construídas no app. O mesmo também fará comunicação com uma outra aplicação construída ao longo desse trabalho, que é do tipo *web service* e tem o propósito de prover as informações e/ou dados para o app. Não existe nenhum conceito negocial específico no aplicativo. A escolha dos cenários, apesar de ilustrar casos da vida real, foi feita somente a fim de coletar informações estatísticas, portanto não se conectam entre si em um fluxo contínuo para o usuário.

Um dos pontos chave desse experimento é o acesso a base de dados e como cada abordagem o faz. Escolhemos MariaDB como banco de dados, por ser um banco leve, gratuito, de código aberto e baseado em linguagem SQL. Cogitamos utilizar também o Sql Server rodando na nuvem da Microsoft, contudo não foi viável e os créditos expiraram durante o processo do trabalho. O acesso a essa base é feito de duas maneiras distintas, diretamente da aplicação cliente, utilizando um pacote de classes oriundos do Java para estabelecer uma conexão na internet via protocolo TCP/IP e recuperar as informações da base de dados ou acessando uma aplicação do tipo *web service*, que segundo Papazoglou (PAPAZOGLOU, 2013) nada mais é que um tipo específico de serviço identificado por um URI, cuja descrição do serviço e transporte utilizem padrões abertos da Internet.

A aplicação cliente foi desenvolvida para rodar em um único tipo de sistema operacional e portanto essa análise só será válida para parte dos aparelhos existentes no mundo, aqueles baseados em sistemas operacionais Android. Os aparelhos utilizados são pessoais, tanto o notebook que hospedou o *web service*, como o celular que hospedou a parte cliente.

Abaixo temos uma tabela descrevendo as principais informações dos aparelhos:

Tabela 1 - Recursos de hardware

Aparelho	Característica	Valor
Celular	Bateria (capacidade máxima)	3.300 mAh
Celular	Versão do Sistema Operacional	8.1.0
Celular	Versão Knox	3.2 (API 26)

Celular	Modelo	Samsung J7 Prime
Notebook	Memória RAM	8GB
Notebook	Processador	Intel(R) Core(TM) i5-4300U CPU @ 1.90GHz, 2494 Mhz, 2 Núcleo(s), 4 Processador(es) Lógico(s)
Notebook	HD/SSD	SSD 256 GB
Notebook	Versão do Sistema Operacional	Windows 10
Notebook	Modelo	LENOVO

Fonte: Autoria própria

Tabela 2 - Recursos de software

Software	Utilidade
Eclipse	IDE utilizada para o desenvolvimento do webservice.
Android Studio	IDE utilizada para o desenvolvimento do aplicativo onde foram executados os testes.
MariaDB	Servidor de banco de dados.
Github	Repositório do código dos sistemas desenvolvidos e controle de versão.
JDK 8	Kit de desenvolvimento necessário para implementação do webservice e da aplicação móvel, ambas desenvolvidas utilizando a linguagem Java.
Spring 4	Framework utilizado para desenvolvimento web, especificamente destinado a construção do webservice.
Cmder	Emulador de terminal Linux utilizado para rodar comandos Unix em uma máquina Windows, utilizado para execução dos comandos do ADB.
Postman	Ferramenta utilizada para testes de funcionamento do webservice, antes da execução do teste para coleta das amostragens.
Notepad++	Ferramenta de visualização e edição de texto utilizada para análise dos arquivos de log coletados nas amostragens.

Fonte: Autoria própria

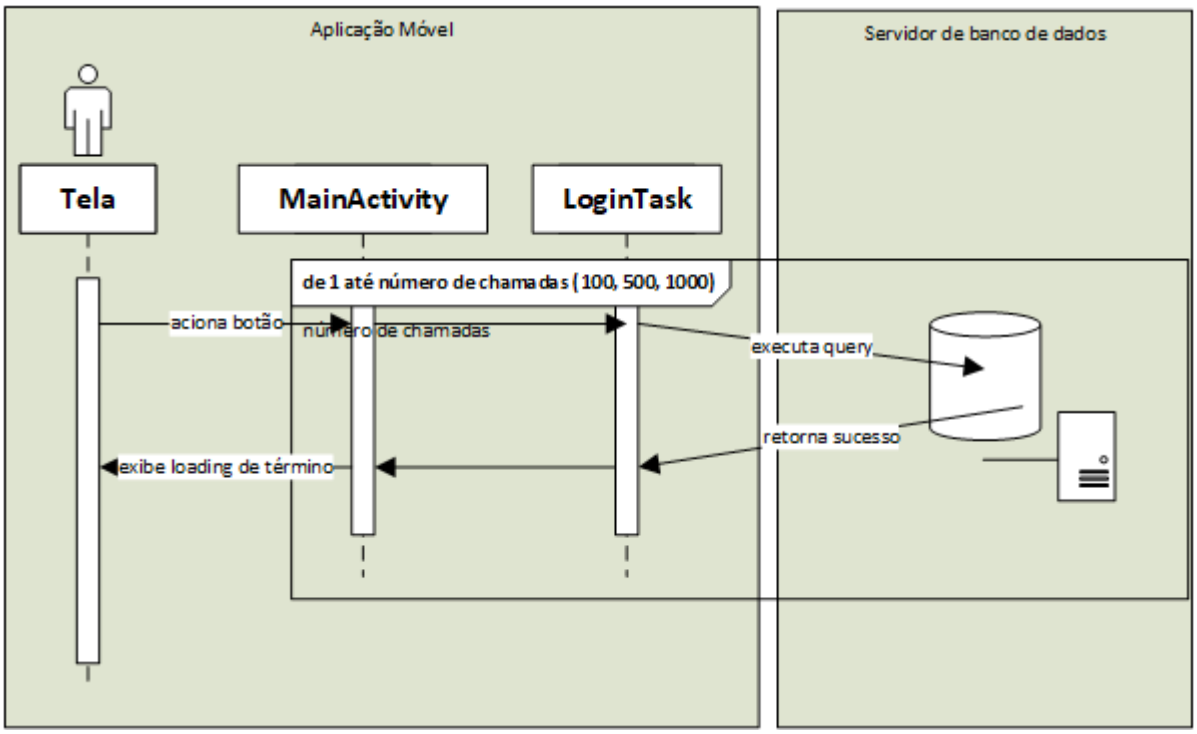
Na tabela 1 se encontram as configurações dos dispositivos de hardwares utilizados neste trabalho. Não foi utilizado nenhum hardware especializado, tratam-se de desktop e celular comuns de mercado e acessíveis, não são as melhores configurações possíveis e também não estão obsoletas.

Na tabela 2 se encontram softwares utilizados para realizar a pesquisa. O desenvolvimento do *web service* foi feito sob um *framework* baseado na linguagem Java, denominado Spring. Esse *framework* de código aberto e gratuito provê, de forma intuitiva e rápida, integrações com outras bibliotecas que fazem acesso à base de dados e expõe serviços na *internet* no padrão *restfull*.

O desenvolvimento da parte cliente foi feito utilizando a linguagem Java, codificado para cada um dos cenários duas formas de acesso, que é o objeto de comparação deste estudo. A divisão entre essas duas formas de acesso pode ser verificada no código fonte através da separação dos pacotes. O pacote que contém as chamadas diretas a base de dados, tem o nome de tarefas, enquanto que o pacote que contém as chamadas ao *webservice* está no pacote *service*.

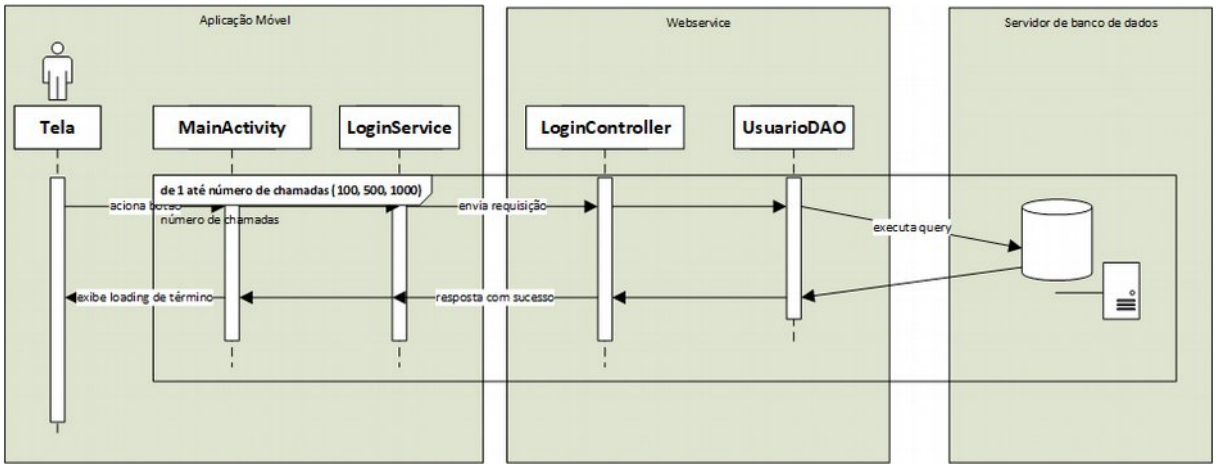
Abaixo temos dois diagramas de sequência que exemplificam a interação do aplicativo com o servidor de banco de dados e com o servidor web que contêm o *webservice*:

Imagem 1 - Técnica de acesso direto ao banco de dados.



Fonte: autoria própria

Imagem 2 - Técnica de acesso a base via webservice.



Fonte: autoria própria

A imagem 1 representa a primeira técnica implementada em que o usuário ao acionar uma opção entre 100, 500 e 1000 mil chamadas na tela, a aplicação cliente fará o acesso direto ao banco de dados. Após a execução da query no banco, a aplicação cliente mostra o status da execução para o usuário final.

A imagem 2 contém a segunda técnica implementada, que semelhante a primeira também possui uma tela com botões com 100, 500 e 1000 chamadas. Ao ser acionada alguma das opções a aplicação cliente faz a chamada para o *webservice* que se encarrega de se conectar com o banco de dados. Sendo assim, após o término da quantidade de execuções necessárias o *webservice* retorna uma resposta para a aplicação cliente com o status da operação realizada.

Para a coleta dos dados foram necessárias outras duas aplicações auxiliares, uma chamada ADB (Android Debug Bridge) e outra Docker. A primeira serve para comunicar com o sistema operacional e dizer para ele obter informações de mais baixo nível a respeito da aplicação, são elas, no nosso caso, dados da bateria e dados de internet. A segunda é uma tecnologia responsável por rodar aplicações independente do que seja necessário para configurá-las na sua máquina local. É como um sistema de arquivos pré-definidos que contém uma camada fina de bibliotecas e binários necessários para fazer seu aplicativo funcionar e até mesmo o código do aplicativo. A isso é dado o conceito de imagem Docker (ANDERSON, 2013). Cada aplicação tem uma imagem que pode ser compartilhada com qualquer outro usuário da rede, essa imagem contém todos os elementos necessários para subir essa determinada aplicação. O Docker em si é uma ferramenta auxiliar para rodar uma aplicação *web* que interpreta um arquivo de texto com

os dados da bateria coletados através do ADB e mostrar em formato gráfico, mas também com números em uma página na internet.

2.5.1.1 Android Debug Bridge

Segundo definição presente na própria documentação do Android, temos que:

> "O ADB é uma ferramenta de linha de comando versátil que permite a comunicação com uma instância de emulador ou com um dispositivo Android conectado."[2]

Neste trabalho, essa ferramenta foi utilizada para auxiliar na coleta dos arquivos de log de bateria gerados pela aplicação em cada um dos cenários de teste que foram realizados. Os comandos principais utilizados para coleta das amostras através do adb foram:

- **adb shell dumpsys batterystats --reset**: comando que reseta o arquivo de log que começa a ser coletado assim que uma instância do adb é vinculada ao dispositivo conectado.
- **adb shell dumpsys batterystats > nome_arquivo_log.txt:** comando que exporta informações do consumo de bateria do dispositivo em um arquivo de texto

Por ter a capacidade de promover uma comunicação entre o computador em que está instalado e o dispositivo, podemos gerenciar a coleta do arquivo de log através da ferramenta dumpsys, que nos provê diversas informações sobre os serviços que estão rodando dentro do dispositivo utilizado, inclusive informações de utilização de bateria, que é representado nesse caso pelo serviço batterystats.

2.5.1.2 Retrofit

Para efetuar o acesso ao *webservice* de dispositivos com sistema operacional Android, existem diversas bibliotecas que provém uma infraestrutura e classes para facilitar a comunicação via protocolo HTTP seja por meio de padrões REST ou SOAP.

2 DEVELOPERS, Android. **Android debug bridge**. . Acessado em: 06/04/2019.

Dentre todas essas bibliotecas as mais utilizadas são Volley e Retrofit[3]. Retrofit é a primeira mais usada, que por sua vez está presente em 19,84% dos aplicativos instalados em todos os dispositivos Android do mundo.

Além disso, conforme indicam os estudos (SAPUTRA, 2018) (SUFYAN 2019), uma forma de economizar bateria é distribuir o processamento para o servidor (offloading), e para isso é possível utilizar em aplicações Android as seguintes bibliotecas: Volley, OkHttp, Retrofit e HttpUrl. O estudo efetuou experimentos com todas as bibliotecas para upload e download de informações para o servidor e verificou performance de cada uma das bibliotecas atuando no 4G e no WIFI (SUFYAN, 2019). A biblioteca Retrofit foi a que apresentou a menor diferença de delay entre chamadas síncronas e assíncronas, teve 100% de sucesso na execução das chamadas tanto síncronas como assíncronas, bem como foi uma das bibliotecas que despendeu menor gasto de bateria para os dispositivos testados, junto com a biblioteca OkHttp. Outro ponto interessante do estudo é que foi constatado que testes de upload efetuados no Wifi despendem menos bateria dos que executados no 4G, bem como chamadas assíncronas também gastam menos bateria que chamadas síncronas (SUFYAN, 2019).

Portanto, constatadas essas informações dos estudos previamente citados, foi escolhida a biblioteca Retrofit para comunicação entre a aplicação Android e o webservice, efetuando sempre chamadas assíncronas para o servidor utilizando Wifi.

2.5.2 Métodos

Tendo como objetivo a análise estatística do consumo de bateria em aplicativos móveis, definimos alguns padrões e processos para a coleta das amostras necessárias para posterior comparação entre o consumo de bateria gasto nas duas técnicas de acesso ao banco de dados.

Inicialmente, foram definidos cenários de teste para coleta das amostras. Cada cenário representa uma tela dentro da aplicação móvel, sendo que cada tela contém título do cenário a ser testado - definido por padrão com numeração, por exemplo: Cenário 1, Cenário 2, etc. Descrição do cenário - objetivo do cenário, qual é a consulta a base de

3 STATS, AppBrain. **Number of android applications**. Android network libraries. . Acessado em: 06/04/2019.

dados que será feita e sua complexidade assintótica. Botão para automatizar envio de requisições via web service, sendo que o número de requisições enviadas será de cem requisições. Botão para automatizar envio das requisições via web service, sendo que o número de requisições enviadas será de quinhentas requisições. Botão para automatizar envio das requisições via web service, sendo que o número de requisições enviadas será de mil requisições. Botão para automatizar envio das requisições acessando diretamente a base de dados, sendo que o número de requisições enviadas será de cem requisições. Botão para automatizar envio das requisições acessando diretamente a base de dados, sendo que o número de requisições enviadas será de quinhentas requisições. Botão para automatizar envio das requisições acessando diretamente a base de dados, sendo que o número de requisições enviadas será de mil requisições.

Note que, para cada uma das técnicas utilizadas, são fornecidas três formas de teste, uma enviando cem requisições, outra enviando quinhentas e uma última enviando mil. Isso foi feito para efetuar testes de maior carga, podendo então utilizar esses dados de consumo de bateria como objeto de análise e comparação.

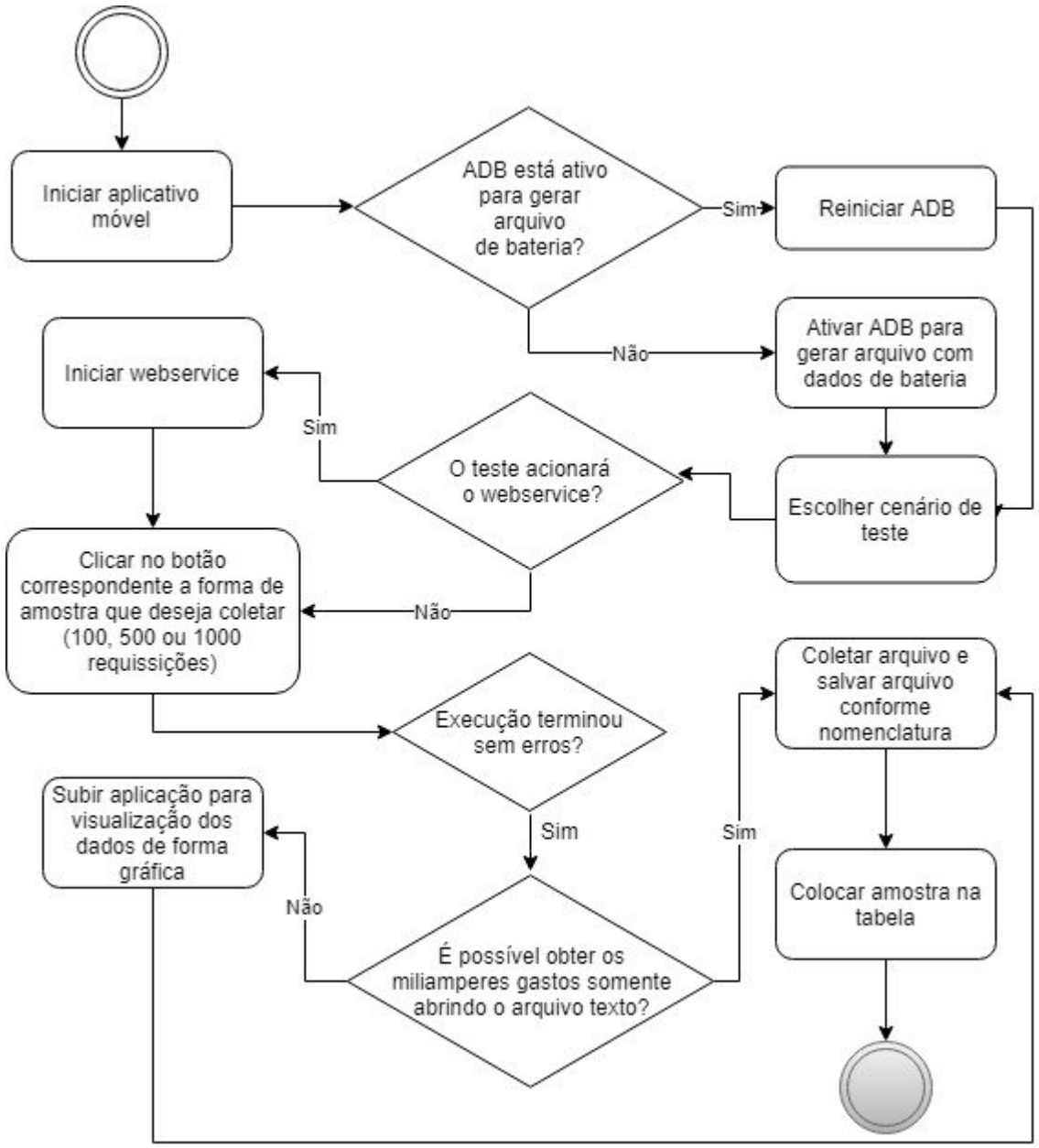
Imagem 3 - Tela aplicação móvel

Fonte: Autoria própria

A imagem 3 contém o layout da tela do aplicativo implementado, conforme foi descrito anteriormente.

O processo completo para efetuar a coleta de todas as amostras desejadas para análise estatística do consumo de bateria da aplicação móvel utilizando as duas formas de acesso a base de dados é definido na imagem abaixo:

Imagem 4 - Processo de coleta



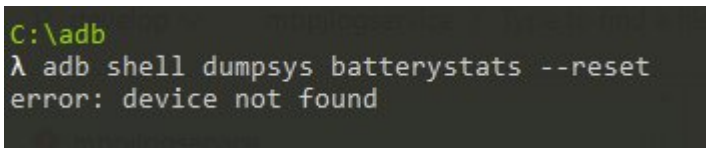
Fonte: Autoria própria

Inicialmente, de acordo com a documentação[4], precisamos limpar o arquivo que contém as informações de bateria que por padrão o ADB já coleta desde que nosso celular foi ligado pela primeira vez, assim podemos focar somente no consumo de bateria que a nossa aplicação efetuar. Para efetuar esse comando de limpeza precisamos

4 DEVELOPERS, Android. **Profile battery usage with Batterystats and Battery Historian**. . Acessado em: 06/04/2019.

conectar o dispositivo ao computador em que o ADB está instalado e executar a seguinte linha de comando:

Imagem 5 - Comando para limpar o arquivo de informações de bateria



Fonte: Autoria própria.

Além disso, precisamos desligar todas as aplicações que estejam em execução em plano de fundo, deixando somente a aplicação que utilizaremos para teste. Após esse passos preliminares, iniciamos a aplicação móvel e escolhemos o cenário de teste, após isso avaliamos se o ADB já está iniciado e, em caso positivo reiniciamos, para iniciar um novo processo de coleta de arquivo com informações do consumo de bateria para aquela execução em específico, ou em caso negativo, iniciamos pela primeira vez o ADB. Após isso, caso o teste a ser efetuado seja utilizando o *web service*, é preciso iniciar o web service, para que não tenhamos erro na execução do teste. Em seguida, clicamos no botão para escolher se queremos obter uma amostragem de teste com cem, quinhentas ou mil requisições. Quando clicamos no botão, o sistema exibe uma mensagem indicando o início das chamadas encadeadas, da mesma forma ao finalizar, outra mensagem é exibida indicando a finalização do processo de chamadas encadeadas. As imagens abaixo demonstram esses comportamentos:

Imagem 6 - Mensagem indicando início do teste

Fonte: autoria própria

Imagem 7 - Mensagem indicando fim do teste

Fonte: autoria própria.

Após o teste ser finalizado, precisamos exportar o arquivo gerado até então pelo ADB utilizando outro comando presente no ADB, conforme a imagem abaixo:

Imagem 8 - Comando de exportação

```
C:\adb
λ adb shell dumpsys batterystats > bateria_2_db_100_10.txt
```

Fonte: autoria própria

Após isso, ao abrimos o arquivo texto gerado com as informações de bateria, abaixo temos uma imagem com informações do arquivo gerado pelo ADB:

Imagem 9 - Arquivo de log de bateria (Histórico)

```
Battery History (0% used, 336 used of 512KB, 11 strings using 986):
                    0 (14) RESET:TIME: 2019-06-09-21-53-40
                    0 (1) 022 status=charging health=good plug=usb temp=272 vo
                    0 (2) 022 top=u0a241:"com.example.clsalgado.tccws"
                    0 (2) 022 job=u0a110:"com.google.android.apps.photos/com.g
                    0 (2) 022 user=0:"0"
                    0 (2) 022 userfg=0:"0"
                +21ms (2) 022 stats=0:"dump"
             +1s005ms (2) 022 stats=0:"battery-state"
             +1s019ms (2) 022 -job=u0a110:"com.google.android.apps.photos/com.
             +1s053ms (6) 022 status=discharging plug=none temp=273 volt=3830
             +1s658ms (2) 022 -top=u0a241:"com.example.clsalgado.tccws"
             +1s658ms (2) 022 +top=u0a55:"com.android.systemui"
             +2s545ms (2) 022 -top=u0a55:"com.android.systemui"
             +2s545ms (2) 022 +top=u0a57:"com.sec.android.app.launcher"
             +2s808ms (2) 022 +tmpwhitelist=u0a163:"broadcast:u0a163:Keepalive
             +2s816ms (2) 022 +job=u0a76:"com.android.chrome/org.chromium.comp
             +2s821ms (2) 022 -job=u0a76:"com.android.chrome/org.chromium.comp
             +3s897ms (2) 022 -top=u0a57:"com.sec.android.app.launcher"
             +3s897ms (2) 022 +top=u0a241:"com.example.clsalgado.tccws"
            +12s814ms (2) 022 -tmpwhitelist=u0a163:"broadcast:u0a163:Keepalive
            +17s212ms (3) 022 current=0 ap_temp=35 pst_temp=35 bat_temp=27 chg
            +29s366ms (2) 022 brightness=dark
            +30s792ms (2) 022 brightness=medium
            +31s079ms (2) 022 volt=3685
            +36s068ms (2) 022 stats=0:"battery-state"
            +36s095ms (2) 022 +job=u0a110:"com.google.android.apps.photos/com.
            +36s128ms (6) 022 status=charging plug=usb volt=3684 misc_event=0
            +36s399ms (2) 022 -top=u0a241:"com.example.clsalgado.tccws"
            +36s399ms (2) 022 +top=1000:"com.samsung.android.MtpApplication"
            +37s274ms (2) 022 -top=1000:"com.samsung.android.MtpApplication"
            +37s274ms (2) 022 +top=u0a241:"com.example.clsalgado.tccws"
            +45s606ms (2) 022 stats=0:"dump"
```

Fonte: autoria própria.

Imagem 10 - Arquivo de log de bateria (Estimativa)

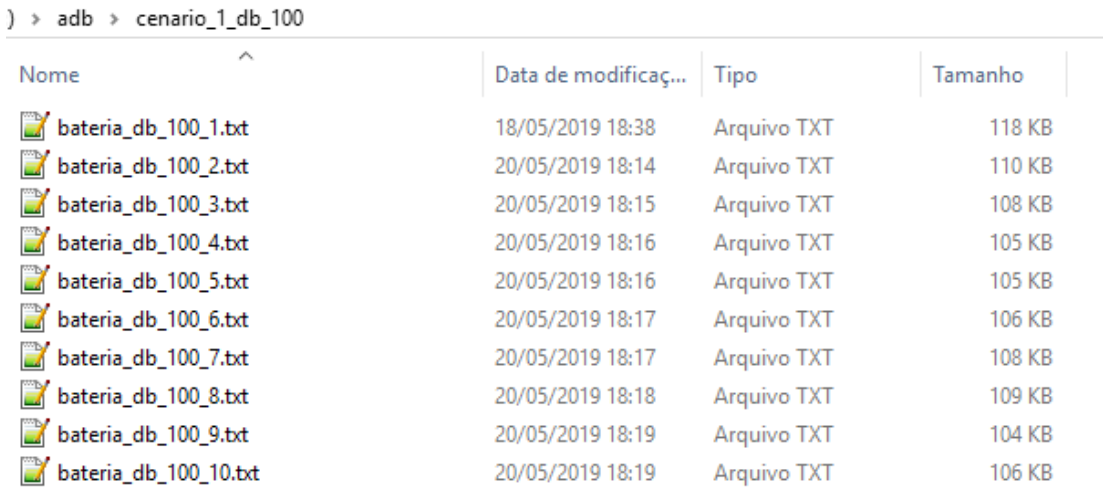
```
Estimated power use (mAh):
  Capacity: 3300, Computed drain: 4.52, actual drain: 0
  Uid u0a241: 2.25 ( cpu=0.267 wifi=1.98 ) Including smearing: 2.37 ( proportional=0.123 )
  Screen: 1.82 Excluded from smearing
  Idle: 0.189 Excluded from smearing
  Uid 1000: 0.105 ( cpu=0.0979 wifi=0.00757 ) Excluded from smearing
  Cell standby: 0.0595 ( radio=0.0595 ) Excluded from smearing
  Uid 0: 0.0434 ( cpu=0.0405 wifi=0.00284 ) Excluded from smearing
  Uid u0a55: 0.0162 ( cpu=0.0162 ) Excluded from smearing
  Wifi: 0.0111 ( cpu=0.00132 wifi=0.00974 ) Including smearing: 0.0117 ( proportional=0.000606 )
  Uid 1036: 0.00595 ( cpu=0.00595 ) Excluded from smearing
  Uid u0a163: 0.00379 ( wifi=0.00379 ) Including smearing: 0.00399 ( proportional=0.000207 )
  Uid u0a16: 0.00255 ( cpu=0.00255 ) Including smearing: 0.00269 ( proportional=0.000140 )
  Uid u0a57: 0.00215 ( cpu=0.00215 ) Including smearing: 0.00227 ( proportional=0.000118 )
  Uid 1001: 0.00177 ( cpu=0.00177 ) Excluded from smearing
  Uid 5004: 0.00132 ( cpu=0.00132 ) Excluded from smearing
  Uid u0a34: 0.00132 ( cpu=0.00132 ) Including smearing: 0.00140 ( proportional=0.0000724 )
  Uid u0a11: 0.000827 ( cpu=0.000827 ) Including smearing: 0.000872 ( proportional=0.0000453 )
  Bluetooth: 0.000661 ( cpu=0.000661 ) Including smearing: 0.000698 ( proportional=0.0000362 )
  Uid u0a120: 0.000496 ( cpu=0.000496 ) Including smearing: 0.000523 ( proportional=0.0000271 )
  Uid u0a135: 0.000496 ( cpu=0.000496 ) Including smearing: 0.000523 ( proportional=0.0000271 )
  Uid u0a110: 0.000443 ( cpu=0.000443 ) Including smearing: 0.000467 ( proportional=0.0000243 )
  Uid 5013: 0.000331 ( cpu=0.000331 ) Excluded from smearing
  Uid u0a7: 0.000331 ( cpu=0.000331 ) Including smearing: 0.000349 ( proportional=0.0000181 )
  Uid u0a28: 0.000331 ( cpu=0.000331 ) Including smearing: 0.000349 ( proportional=0.0000181 )
  Uid u0a51: 0.000331 ( cpu=0.000331 ) Including smearing: 0.000349 ( proportional=0.0000181 )
  Uid 5006: 0.000165 ( cpu=0.000165 ) Excluded from smearing
  Uid 5017: 0.000165 ( cpu=0.000165 ) Excluded from smearing
  Uid u0a42: 0.000165 ( cpu=0.000165 ) Excluded from smearing
  Uid u0a88: 0.000165 ( cpu=0.000165 ) Including smearing: 0.000174 ( proportional=0.00000905 )
  Uid u0a143: 0.000165 ( cpu=0.000165 ) Including smearing: 0.000174 ( proportional=0.00000905 )
  Uid u0a160: 0.000165 ( cpu=0.000165 ) Including smearing: 0.000174 ( proportional=0.00000905 )
  Uid u0a186: 0.000165 ( cpu=0.000165 ) Including smearing: 0.000174 ( proportional=0.00000905 )
```

Fonte: autoria própria

A informação de gasto de bateria estimado pela execução está presente no exemplo acima na segunda linha da imagem, ao lado do texto “Computed drain”. Após identificado o gasto de bateria em mAh (miliamperes), colocamos a informação na tabela de amostras e guardamos o arquivo texto seguindo o padrão de nomenclatura bateria_{cenario}_{tecnica}_{qtd_requisicoes}_{numero_amostra}.txt, sendo que {tecnica} é definido por uma das seguintes informações: db - que significa a técnica acessando diretamente a base de dados e ws - que significa a técnica acessando a base de dados através de um webservice, {cenario} identifica o número do cenário, {numero_amostra}

identifica o número da amostra e {qtd_requisicoes} identifica o número de requisições utilizado no teste, sendo que esse valor pode ser cem, quinhentos ou mil, que foram as opções que disponibilizamos para efetuar níveis diferentes de testes. Abaixo temos um exemplo de arquivos gerados para coleta de dez amostras dentro da pasta de coleta de amostras para o cenário 1:

Imagem 11 - Arquivos de log exportados



Fonte: Autoria própria

Ao fim do processo de coleta das amostras, para cada um dos cenários, preenchemos a tabela abaixo com os dados de consumo de bateria, para posteriormente ser calculado a média e mediana.

Tabela 3 - Cálculos do conjunto de amostras

Tabela de amostras de consumo de bateria coletadas para o cenário X						
	Acesso direto ao banco			Web Service		
NA	100	500	1000	100	500	1000

1	Amostra 1	Amostra 11	Amostra 21	Amostra 31	Amostra 41	Amostra 51
2	Amostra 2	Amostra 12	Amostra 22	Amostra 32	Amostra 42	Amostra 52
3	Amostra 3	Amostra 13	Amostra 23	Amostra 33	Amostra 43	Amostra 53
4	Amostra 4	Amostra 14	Amostra 24	Amostra 34	Amostra 44	Amostra 54
5	Amostra 5	Amostra 15	Amostra 25	Amostra 35	Amostra 45	A55
6	Amostra 6	Amostra 16	Amostra 26	Amostra 36	Amostra 46	Amostra 56
7	Amostra 7	Amostra 17	Amostra 27	Amostra 37	Amostra 47	Amostra 57
8	Amostra 8	Amostra 18	Amostra 28	Amostra 38	Amostra 48	Amostra 58
9	Amostra 9	Amostra 19	Amostra 29	Amostra 39	Amostra 49	Amostra 59
10	Amostra 10	Amostra 20	Amostra 30	Amostra 40	Amostra 50	Amostra 60
ME	Cálculo de média, mediana para cada um dos conjuntos de amostragem coletados.					
MA						

Fonte: Autoria própria.

Abaixo, temos uma tabela contendo a legenda das siglas utilizadas na tabela acima:

Tabela 4 - Siglas usadas no levantamento das amostras

NA	Número da amostra
ME	Média das amostras
MA	Mediana das amostras

Fonte: Autoria própria.

3 TRABALHOS RELACIONADOS

Sobre consumo de energia em dispositivos móveis existem pesquisas que tratam do mesmo tema nos aspectos mais variados da arquitetura e organização de um dispositivo móvel, tais como, *hardware*, sistema operacional, *firmware* e camada de aplicação ou *software*. A seguir será apresentado trabalhos relacionados à área, em suas várias categorias, para colocar o presente trabalho no contexto e tentar definir o estado da arte para a mesma.

No nível de hardware foi proposto trabalhos como LongRun (FLEISCHMANN, 2001), que mede a utilização do processador ao nível do *hardware* e altera a velocidade da CPU com base em sua medição. LongRun é um sistema sem memória e puramente reativo, que mede a utilização da CPU para um intervalo e se há tempo ocioso ele reduz a velocidade do CPU e voltagem para levar a utilização de volta a cem por cento. Abordagens de *hardware* não tem conhecimento de qualquer demanda futura, não conseguem interpretar os padrões de demandas de energia e não têm conhecimento das necessidades de aplicação.

Em nível de sistema operacional e aplicação os esforços de pesquisa têm se concentrado em técnicas como a adaptação de energia por parte das aplicações (FLINN, 1999) que demonstram como os aplicativos podem modificar dinamicamente seu comportamento para economizar energia e como o sistema operacional pode orientar essa adaptação para alcançar duração de bateria desejada. Segundo suas avaliações, estendem a vida útil da bateria em até trinta por cento. Outra abordagem é o eLens (HAO, et al. 2013), que propõe a modelagem de energia por instrução executada do aplicativo no dispositivo. Esta abordagem não exige que o desenvolvedor possua hardware especializado, não afeta a usabilidade dos aplicativos e pode medir o uso de energia por método. Outras pesquisas são voltadas ao gerenciamento de energia, que tratam de classificar e avaliar o consumo dos aplicativos de smartphone e/ou gerar perfis de consumo (MARCU, 2007) que monitoram parâmetros de status da bateria ao executar determinada carga de trabalho e apresentam benchmarks e uma análise a respeito de como aplicações de software têm efeito sobre o consumo total de energia de um dispositivo móvel. Há outros que fazem uso de técnicas de *machine learning* como Mehrotra (et al. 2018) onde o experimento foi conduzido em algumas aplicações móveis

populares sob diferentes ambientes. A energia usada pelos aplicativos foi registrada e usada como um conjunto de treinamento para desenvolver um modelo de aprendizado de máquina que classifica essas aplicações móveis em três categorias: baixa, média e alta dependendo do uso de energia, usando a abordagem de classificação. Vários algoritmos de classificação são explorados para encontrar o modelo mais preciso que pode ser usado para classificar os aplicativos móveis.

Há trabalhos, assim como o nosso, focados em estimar e analisar a complexidade energética através da comunicação entre sistemas distribuídos (CHAMAS, 2018). Neste trabalho, é comparado protocolos como REST, SOAP, Socket e RPC na execução de algoritmos de ordenação de diferentes complexidades aplicados sobre vetores de diversos tamanhos e tipos de dados. Os resultados mostram que a execução local é mais econômica com algoritmos menos complexos, pequeno tamanho de entrada e tipo de dados menos complexos. Há abordagens que comparam tecnologias e *frameworks mobile* utilizando o desenvolvimento de aplicações móveis de forma nativa versus forma híbrida (Kurtsz, Noguez et al). Concluem que quanto maior a carga de processamento aplicada para execução do algoritmo que trata de uma ordenação de vetor, mais econômico é a aplicação híbrida, por executar o processamento no servidor. O processamento do algoritmo relatado é feito no servidor versus internamente dentro da aplicação móvel, o que no nosso caso é distinto pois não estamos efetuando um processamento da informação coletada na base de dados, mas sim uma comparação de duas formas distintas de acessar a base de dados.

Também existem pesquisas de benchmarks e análise de desempenho voltadas à sistemas ou bibliotecas. Como é o caso do trabalho de Saputra (2018) onde criaram uma API para observar o desempenho de velocidade de recuperação de dados usando as bibliotecas Retrofit e Volley, sendo que utilizamos neste projeto a Retrofit. O teste de desempenho de velocidade Retrofit vs Volley foi implementado em um aplicativo simples baseado em Android e a aplicação analisou os dados no formato JSON. Cada uma das bibliotecas solicitou e analisou os dados por cinco vezes para cada volume de dados escolhidos variando de 10 a 100.000. O resultado do experimento mostra que o Retrofit funciona mais rápido que o Volley na análise de dados.

Apesar de encontrar abordagens semelhantes, não foi encontrado nenhuma literatura que discutiu o custo energético de executar as tarefas localmente versus executá-las remotamente junto ao estilo arquitetônico escolhido.

4 RESULTADOS E DISCUSSÕES

4.1 PRIMEIRO CENÁRIO

O primeiro cenário abordado efetua o teste de acesso a uma base de dados simulando um cenário comum na maioria das aplicações móveis - um cenário de login. Além de ser um cenário comum em qualquer aplicação móvel, esse cenário é importante por ter baixa complexidade, já que é uma simples consulta com condição e não exige operações de união e interseção entre tabelas. Além disso, a complexidade é ainda menor pelo fato de existir somente um usuário inserido no banco de dados local, o que não exige otimização de índices e grandes buscas na base. O intuito com esse cenário foi definir um cenário base, de baixa complexidade para entender como as duas abordagens se comportam em um cenário mais simplista.

A tabela abaixo contêm todas as amostras coletadas para análise do primeiro cenário, com todos os dados coletados em miliamperes:

Tabela 5 - Amostras análise primeiro cenário.

Tabela de amostras de consumo de bateria coletadas para o primeiro cenário						
	Acesso direto ao banco			Web Service		
NA	100	500	1000	100	500	1000
1	4,27	11,10	21,00	3,11	3,70	5,21
2	3,24	12,40	23,00	1,55	3,33	5,31
3	3,07	12,30	22,80	1,60	3,21	5,07
4	3,12	12,20	23,10	1,70	3,43	5,09
5	2,78	12,40	22,80	1,60	3,42	5,33
6	2,90	12,00	23,30	1,82	3,29	5,42
7	3,03	12,20	22,70	1,48	3,52	5,29
8	2,81	12,20	23,00	1,74	3,46	5,53

9	2,78	12,70	23,30	1,68	3,28	5,39
10	3,08	12,20	23,30	1,69	3,36	5,30
ME	**3,108**	**12,17**	**22,83**	**1,797**	**3,36**	**5,30**
MA	**3,05**	**12,2**	**23**	**1,685**	**3,39**	**5,305**

Fonte: Autoria própria.

Nota-se na tabela acima que para o primeiro cenário, quando o teste foi de cem chamadas encadeadas para cada uma das abordagens, tivemos 3,108 mAh gastos na primeira técnica e 1,797 mAh na segunda técnica, resultando assim em uma diferença de 1,311 mAh gastos a menos quando utilizada a abordagem com acionamento do webservice. Em seguida, a medida que aumentaram o volume de chamadas encadeadas, ou seja, aumentando o nível de exigência de processamento da aplicação, notamos que o acréscimo no consumo de bateria também vai crescendo naturalmente. Quando analisada a coluna que contêm as amostras para quinhentas chamadas consecutivas para ambas as abordagens, percebe-se uma diferença ainda maior na média de consumo de bateria novamente na utilização da abordagem com webservice - 8,81 mAh. Por fim, quando aumentamos a ordem de grandeza das chamadas encadeadas - mil chamadas, percebemos que a diferença novamente aumenta entre as duas abordagens, 17,53 mAh, sempre prevalecendo a economia de energia na utilização da abordagem com pré processamento no webservice. Importante notar que não houve aumento da complexidade ciclomática das aplicações nas duas abordagens, o que poderia trazer impacto devido a menor capacidade de processamento de uma aplicação móvel se comparada a um notebook.

Outro ponto importante que identificou-se com o auxílio das amostras é que a medida que aumentaram a quantidade de chamadas encadeadas, menor é o impacto no consumo de bateria na abordagem que utiliza o webservice, pois sua variação nas três diferentes formas de amostragem foi pequena, totalizando 1,563 mAh do cenário com cem requisições para o cenário com quinhentas requisições, e de 1,94 mAh do cenário com quinhentas requisições encadeadas para o cenário com mil requisições, enquanto que no caso da abordagem acessando diretamente o servidor de banco de dados, temos diferenças de 9,062 mAh e 10,66 mAh respectivamente.

4.2 SEGUNDO CENÁRIO

O segundo cenário também é considerado um cenário base e simples, porém não efetuando mais uma operação de consulta, e sim de inserção na base de dados. O cenário simula um cadastro, outra operação comum em qualquer aplicação móvel presente no mercado.

A tabela abaixo contêm todas as amostras coletadas para análise do segundo cenário, com todos os dados coletados em miliamperes:

Tabela 6 - Amostras análise segundo cenário.

Tabela de amostras de consumo de bateria coletadas para o segundo cenário						
	Acesso direto ao banco			Web Service		
NA	100	500	1000	100	500	1000
1	4,94	13,1	24,2	1,3	2,86	4,52
2	3,63	12,8	22	1,25	2,74	4,5
3	3,77	13	22,1	1,3	2,86	4,59
4	4,3	12,5	22,4	1,37	2,9	4,76
5	3,48	12,4	21,9	1,53	2,79	4,5
6	3,64	12,7	22	1,33	2,91	4,45
7	3,66	12,4	22,1	1,35	2,83	4,91
8	3,93	12,5	22,5	1,34	2,68	4,48
9	3,63	12,4	22,3	1,3	2,74	4,45
10	3,76	11,6	22,1	1,42	2,73	4,52
ME	**3,874**	**12,54**	**22,36**	**1,349**	**2,804**	**4,568**
MA	**3,71**	**12,5**	**22,1**	**1,335**	**2,81**	**4,51**

Fonte: Autoria própria.

Se comparado a tabela presente no cenário 1 - caso de teste de select em tabela com um único registro, com a tabela presente no atual cenário, podemos ver uma diminuição na média do consumo de bateria para todos as formas de teste (100, 500 e

1000 requisições) para ambas as técnicas - *webservice* e acesso direto ao banco de dados.

Da mesma forma como no cenário 1, a abordagem que utilizou o webservice como intermédio para comunicação com o banco de dados se saiu mais econômica no que diz respeito a consumo de bateria, e também mostrou um aumento menor de consumo quando utilizados os testes com maior carga de chamadas encadeadas (500 e 1000 chamadas).

4.3 TERCEIRO CENÁRIO

O terceiro cenário aborda mais um dos cenários base de operações que efetuam somente alteração em uma única linha da tabela, porém diferentemente de uma inserção na base, efetuou-se uma atualização, utilizando o método UPDATE da linguagem SQL. Importante notar que como esse teste foi feito após as diversas coleta de amostras do cenário 2, temos um update feito em uma linha dentre mais de 33000 registros dentro da base de dados.

A tabela abaixo contêm todas as amostras coletadas para análise do terceiro cenário, com todos os dados coletados em miliamperes:

Tabela 7 - Amostras análise terceiro cenário.

Tabela de amostras de consumo de bateria coletadas para o segundo cenário						
	Acesso direto ao banco			Web Service		
NA	100	500	1000	100	500	1000
1	3,36	12,5	23,2	1,59	3,36	5,19
2	3,27	12,2	23,3	1,68	3,29	5,05
3	3,34	12,8	25,1	1,53	3,32	4,94

4	3,38	12,3	23	1,55	3,17	4,96
5	3,19	12,3	23,8	1,63	3,26	4,97
6	3,2	12,1	23,7	1,46	3,16	5,11
7	3,32	12,3	23,3	1,62	3,25	5,14
8	3,37	12,6	22,7	1,55	3,28	5,16
9	3,31	12,5	24	1,56	3,17	4,96
10	3,68	12,2	23,4	1,66	3,31	4,98
ME	**3,342**	**12,38**	**23,55**	**1,583**	**3,257**	**5,046**
MA	**3,33**	**12,3**	**23,35**	**1,575**	**3,27**	**5,015**

Fonte: Autoria própria.

No caso do cenário em que tratou-se de efetuar a operação *UPDATE* na tabela, podemos identificar um aumento no consumo de bateria quando utilizamos a abordagem com webservices se compararmos com o cenário de inserção na base (cenário 2), porém identificou-se também uma leve diminuição no consumo se comparado com o cenário de consulta na base de dados (cenário 1). A abordagem utilizando acesso direto a base de dados novamente se mostrou mais custosa que a abordagem utilizando webservices, porém se mostrou mais econômica com um número menor de chamadas encadeadas (100 e 500) com relação ao cenário 2 (*INSERT*), porém maior em consumo se comparada com o cenário 1 (*SELECT*), porém, quando observou-se a coluna com mil chamadas encadeadas, com relação a técnica de acesso direto, foi a que mais consome bateria do dispositivo móvel.

4.5 CONSIDERAÇÕES FINAIS

Com o presente experimento conseguimos metrificar e coletar amostragem para comparar duas técnicas diferentes que tem como objetivo acessar a mesma informação final. A coleta dos cenários base serve de início para uma pesquisa que pode se aprofundar ainda mais, explorando-se mais cenários e aprofundando as análises a nível de detalhes, para descobrir os motivos e razões que levam a essa disparidade. Nosso trabalho mostrou que a abordagem que utiliza como intermediário para comunicação entre a aplicação móvel e o servidor de banco de dados um webservice, provê uma maior

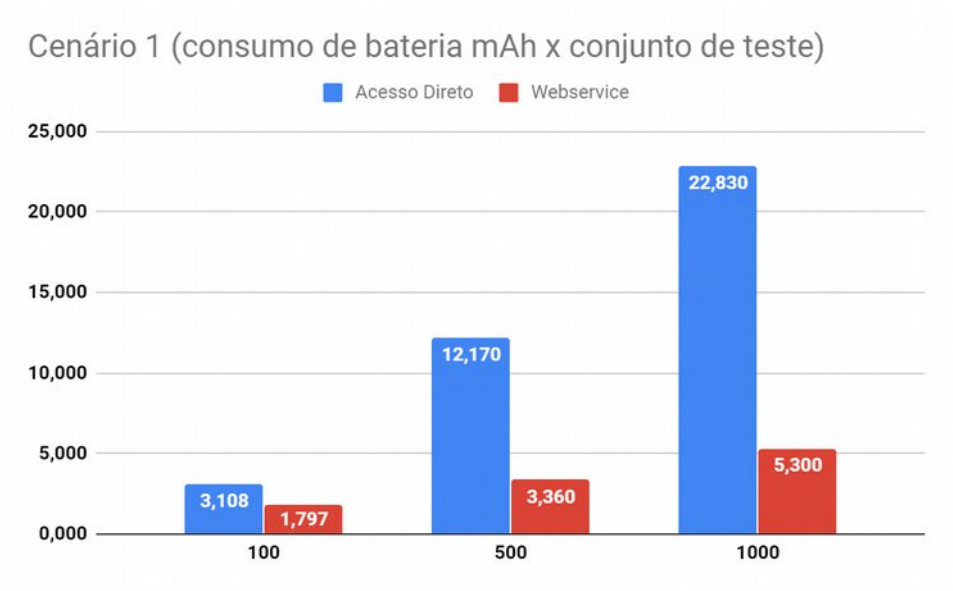
economia de bateria para o dispositivo móvel, pois em todos os cenários analisados e dentre todos os casos de teste com chamadas encadeadas (dos mais leves aos mais altos), a abordagem se mostrou mais econômica.

Assim, podemos concluir que por mais que a comunicação seja menos direta, por envolver um intermediário, o consumo de bateria é otimizado utilizando web services, além de prover diversas outras vantagens, conforme apresentado neste trabalho.

5 CONCLUSÃO

Com esta pesquisa conseguimos evidenciar que entre as duas formas de se implementar uma aplicação móvel, a que faz acesso ao banco de dados diretamente de uma aplicação cliente consumirá muito mais energia do dispositivo, fazendo com que a bateria diminua drasticamente. Por outro lado a técnica que se comunica com o *webservice* chega a ser 78.57% mais econômica. Isto, considerando o ambiente controlado que propusemos neste trabalho e sem considerar o *overhead* de outras aplicações.

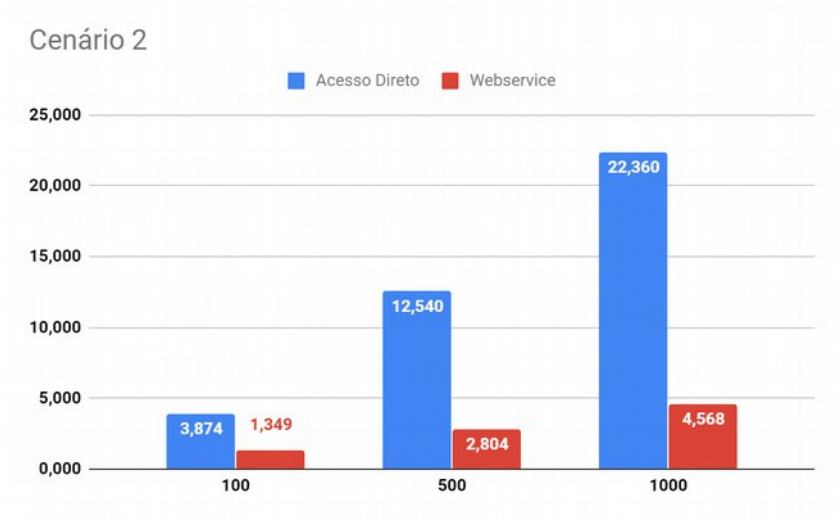
Imagem 12 - Resultado do cenário 1



Fonte: Autoria própria

A imagem 12 contém a quantidade gasta em mAh em cada conjunto de teste para o primeiro cenário que foi feito consulta no banco de dados. Se percebe que a medida que foi aumentada a quantidade de consultas na base, para ambas as técnicas foi-se aumentando o consumo. Contudo a técnica utilizando o webservice foi 42.18% mais econômica no primeiro conjunto. No segundo conjunto foi 72.39% mais econômica e no terceiro foi 76.78%.

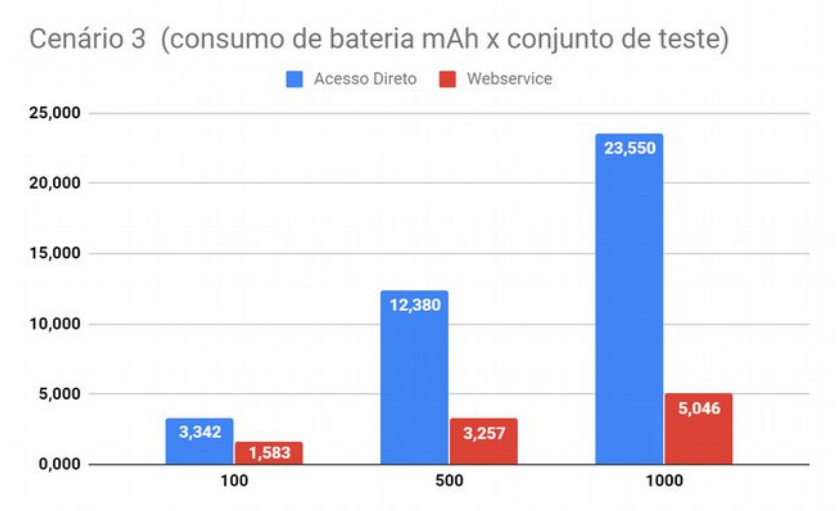
Imagem 13 - Resultado do cenário 2



Fonte: Autoria própria

No terceiro cenário, contido na imagem 13, foram realizados *inserts* encadeados na base de dados e analisado a quantidade gasta em mAh em cada conjunto de teste. Novamente a técnica de acesso direto consumiu bem mais energia do dispositivo móvel. Para o primeiro conjunto a técnica de acesso direto foi 187.17% mais custosa, para o segundo conjunto foi de 347.21% e no último conjunto foi 389.49% mais custosa.

Imagem 14 - Resultado do cenário 3



Fonte: Autoria própria

No último cenário, representado na imagem 14, foram realizados operações de *updates* em registros no banco de dados. E outra vez, utilizando a técnica que possui o web service se obteve mais economia. Sendo que no primeiro conjunto a técnica de acesso direto consumiu 111.11% mais energia que na técnica contendo *webservice*. 280.10% consumidos a mais na técnica de acesso direto para o segundo conjunto e no último conjunto foi consumido 366.70% a mais pela mesma técnica.

A seguir serão apresentadas perguntas que foram formuladas a fim de traçar o objetivo desta presente pesquisa e as respostas que obtivemos com o decorrer do experimento.

Pergunta 1: O que traz mais economia energética no contexto móvel, executar tarefas remotamente via web service ou executá-las diretamente da aplicação?

Ficou evidente que executar via web é a abordagem mais econômica e em algum dos casos de testes chegando a economizar 400% da bateria do dispositivo.

Pergunta 2: Qual é a diferença energética entre as duas abordagens quando se efetuam diferentes conjuntos de teste?

Cenário 1 - consulta: até 17,53 mAh gastos a mais pela técnica de acesso direto.

Cenário 2 - inserção: até 17,792 mAh gastos a mais pela técnica de acesso direto.

Cenário 3 - atualização: até 18,504 mAh gastos a mais pela técnica de acesso direto.

Pergunta 3: Como o tempo de resposta se comporta em cada uma dessas técnicas?

Empiricamente foi identificado que quando efetuados testes de maior carga, o tempo de resposta da técnica utilizando acesso direto a base de dados foi superior em mais de 1 minuto com relação a técnica de acesso via webservice.

5.1 CONTRIBUIÇÕES

Da mesma forma que fizemos uso de outras pesquisas e trabalhos para a construção deste, essa pesquisa também tem o intuito de contribuir com a comunidade científica. Pois o tema é recorrente, porém o objeto de estudo ainda não tinha sido tratado e não foi encontrado em repositórios de pesquisa como IEEE e outros semelhantes.

Contribui-se também com a comunidade de desenvolvedores de software móveis e outros interessados nessa temática, pois essa pesquisa quantitativa, qualitativa e objetiva auxiliará no momento de escolha de uma abordagem mais eficientes em termos energéticos, dentre as citadas aqui.

5.2 TRABALHOS FUTUROS

Como possibilidade de trabalhos futuros temos o estudo aplicado a outros sistemas operacionais de dispositivos móveis, como por exemplo IOS ou Windows phone. Além disso, nosso trabalho foi focado em um número pequeno de cenários, porém outros cenários a nível de complexidade podem ser estudados e analisados. Outra possibilidade é o estudo em outras versões de sistema operacional, ou até mesmo em outras famílias de celulares. Outra possibilidade é efetuar essa análise comparativa de consumo de bateria entre funções distintas dentro de uma aplicação móvel, como por exemplo carregamento de um mapa x upload de imagem, consumo de wireless x consumo de 4G, diversas análises podem ser efetuadas quando analisado detalhadamente o arquivo de log de bateria gerado pelo ADB.

Por fim, temos que lembrar que o procedimento de teste foi efetuado dentro de uma rede local, portanto poderíamos avançar mais no sentido de ampliar a latência e a distância física entre os servidores web e de banco de dados com o dispositivo móvel, verificando se esse aumento de latência possa impactar no consumo de bateria de maneira positiva ou negativa.

6 REFERÊNCIAS BIBLIOGRÁFICAS

[1] Liu, Xiaotao, Prashant Shenoy, and Mark D. Corner. "**Chameleon: Application-level power management."** IEEE Transactions on Mobile Computing 7.8 (2008): 995-1010.

[2] NAGATA, Kyosuke; YAMAGUCHI, Saneyasu; OGAWA, Hisato. **“A power saving method with consideration of performance in android terminals”**. In: 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing. IEEE, 2012. p. 578-585.

[3] COULOURIS, George, et al. “**Sistemas Distribuídos-: Conceitos e Projeto”. Bookman Editora**, 2013. Capítulo 19.

[4] DISCHLER, Jerry. **“Building for the next moment. Inside Ad-Words”**, 2015. Disponível em: <https://adwords.googleblog.com/2015/05/building-for-next-moment.html>. Acessado em: 06/04/2019.

[5] STATISTA. **“Number of mobile phone users worldwide from 2013 to 2019”** (in billions). 2015. Disponível em: <https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide>. Acessado em: 05/04/2019.

[6] Silva, Edna Lúcia da, and Estera Muszkat Menezes. **“Metodologia da pesquisa e elaboração de dissertação”**. (2001).

[7] Gil, Antonio Carlos. **“Métodos e técnicas de pesquisa social”**. 6. ed. Ediitora Atlas SA, 2008.

[8] HAO, Shuai, et al. **Estimating mobile application energy consumption using program analysis.** In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013. p. 92-101.

[9] Kurihara, S., Fukuda, S., Yamaguchi, S., & Oguchi, M. **Estimation of power consumption of each application based on software dependency in android**. In 2017 IEEE 6th Global Conference on Consumer Electronics (GCCE) (pp. 1-2). IEEE.

[10] CHAMAS, Carolina Luiza. **Consumo de energia em dispositivos móveis Android: análise das estratégias de comunicação utilizadas em Computation Offloading**. PhD Thesis. Universidade de São Paulo. 2018

[11] MEHROTRA, Deepti, et al. **Multiclass classification of mobile applications as per energy consumption**. Journal of King Saud University-Computer and Information Sciences, 2018.

[12] Marcu, M., Tudor, D., Moldovan, H., & Micea, M. (2007, December). **Power profile evaluation of battery-powered mobile applications.** In 2007 14th IEEE International Conference on Electronics, Circuits and Systems (pp. 1015-1018). IEEE.

[13] LIU, Xiaotao; SHENOY, Prashant; CORNER, Mark D. Chameleon: **Application-level power management**. IEEE Transactions on Mobile Computing, 2008, 7.8: 995-1010.

[14] FLINN, Jason; SATYANARAYANAN, Mahadev. **Energy-aware adaptation for mobile applications.** ACM, 1999.

[15] LUFEI, Hanping; SHI, Weisong. **Energy-aware QoS for application sessions across multiple protocol domains in mobile computing.** Computer Networks, 2007, 51.11: 3125-3141.

[16] FLEISCHMANN, Marc. **LongRun power management-Dynamic power management for Crusue processors**. Disponível em: <http://www. transmeta. com/corporate/pressroom/-whitepapers.html>, 2001.

[17] DEVELOPERS, Android. **Profile battery usage with Batterystats and Battery Historian**. Disponível em: <https://developer.android.com/studio/profile/battery-historian>. Acessado em: 06/04/2019.

[18] Kurtz, Kellerson, Noguez, Marcelo et al. **Comparing Performance and Energy Consumption of Android Applications: Native Versus Web Approaches**. VII Brazilian Symposium on Computing Systems Engineering (2017).

[19] PAPAZOGLOU, Michael P.; GEORGAKOPOULOS, Dimitrios. **Service-oriented computing**. Communications of the ACM, 2003, 46.10: 25-28.

[20] ANDERSON, Charles. **Docker [software engineering]**. IEEE Software, 2015, 32.3: 102-c3.

[21] DEVELOPERS, Android. **D**. Disponível em: <https://developer.android.com/studio/command-line/dumpsys>. Acessado em: 06/04/2019.

[22] DEVELOPERS, Android. **Android debug bridge**. Disponível em: <http://developer. android. com/tools/help/adb.html>. Acessado em: 06/04/2019.

[23] STATS, AppBrain. **Number of android applications**. Android network libraries. Disponível em: <https://www.appbrain.com/stats/libraries/tag/network/android-network-libraries>. Acessado em: 06/04/2019.

[24] SAPUTRA, Kurnia; FARHAN, Khalid; IRVANIZAM, Irvanizam. **Analysis on the Comparison of Retrofit and Volley Libraries on Android-Based Mosque Application.** In: 2018 International Conference on Electrical Engineering and Informatics (ICELTICs) (44501). IEEE, 2018. p. 117-121.

[25] SUFYAN, Farhan; BANERJEE, Amit; **Comparative Analysis of Network Libraries for Offloading Efficiency in Mobile Cloud Environment.** In: (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 10, No. 2, 2019

APÊNDICE A - Código fonte do dispositivo móvel

Repositório no github:

Arquivo XML que representa a tela utilizada para testes dos cenários:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/titulo_cenario"
        android:layout_width="wrap_content"
        android:layout_height="22dp"
        android:layout_marginBottom="8dp"
        android:text="@string/titulo_cenario"
        android:textColor="@color/colorAccent"
        app:layout_constraintBottom_toTopOf="@+id/botao_1000"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/titulo_webservice"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        android:text="@string/titulo_webservice"
        android:textColor="@color/colorAccent"
        app:layout_constraintBottom_toTopOf="@+id/botao_1000"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/titulo_cenario" />


    <Button
        android:id="@+id/botao_100"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="40dp"
        android:layout_marginRight="20dp"
        android:background="@color/colorPrimaryDark"
        android:text="@string/botao_100"
```

```
    android:textColor="@color/colorAccent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintStart_toEndOf="@+id/botao_500"
    app:layout_constraintTop_toTopOf="@id/botao_500"
    app:layout_constraintVertical_bias="0.0" />

<Button
    android:id="@+id/botao_500"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="32dp"
    android:background="@color/colorPrimaryDark"
    android:text="@string/botao_500"
    android:textColor="@color/colorAccent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintStart_toEndOf="@+id/botao_1000"
    app:layout_constraintTop_toTopOf="@id/botao_1000"
    app:layout_constraintVertical_bias="0.0" />

<Button
    android:id="@+id/botao_1000"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:background="@color/colorPrimaryDark"
    android:text="@string/botao_1000"
    android:textColor="@color/colorAccent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.41" />

<TextView
    android:id="@+id/titulo_acessodireto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:layout_marginBottom="8dp"
    android:text="@string/titulo_acessodireto"
    android:textColor="@color/colorAccent"
    app:layout_constraintBottom_toTopOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/botao_1000"
```

```
        app:layout_constraintVertical_bias="0"/>

    <Button
        android:id="@+id/botao_100_db"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="40dp"
        android:layout_marginRight="20dp"
        android:background="@color/colorPrimaryDark"
        android:text="@string/botao_100"
        android:textColor="@color/colorAccent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintStart_toEndOf="@+id/botao_500_db"
        app:layout_constraintTop_toTopOf="@id/botao_500_db"
        app:layout_constraintVertical_bias="0.0" />

    <Button
        android:id="@+id/botao_500_db"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:background="@color/colorPrimaryDark"
        android:text="@string/botao_500"
        android:textColor="@color/colorAccent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintStart_toEndOf="@+id/botao_1000_db"
        app:layout_constraintTop_toTopOf="@id/botao_1000_db"
        app:layout_constraintVertical_bias="0.0" />

    <Button
        android:id="@+id/botao_1000_db"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:background="@color/colorPrimaryDark"
        android:text="@string/botao_1000"
        android:textColor="@color/colorAccent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="@id/titulo_acessodireto"
        app:layout_constraintVertical_bias="0.41" />

</android.support.constraint.ConstraintLayout>
```

Classe que mapeia os objetos em tela representados no xml acima:

```
package com.example.clsalgado.tccws;


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.Button;
import android.widget.Toast;

import com.example.clsalgado.tccws.app.TccApplication;
import com.example.clsalgado.tccws.callback.CallbackLogin;
import com.example.clsalgado.tccws.modelo.Usuario;
import com.example.clsalgado.tccws.service.LoginService;
import com.example.clsalgado.tccws.tarefa.LoginTask;

import javax.inject.Inject;

import butterknife.BindView;
import butterknife.ButterKnife;
import butterknife.OnClick;
import okhttp3.ResponseBody;
import retrofit2.Call;

public class MainActivity extends AppCompatActivity {

    @Inject
    LoginService loginService;

    @BindView(R.id.botao_100)
    Button btn100Ws;

    @BindView(R.id.botao_500)
    Button btn500Ws;

    @BindView(R.id.botao_1000)
    Button btn1000Ws;

    @BindView(R.id.botao_100_db)
    Button btn100Db;

    @BindView(R.id.botao_500_db)
    Button btn500Db;

    @BindView(R.id.botao_1000_db)
    Button btn1000Db;

    private int contadorIteracoes = 1;
```

```
private int maximoChamadas = 100;

private Usuario usuario;

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

    // responsável por injeter as dependências com Dagger
    ((TccApplication) getApplication()).getComponent().inject(this);

    ButterKnife.bind(this);

    // configurando usuário fixo
    usuario = new Usuario(1505602, "789456123");

}

@OnClick(R.id.botao_100)
public void efetuarChamadaWs100Requisicoes() {
    contadorIteracoes = 1;
    maximoChamadas = 100;
    executaChamadaWebService(usuario);
}

@OnClick(R.id.botao_500)
public void efetuaChamadaWs500Requisicoes() {
    contadorIteracoes = 1;
    maximoChamadas = 500;
    executaChamadaWebService(usuario);
}

@OnClick(R.id.botao_1000)
public void efetuaChamadaWs1000Requisicoes(){
    contadorIteracoes = 1;
    maximoChamadas = 1000;
    executaChamadaWebService(usuario);
}

@OnClick(R.id.botao_100_db)
public void efetuarChamadaDb100Requisicoes() {
    contadorIteracoes = 1;
    maximoChamadas = 100;
    executaChamadaServidorBancoDeDados(usuario);
}

@OnClick(R.id.botao_500_db)
public void efetuaChamadaDb500Requisicoes() {
    contadorIteracoes = 1;
```

```
        maximoChamadas = 500;
        executaChamadaServidorBancoDeDados(usuario);
    }

    @OnClick(R.id.botao_1000_db)
    public void efetuaChamadaDb1000Requisicoes(){
        contadorIteracoes = 1;
        maximoChamadas = 1000;
        executaChamadaServidorBancoDeDados(usuario);
    }

    /**
     * Método que faz a chamada ao web service passando um usuário no formato JSON como parâmatro
     * @param usuario objeto que contêm matrícula e senha
     */
    private void executaChamadaWebService(Usuario usuario) {
                                Toast.makeText(this, "Iniciando chamadas webservice", Toast.LENGTH_SHORT).show();
        String tagLog = "WEBSERVICE" + maximoChamadas;
            Log.i(tagLog, "Iniciando chamadas webservice " + contadorIteracoes + " DE " + maximoChamadas);
        Call<ResponseBody> call = loginService.login(usuario);
        call.enqueue(new CallbackLogin(this));
    }

    public void trataRetornoChamadaWebService() {
        if(contadorIteracoes <= maximoChamadas) {
            contadorIteracoes++;
            String tagLog = "WEBSERVICE" + maximoChamadas;
            Log.i(tagLog, "Continuação chamadas webservice " + contadorIteracoes + " DE " + maximoChamadas);
            Call<ResponseBody> call = loginService.login(usuario);
            call.enqueue(new CallbackLogin(this));
        }else {
                                    Toast.makeText(this, "Finalizando chamadas webservice", Toast.LENGTH_SHORT).show();
        }
    }

    /**
     * Método que aciona tarefa assíncrona para executar chamada ao banco de dados
     * @param usuario objeto usuário
     */
    private void executaChamadaServidorBancoDeDados(Usuario usuario) {
                    Toast.makeText(this, "Iniciando chamadas direta a base de dados", Toast.LENGTH_SHORT).show();
        String tagLog = "DIRETODATABASE" + maximoChamadas;
        Log.i(tagLog, "Iniciando chamadas direta ao banco de dados " + contadorIteracoes + " DE " + maximoChamadas);
```

```
        new LoginTask(this).execute(usuario);
    }

    /**
     * Método que trata retorno da chamada ao servidor de banco de dados
     * @param usuario objeto usuário
     */
    public void trataRetornoChamadaServidorBancoDeDados(Usuario usuario) {
        if(contadorIteracoes <= maximoChamadas) {
            contadorIteracoes++;
            String tagLog = "DIRETODATABASE" + maximoChamadas;
                    Log.i(tagLog, "Continuação chamadas direta ao banco de dados " +
contadorIteracoes + " DE " + maximoChamadas);
            new LoginTask(this).execute(usuario);
        }else {
                    Toast.makeText(this, "Finalizando chamadas direta a base de dados",
Toast.LENGTH_SHORT).show();
        }
    }
}
```

Classe que configura biblioteca Retrofit para chamadas via webservice:

```
package com.example.clsalgado.tccws.module;

import com.example.clsalgado.tccws.service.LoginService;
import com.example.clsalgado.tccws.service.UsuarioService;

import dagger.Module;
import dagger.Provides;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

@Module
public class TccModule {

    @Provides
    public LoginService getLoginService() {
        Retrofit retrofit = configuracaoRetrofit();
        return retrofit.create(LoginService.class);
    }

    @Provides
    public UsuarioService getUsuarioService() {
        Retrofit retrofit = configuracaoRetrofit();
        return retrofit.create(UsuarioService.class);
    }

    private Retrofit configuracaoRetrofit() {
        return new Retrofit.Builder()
```

```
            .baseUrl("http://192.168.0.12:8080/wsjpa/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();
    }
}
```

Classe utilizada para execução de chamada direta ao banco de dados para o cenário 1 (*SELECT*):

```
package com.example.clsalgado.tccws.tarefa;

import android.os.AsyncTask;
import android.util.Log;

import com.example.clsalgado.tccws.MainActivity;
import com.example.clsalgado.tccws.modelo.Usuario;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * Classe criada para executar chamada ao servidor de banco de dados
 * Necessário criação de uma AsyncTask pois não é permitido fazer esse tipo de acesso na thread atual da Activity
 */
public class LoginTask extends AsyncTask<Usuario, Integer, Usuario> {

    private MainActivity mainActivity;

    public LoginTask(MainActivity mainActivity) {
        this.mainActivity = mainActivity;
    }

    @Override
    protected Usuario doInBackground(Usuario... usuarios) {
        Usuario retorno = new Usuario();
        try {
            Connection connection =
                    DriverManager.getConnection("jdbc:mariadb://192.168.0.12:3306/utfpr_tcc_wsjpa", "root", "root");

            PreparedStatement preparedStatement =
                    connection.prepareStatement("select * from usuario where matricula = ? and senha = ?");
            preparedStatement.setInt(1, usuarios[0].getMatricula());
            preparedStatement.setString(2, usuarios[0].getSenha());
```

```
            boolean sucesso = preparedStatement.execute();

            if(!sucesso) {
                Log.d("DIRETODATABASE", "Não tem resultado para essa consulta!");
            }
            preparedStatement.close();
            connection.close();
        } catch (SQLException e) {
            Log.e("DIRETODATABASE-ERRO", e.getMessage());
        }
        return usuarios[0];
    }

    @Override
    protected void onPostExecute(Usuario usuario) {
        mainActivity.trataRetornoChamadaServidorBancoDeDados(usuario);
    }
}
```

Interface utilizada em conjunto com a biblioteca Retrofit que configura método HTTP e objetos de entrada e saída na comunicação com webservice:

```
package com.example.clsalgado.tccws.service;

import com.example.clsalgado.tccws.modelo.Usuario;

import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.POST;

public interface LoginService {

    @POST("login")
    Call<ResponseBody> login(@Body Usuario usuario);
}
```

Classe que configura a biblioteca Dagger para possibilitar a injeção de dependências nas classes que fazem comunicação com a tela:

```
package com.example.clsalgado.tccws.component;

import com.example.clsalgado.tccws.MainActivity;
import com.example.clsalgado.tccws.UsuarioActivity;
import com.example.clsalgado.tccws.module.TccModule;

import dagger.Component;
```

```
@Component(modules = TccModule.class)
public interface TccComponent {

    void inject(MainActivity activity);

    void inject(UsuarioActivity activity);
}

package com.example.clsalgado.tccws.app;

import android.app.Application;

import com.example.clsalgado.tccws.component.DaggerTccComponent;
import com.example.clsalgado.tccws.component.TccComponent;

public class TccApplication extends Application {

    private TccComponent component;

    @Override
    public void onCreate() {
        super.onCreate();
        component = DaggerTccComponent.builder().build();
    }

    public TccComponent getComponent() {
        return component;
    }
}
```

APÊNDICE B - Código fonte do dispositivo webservice

Repositório no github:

Arquivo xml que define todas as dependências do projeto web. No caso desse projeto foi utilizado o maven como gerenciador de dependências:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>br.com.utfpr.tcc.ws.spring</groupId>
	<artifactId>wsjpa</artifactId>
	<packaging>war</packaging>
	<version>0.0.1-SNAPSHOT</version>
	<name>wsjpa Maven Webapp</name>
	<url>http://maven.apache.org</url>
	<dependencies>

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>4.1.6.RELEASE</version>
		</dependency>

		<dependency>
			<groupId>org.mariadb.jdbc</groupId>
			<artifactId>mariadb-java-client</artifactId>
			<version>1.7.3</version>
		</dependency>

		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-entitymanager</artifactId>
			<version>4.3.0.Final</version>
		</dependency>

		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>4.3.0.Final</version>
		</dependency>

		<dependency>
			<groupId>org.hibernate.javax.persistence</groupId>
			<artifactId>hibernate-jpa-2.1-api</artifactId>
			<version>1.0.0.Final</version>
		</dependency>

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-orm</artifactId>
			<version>4.1.0.RELEASE</version>
		</dependency>

		<dependency>
	<groupId>com.fasterxml.jackson.core</groupId>
	<artifactId>jackson-databind</artifactId>
```

```
            <version>2.2.3</version>
        </dependency>

                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>jstl</artifactId>
                        <version>1.2</version>
                </dependency>

                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>3.8.1</version>
                        <scope>test</scope>
                </dependency>

                <!-- https://mvnrepository.com/artifact/javax.servlet/servlet-api -->
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>servlet-api</artifactId>
                        <version>2.3</version>
                        <scope>provided</scope>
                </dependency>
        </dependencies>
        <build>
                <finalName>wsjpa</finalName>
        </build>
</project>
```

Classes de configuração do container web que fazem a configuração do framework web Spring MVC.

```
package br.com.utfpr.tcc.ws.spring.wsjpa.config;

import javax.servlet.Filter;

import org.springframework.web.filter.CharacterEncodingFilter;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitiali
zer;

public class ServletSpringMVC extends
AbstractAnnotationConfigDispatcherServletInitializer {

        @Override
        protected Class<?>[] getRootConfigClasses() {
                return new Class[] {AppWebConfiguration.class, JPAConfiguration.class};
        }
```

```
	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class[] {};
	}

	@Override
	protected String[] getServletMappings() {
		return new String[] {"/"};
	}

	@Override
	protected Filter[] getServletFilters() {
		CharacterEncodingFilter encodingFilter = new CharacterEncodingFilter();
		encodingFilter.setForceEncoding(true);
		encodingFilter.setEncoding("UTF-8");

		return new Filter[] {encodingFilter};
	}

}
```

Classe de configuração do Hibernate para utilização de JPA para comunicação com a base de dados.

```
package br.com.utfpr.tcc.ws.spring.wsjpa.config;

import java.util.Properties;

import javax.persistence.EntityManagerFactory;

import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.JpaVendorAdapter;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@EnableTransactionManagement
public class JPAConfiguration {
	@Bean
	public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
		LocalContainerEntityManagerFactoryBean factoryBean = new
LocalContainerEntityManagerFactoryBean();

		JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
```

```
		factoryBean.setJpaVendorAdapter(vendorAdapter);

		DriverManagerDataSource dataSource = new DriverManagerDataSource();
		dataSource.setUsername("root");
		dataSource.setPassword("root");
		dataSource.setUrl("jdbc:mariadb://localhost:3306/utfpr_tcc_wsjpa");
		dataSource.setDriverClassName("org.mariadb.jdbc.Driver");

		factoryBean.setDataSource(dataSource);

		Properties properties = new Properties();
		properties.setProperty("hibernate.dialect",
"org.hibernate.dialect.MySQL5Dialect");
		properties.setProperty("hibernate.show_sql", "true");
		properties.setProperty("hibernate.format_sql", "true");
		properties.setProperty("hibernate.hbm2ddl.auto", "update");

		factoryBean.setJpaProperties(properties);

		factoryBean.setPackagesToScan("br.com.utfpr.tcc.ws.spring.wsjpa.model");

		return factoryBean;
	}

	@Bean
	public JpaTransactionManager transactionManager(EntityManagerFactory emf) {
		return new JpaTransactionManager(emf);
	}
}
```

Classe modelo que representa a tabela utilizada no nosso banco de dados.

```
package br.com.utfpr.tcc.ws.spring.wsjpa.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Usuario {

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int matricula;

	private String senha;

	private int contador;
```

```
	public int getMatricula() {
		return matricula;
	}

	public void setMatricula(int matricula) {
		this.matricula = matricula;
	}

	public String getSenha() {
		return senha;
	}

	public void setSenha(String senha) {
		this.senha = senha;
	}

	public int getContador() {
		return contador;
	}

	public void setContador(int contador) {
		this.contador = contador;
	}

	public boolean mesmaSenha(String senha) {
		return this.senha.equals(senha);
	}
}
```

Classe modelo que representa o objeto de resposta enviado pelos servidor quando o webservice é acionado.

```
package br.com.utfpr.tcc.ws.spring.wsjpa.model;

public class Resposta {

	private int httpStatus;

	private String mensagemErro;

	private Object retorno;

	public Resposta(int httpStatus, String mensagemErro) {
		this.httpStatus = httpStatus;
		this.mensagemErro = mensagemErro;
	}

	public Resposta(int httpStatus, String mensagemErro, Object retorno) {
```

```
		this.httpStatus = httpStatus;
		this.mensagemErro = mensagemErro;
		this.retorno = retorno;
	}

	public int getHttpStatus() {
		return httpStatus;
	}

	public String getMensagemErro() {
		return mensagemErro;
	}

	public Object getRetorno() {
		return retorno;
	}
}
```

Classe DAO que contêm todos as operações entre o webservice e a base de dados utilizadas no contexto desse projeto.

```
package br.com.utfpr.tcc.ws.spring.wsjpa.dao;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;

import org.springframework.stereotype.Repository;

import br.com.utfpr.tcc.ws.spring.wsjpa.model.Usuario;

@Repository
@Transactional
public class UsuarioDAO {

	@PersistenceContext
	private EntityManager manager;

	public Usuario busca(int matricula) {
		return manager.find(Usuario.class, matricula);
	}

	public void atualizar(Usuario usuario) {
		manager.merge(usuario);
	}

	public void inserir(Usuario usuario) {
```

```
		manager.persist(usuario);
	}

	public void remover(Usuario usuario) {
		manager.remove(usuario);
	}
}
```

Classes que disponibilizam recursos acessíveis na web (endpoints), que podem ser acessados através de métodos HTTP enviando e recebendo JSON.

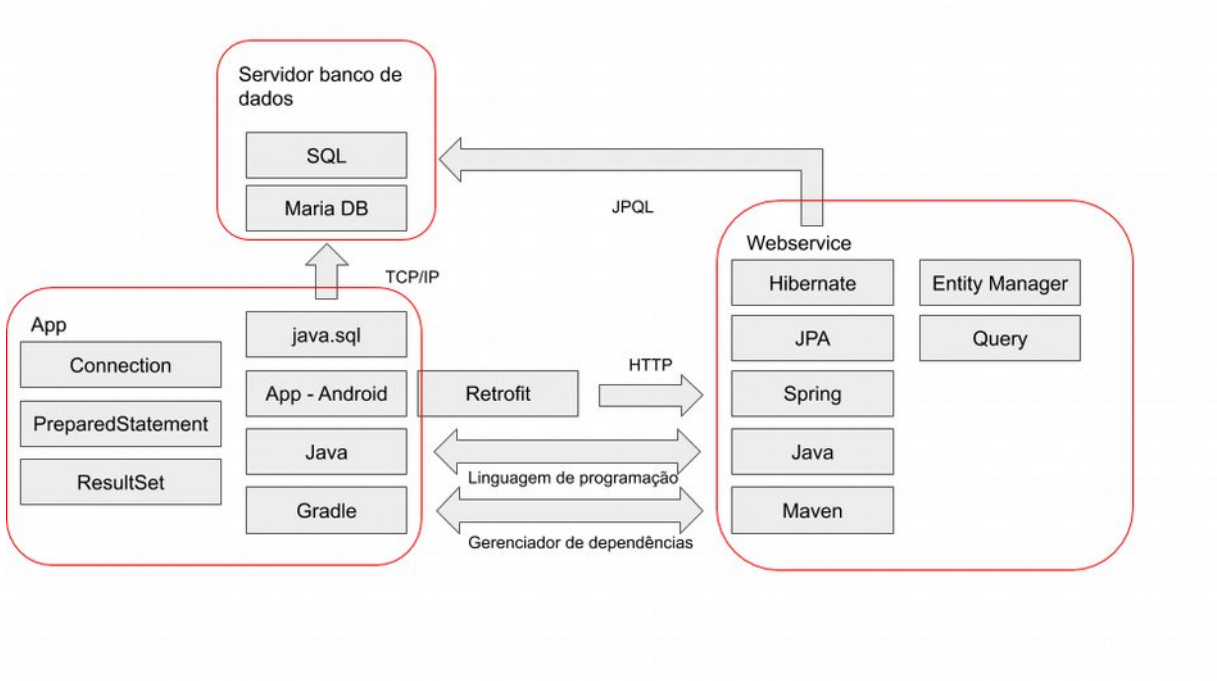
```
package br.com.utfpr.tcc.ws.spring.wsjpa.restcontroller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import br.com.utfpr.tcc.ws.spring.wsjpa.dao.UsuarioDAO;
import br.com.utfpr.tcc.ws.spring.wsjpa.model.Resposta;
import br.com.utfpr.tcc.ws.spring.wsjpa.model.Usuario;

@RestController
@RequestMapping(consumes = "application/json", produces = "application/json")
public class LoginController {

	@Autowired
	private UsuarioDAO usuarioDAO;

	@RequestMapping(value="/login", method=RequestMethod.POST)
	public ResponseEntity<Resposta> login(@RequestBody Usuario usuarioEntrada) {

		Usuario usuarioLogin = usuarioDAO.busca(usuarioEntrada.getMatricula());

		if(usuarioLogin == null) {
			return
configuraRetornoErro(HttpStatus.INTERNAL_SERVER_ERROR, "Usuario inexistente");
		}

		if(!(usuarioLogin.mesmaSenha(usuarioEntrada.getSenha()))) {
			return configuraRetornoErro(HttpStatus.PRECONDITION_FAILED,
"Usuario ou senha incorretos");
		}

		return new ResponseEntity<Resposta>(HttpStatus.OK);
```

```
	}

	@RequestMapping(value="/cadastrar", method=RequestMethod.POST)
	public ResponseEntity<Resposta> cadastrar(@RequestBody Usuario
usuarioEntrada) {

		try {
			usuarioDAO.inserir(usuarioEntrada);
		} catch (Exception e) {
			e.printStackTrace();
			return new
ResponseEntity<Resposta>(HttpStatus.INTERNAL_SERVER_ERROR);
		}

		return new ResponseEntity<Resposta>(HttpStatus.OK);
	}

	@RequestMapping(value="/cadastrar", method=RequestMethod.PUT)
	public ResponseEntity<Resposta> atualizar(@RequestBody Usuario
usuarioEntrada) {

		try {
			usuarioDAO.atualizar(usuarioEntrada);
		} catch (Exception e) {
			e.printStackTrace();
			return new
ResponseEntity<Resposta>(HttpStatus.INTERNAL_SERVER_ERROR);
		}

		return new ResponseEntity<Resposta>(HttpStatus.OK);
	}

	@RequestMapping(value="/cadastrar", method=RequestMethod.DELETE)
	public ResponseEntity<Resposta> remover(@RequestBody Usuario
usuarioEntrada) {

		try {
			usuarioDAO.remover(usuarioEntrada);
		} catch (Exception e) {
			e.printStackTrace();
			return new
ResponseEntity<Resposta>(HttpStatus.INTERNAL_SERVER_ERROR);
		}

		return new ResponseEntity<Resposta>(HttpStatus.OK);
	}
	private ResponseEntity<Resposta> configuraRetornoErro(HttpStatus status, String
msg) {
		return new ResponseEntity<Resposta>(new Resposta(status.value(), msg),
status);
```

```
        }
}
```

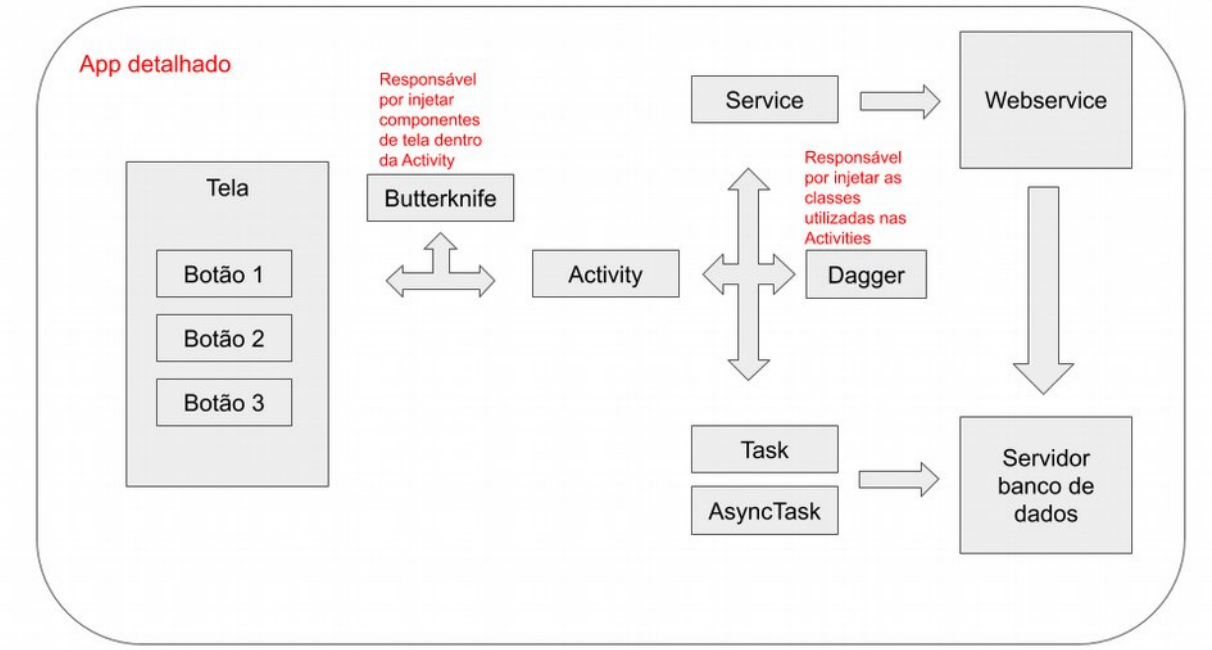
APÊNDICE C - Diagramas e tabela utilizada no banco de dados

Diagrama de solução/tecnologias utilizadas



Fonte: autoria própria.

Diagrama detalhado App/Bibliotecas utilizadas



Fonte: autoria própria.

Tabela utilizada/Operações efetuadas

Tabela Usuário		
nome campo	tipo	propriedades
matricula	int(11)	primary key
contador	int(11)	not null
senha	varchar	null

Consultas efetuadas	
Tipo operação	SQL
Consulta	select * from usuario where matricula = 1505602 and senha = "789";
Inserção	insert into usuario values(1, "789");
Atualização	update usuario set senha = "123" where contador = 1;

Fonte: autoria própria.