INSTITUTO DE EDUCAÇÃO SUPERIOR DA PARAÍBA – IESP
SISTEMA DE INFORMAÇÃO
ERIVAN JOSÉ CORREIA JUNIOR

CONSTRUÇÃO DE UM APLICATIVO DE APOIO A DECISÃO SOBRE ASPECTOS RELACIONADOS À ARQUITETURA DE *SOFTWARE*

CABEDELO - PB
2019

ERIVAN JOSÉ CORREIA JUNIOR

CONSTRUÇÃO DE UM APLICATIVO DE APOIO A DECISÃO SOBRE ASPECTOS RELACIONADOS À ARQUITETURA DE *SOFTWARE*

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação no Instituto de Educação Superior da Paraíba, como requisito parcial para a Obtenção do grau de Bacharel em Sistemas de Informação.

ORIENTADOR: Prof. Dsc. Alana Marques de Morais

CABEDELO - PB
2019

ERIVAN JOSÉ CORREIA JUNIOR

CONSTRUÇÃO DE UM APLICATIVO DE APOIO A DECISÃO A GESTORES DE TI EM RELAÇÃO À ARQUITETURA DE *SOFTWARES*

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação no Instituto de Educação Superior da Paraíba, como requisito parcial para a Obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado em: _____ de _____ de 2019.

BANCA EXAMINADORA

_____

Prof. Alana Morais
Instituto de Educação Superior da Paraíba

_____

Prof.. XXXX
Instituto de Educação Superior da Paraíba

_____

Prof.. XXXX
Instituto de Educação Superior da Paraíba

À minha família e amigos

AGRADECIMENTOS

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema móvel, que auxilia estudantes e profissionais a escolherem a arquitetura de *software* que se aplica melhor a sua necessidade de projeto, para que não haja desperdício de esforços no engajamento e construção de um sistema por utilizar uma estratégia equivocada. O sistema foi desenvolvido na linguagem KOTLIN, utilizando a ferramenta de desenvolvimento Android Studio, e durante o processo de construção foram aplicados conceitos da engenharia de *software.* Dentre as arquiteturas abordadas durante o trabalho, destacou-se as arquitetura de microsserviços e a arquitetura de monolítica, expondo suas características e usabilidade.

PALAVRAS-CHAVE: Arquitetura de *software*; Desenvolvimento *mobile*; Kotlin.

ABSTRACT

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

É fato que o conteúdo presente na *Web* tem se tornado cada vez maior, seja na quantidade de materiais publicados, quanto os sistemas que atendem as demandas de diversos usuários, e com isso surge a necessidade de aplicações que conduzam a um gerenciamento e tratamento do fluxo de recursos resultante do processamento das informações.

Os sistemas desenvolvidos com a finalidade de gerir recursos, vem passando por constantes adaptações conforme as exigências e ocasiões levantadas, sob a justificativa de explorar ou resolver uma determinada área. Essas adaptações são padronizadas e trabalhadas até o momento em que sua aplicabilidade não corresponder com o surgimento de um novo problema.

Uma arquitetura de *software* define a estrutura e os padrões que irão compor uma aplicação, facilitando o andamento do desenvolvimento ao quebrar o nível de complexidade em partes menores para que seja possível o entendimento e gerenciamento de cada componente a ser construído (ENGEHOLM, 2010).

Sabendo da importância que uma arquitetura de *software* apresenta no processo de de análise e desenvolvimento de um sistema, torna-se inviável não recorrer a tal auxílio, e o arquiteto de *software* precisa entender o problema de negócio, entender as arquiteturas, a combinação de tais entendimentos produzem uma decisão acerca do projeto, mitigando as falhas e melhorando a produtividade. A existência desse auxílio permite um maior conforto em saber que no decorrer do projeto será realizado seguindo passos fundamentados para chegar a um final estabelecido, porém existem vários padrões arquiteturais, dificultando a escolha de qual utilizar.

O gestor de projeto ou o arquiteto de *software,* além de saber tais técnicas, precisa entender em quais circunstâncias são necessário aplicá-las, pois uma decisão como essa pode impactar no planejamento, no escopo, na vida útil do produto final.

Definir a estrutura e os padrões que irão estar presentes no desenrolar dos serviços propostos, não garante futuras modificações, com a intenção de melhoria

e/ou adaptação, afinal trata-se da tecnologia, que sempre passará por mudanças, conforme as necessidades e exigências.

Um exemplo que reforça essa idéia, é uma das plataformas compartilhadas, que recebe uma grande quantidade de acessos diariamente, a Netflix, inicialmente construída baseada em uma arquitetura monolítica, observou a necessidade de melhoria, ocasionada pela crescente massa de requisições a ser atendidas, e desejo de maior velocidade de resposta.

Com a vontade de alcançar melhorias, foi adotado uma arquitetura de microsserviços, algo desafiador a um sistema tão robusto e consagrado, dividindo casa serviços em fatias menores, funcionando separadamente, garantindo que o sistema irá funcionar mesmo que uma dessas fatias esteja sujeita a falhas.

Ao observar o andamento de alguns anos atrás e as mudanças relacionadas a tecnologia de acordo com os espaços de tempo por período, é possível notar a popularização de microcomputadores.

O resultado de tal fato provocou uma disseminação e a facilidade entre pessoas de todas as classes possuírem um computador, como ferramenta indispensável e não apenas isso, mudanças significativas como, TVs inteligentes, computação móvel, crescimento do alcance da internet e a forma que é distribuída (LAURINDO, 2002).

Mudanças como estas impactam na forma como são construídas as aplicações *Web*, exigindo cada vez mais mobilidade, heterogeneidade, execução concorrente de programas.

Neste contexto, o que atende os requisitos citados acima, são os sistemas distribuídos, revelando a importância de explorar a área, as tecnologias que cercam esse conceito, e principalmente as arquiteturas pensadas para a distribuição do sistema por meio da web (COULOURIS et al., 2013).

O presente trabalho entende que ter como objeto de estudo e pesquisa as arquiteturas distribuídas, é necessário para investigar as constantes mudanças que contribuem na modernização, aprimoração e construção de sistemas que não

dependem de especificações de sistemas operacionais, e não se limitam a *hardwares*.

Considerando as discussões mencionados, o presente trabalho vai de encontro a compreensão do estado da arte, problemas e soluções relacionadas a um cenário de arquiteturas distribuídas.

1.2 OBJETIVO GERAL

Planejar e Construir um *software*, cuja finalidade é auxiliar o gestor de projeto na tomada de decisão a respeito da demanda de construção de um sistema *Web* com arquitetura distribuída.

1.2.1 OBJETIVOS ESPECÍFICOS

- Investigar e entender as característica de um sistema distribuído e monolítico.
- Demonstrar a importância dos sistemas distribuídos e monolíticos.
- Discutir a evolução dos padrões envolvidos na construção dos sistemas distribuídos e monolíticos.
- Planejar e construir um aplicativo de apoio à tomada de decisão do gestor.

1.3 ESTRUTURA DO DOCUMENTO

Este documento foi organizado da seguinte forma. O capítulo 2 apresenta o referencial teórico, com as tecnologias utilizadas, necessárias para o desenvolvimento da aplicação em questão. No capítulo 3, é detalhada a metodologia na qual foi escolhida para a construção do projeto. No capítulo 4, é apresentado o *software* EasyArquiterura, junto de suas funcionalidades e os resultados obtidos no processo de desenvolvimento. E por fim, o capítulo 5 é finalizado o trabalho por meio da conclusão do processo desenvolvido neste trabalho.

REFERENCIAL TEÓRICO

Mediante a importância que os *softwares* desempenham no auxílio organizacional das demandas em diferentes cenários, em um sistema de estoque ou gerenciando a parte contábil de uma empresa, torna-se inviável a falta de um *software* que contribua no andamento de suas ações.

Segundo Zimmerer (1994), são notórias as vantagens da utilização de sistemas informatizados em pequenas empresas como: a melhoria das informações na tomada de decisões, a automatização de tarefas que são rotineiras e melhoria do controle interno das operações. Acompanhando esse pensamento, é possível ver que o mercado de trabalho tem entendido a importância da tecnologia como um meio que impulsione a melhoria de serviços em contextos distintos.

A ABES (Associação Brasileira das Empresas de *Softwares*) em suas pesquisas anuais tem realizado um levantamento animador sobre os investimentos em TI no Brasil, abrangendo *hardwares*, *softwares* e serviços.Em 2017, foi estimado um crescimento de cerca de 4,5% no mercado de TI, ocasionado por um valor de investimento de U$ 38 bilhões, sendo o líder investidor da área, na américa latina e o nono lugar no *ranking* mundial (ABES, 2017). Não satisfeito com os dados adquiridos neste ano, em 2018 a pesquisa voltou a surpreender, e o crescimento dos investimentos na área de TI foi de 9,8%, e um valor de U$42 bilhões.

Dando um foco considerável nas formas de planejar e realizar os processos de construção de *software*, com menores custos e maior produtividade, papel fundamental da engenharia de *software*.

2.1 ENGENHARIA DE *SOFTWARE*

A engenharia de *software* é uma aliada na construção de sistemas por compreender os processos necessários para a realização do objetivo final. Ela está diretamente ligada a qualidade com o qual será apresentado o produto final (ENGHOLM, 2010). Por ela, é possível avaliar se os requisitos funcionais foram

atendidos e implementados, de acordo com o que foi solicitado, por meio de documentos que centralizam as ideias em questão.

Ao se privar o uso da engenharia de *software*, grandes impactos podem ser evidenciados no andamento de um projeto, como a falta da compreensão do que está sendo construído e má elaboração de código. As consequências podem estar presentes após a finalização do projeto, como erros em partes do sistema, dificuldade de lançar atualizações, um baixo desempenho e dificuldade de uso.

Pressman e Maxim (2016) defendem uma ideia de que a engenharia de *software* possui uma metodologia de processos descrita de forma genérica possuindo cinco atividade principais, que são, comunicação, planejamento, modelagem, construção e entrega. A comunicação é parte do início de qualquer projeto, para que se tenha uma colaboração maior entre os agentes envolvidas, entendendo os requisitos de maneira coesa. No planejamento, é realizado um esforço no qual todo o trabalho será guiado por uma espécie de mapa, conduzindo a equipe no decorrer do projeto, dividindo os papéis e tarefas de cada atuante, riscos, recursos, além de estimar os custos. A atividade de modelagem está responsável por apresentar um esboço, representando uma visão macro do que é foi prometido, geralmente apresentados por diagramas UML e de classe. O que foi planejado é construído, a atividade de construção é direcionada a parte de codificação, e testes para garantir que tudo o que foi construído encontra-se com qualidade. Após realização de todas as atividade anteriores, é realizada a entrega do produto, e assim submetida a avaliações por meio de *feedback*.

Um dos processos indispensáveis descritos pela engenharia de *software* é a utilização de uma arquitetura de *software*, para estruturação, construção e manutenção, de um projeto.

2.2 ARQUITETURA DE *SOFTWARE*

Ao citar a palavra arquitetura, o primeiro pensamento obtido, se dá pela associação as construções urbanísticas, porém seu significado não está restrito a

esse sentido (LEMOS, 1980). Em uma compreensão geral, uma definição que pode abstraí-la de forma genérica, é afirmar sua força como um planejamento, contendo decisões iniciais que impactarão o andamento de um determinado projeto.

As arquiteturas de *softwares* não estão distantes dessa realidade, pensando sempre na organização estrutural do projeto a ser finalizado. Conseguindo parametrizar os passo iniciais, indispensáveis para as futuras tomadas de decisões.

De acordo com Sommerville(2011):

> "*O projeto de arquitetura está preocupado com a compreensão de como um sistema deve ser organizado e com a estrutura geral desse sistema. No modelo do processo de desenvolvimento de software, o projeto de arquitetura é o primeiro estágio no processo de projeto de software.*" (SOMMERVILLE, 2011, p.103)

Dessa forma, a arquitetura de *software* é uma estrutura que envolve os variados elementos do sistema, informando como esses elementos se relacionam ou como devem se relacionar.

> *"A arquitetura de software de um sistema é o conjunto de estruturas necessárias para raciocinar sobre o sistema, que compreendem elementos de software, relações entre eles e propriedades de ambos.*" (Len Bass *et al.*, 2012).

As concentrações dos esforços submetidos para a realização de um projeto de sistema, não são mais voltados as linguagem de programação, após entender o papel de uma arquitetura de *software*. A junção desses dois ativos agregam e fornecem uma visão moderna e complexa dos métodos responsáveis pela criação de *softwares*, e a manutenção e reformulação.

Ao discutir sobre a arquitetura de *software,* é comum destacar dois modelos de arquitetura que possuem uma maior popularidade, e se adequam mediante a uma necessidade, sendo o modelo monolítico e de microsserviços.

2.2.1 Arquitetura Monolítica

O modelo monolítico está presente em muitos projetos atuais e seu uso varia conforme a caracterização de escopo de um projeto, mesmo sendo usado

atualmente, é o mais antigo modelo entre os citados anteriormente (NEWMAN, 2015). Essa arquitetura descreve um sistema que engloba em um único projeto todos os módulos que irão atender os requisitos funcionais exigidos, como por exemplo um *login*, listagem, realização de pedidos, ou seja uma única aplicação está responsável por todos os processos.

Aplicações desse tipo contam com um diferencial de serem auto-suficientes e independentes de outras aplicações, partindo do princípio de que o aplicativo não irá se preocupar com apenas uma tarefa, mas poderá executar todos os passos exigidos para finalizar o objetivo maior.

Sistemas como esses possuem uma tendência de crescer dependendo da pretensão do projeto. Com uma complexidade maior exigida, os números de pacotes e a quantidade de classes e código serão maiores, gerando futuros problemas de manutenção.

Os impactos de uma falha no produto podem percorrer mais módulos, tendo como consequência a perda momentânea de mais de uma serviço, e na pior das hipóteses todo o sistema.

A figura 1 ilustra como uma arquitetura monolítica divide a interface do usuário (camada UI), a lógica de negócio (*business logic*) e uso de armazenamento (*data access layer*).

Projetos monolíticos possuem uma característica popular por utilizar um padrão arquitetural, como o MVC(*Model-View-Controller*) ou o MVP(Model-View-Presenter). O EasyArquitetura utiliza o padrão de arquitetura baseado em um padrão muito conhecido pela comunidade de desenvolvedores, estudantes e entusiastas do segmento, o padrão MVC(BUSCHMANN, 2016). O padrão foi escolhido por possibilitar a realização de divisões do projeto em camadas previamente definidas. Sua utilização auxilia no isolamento de partes que representam regras de negócio, da lógica empregada e da interação com o usuário.

Destacam-se três camadas bases representadas neste padrão arquitetural, são elas: Modelo, Visualização e Controlador, podendo haver em uma aplicação

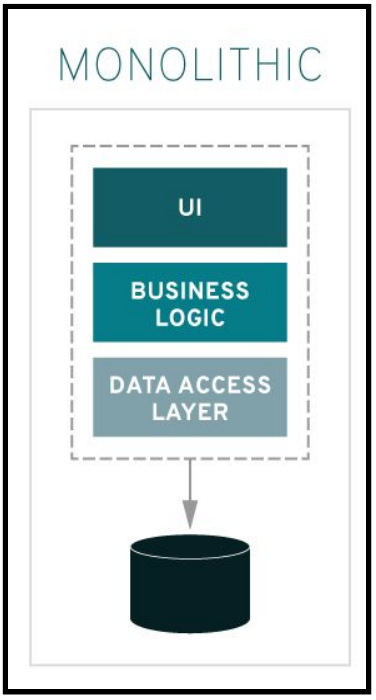
vários modelos, várias estruturas de visualização e vários controladores, o que pode determinar a quantidade de cada uma dessas camadas será o escopo do projeto.

Na camada de modelo, estão representados de forma abstrata componentes ou partes de uma representação de um sistema com o intuito de armazenar dados conforme a inserção determinada pela aplicação.

Na camada de visualização, são destacados estratégias de apresentação para que o usuário permita-se interagir com o *software* da melhor forma. No presente estudo, a camada de visualização foi construída com o auxílio do Android Studio que auxilia na construção de interfaces por meio de arquivos XML.

E por fim o padrão arquitetural MVC possui controladores, estes são responsáveis por realizar a organização dos dados que irão percorrer o sistema, recebendo informações e por sua vez reagindo com respostas.

Figura 1 - Arquitetura monolítica



Fonte: Red Hat (2018)

2.2.2 Arquitetura de Microsserviços

A arquitetura de microsserviços é uma abordagem moderna que tem ganhado espaço cada vez mais. Destacado por sua praticidade e maior velocidade de construção de aplicações.

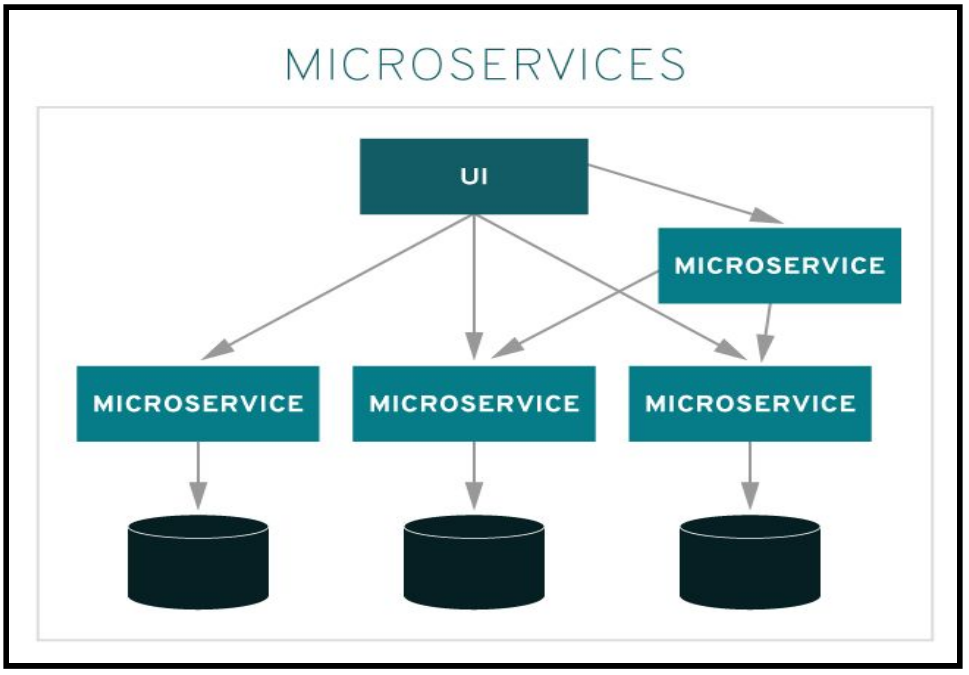
> "*O termo 'Arquitetura de Microsserviços' surgiu nos últimos anos para descrever uma maneira particular de projetar aplicativos de software como suítes de serviços implementáveis independentemente. Embora não haja uma definição precisa desse estilo arquitetônico, existem certas características comuns em torno da organização, da capacidade comercial, implantação automatizada, inteligência nas chamadas da interface do sistema e controle descentralizado de idiomas e dados*." (FOWLER, 2017)

O sistema é construído por outros pequenos sistemas que serão alinhados formando o todo. Cada aplicação funciona de forma independente, no entanto o sistema em si depende de cada um desses pequeno sistema.

Dessa forma, a interação com o usuário de dá por diferentes aplicações em um único sistema. Diferente do modelo monolítico que conta com um banco de dados para todo o sistema, uma arquitetura de microsserviços conta com um banco de dados para cada microsserviço.

A figura 2 exibe como é dividido a estrutura de uma arquitetura de microsserviços. É possível observar na figura 2 que a aplicação apresentada é composta por quatro microsserviços distintos, três bases de dados independentes e uma interface única de conversação com o cliente.

Figura 2 - Arquitetura de microsserviço



Fonte: Red Hat (2018)

2.3 TECNOLOGIAS

As tecnologias escolhidas para a construção do aplicativo foram o sistema operacional Android, banco de dados SQLite, linguagem de programação Kotlin, e a ferramenta que possibilita a empregabilidade de tal linguagem, o Android Studio como IDE de desenvolvimento.

2.3.1 ANDROID

O Android é um sistema operacional móvel *Open Source* desenvolvido pela Google, cuja arquitetura foi baseada no *kernel* Linux versão 2.6 para tarefas do sistema como gerenciamento de memória e processos.

De acordo com Tellier (2011), este sistema foi criado em 2006, por meio de um projeto de uma jovem empresa americana, a Android *Inc.*, dando vida a uma nova plataforma operacional baseada em linux para dispositivos móveis, sendo posteriormente adquirida pela Google.

Em 2008, o Android tornou-se *Open Source,* seu código foi publicado como AOSP(*Android Open Source Project*) e um ano mais tarde foi lançado o primeiro aparelho celular com esse sistema operacional: o HTC G1.

A versão Android escolhida para aplicar ao projeto foi a 4.4 *Ice Cream Sandwich*, mesmo havendo diversas versões lançadas na história do Android, entre as principais se destacam (ANDROID, 2019):

- Android 1.1 - Fevereiro de 2009;
- Android 1.5 Cupcake - Maio de 2009;
- Android 1.6 Donut - Setembro de 2009;
- Android 2.1 Eclair - Outubro de 2009;
- Android 2.2 FroYo - Maio de 2010;
- Android 2.3 Gingerbread - Dezembro de 2010;
- Android 3.0 Honeycomb - Fevereiro de 2011;
- Android 4.0 Ice Cream Sandwich - Outubro de 2011;
- Android 4.1 Jelly Bean - Julho de 2012;
- Android 4.4 KitKat - Outubro de 2013;
- Android 5.0 Lollipop - Novembro de 2014;
- Android 6.0 Marshmallow - Outubro de 2015;
- Android 7.0 Nougat - Agosto de 2016;
- Android 8.0 Oreo - Agosto de 2017;
- Android 9.0 Pie - Julho de 2018;
- Android 10 - Setembro de 2019.

Um forte diferencial desse sistema é o fato de que no mercado *mobile* é a primeira plataforma de código aberto, possibilitando a inclusão de diversos fabricantes utilizarem do sistema operacional, causando uma maior competitividade comercial. *“O Android é código aberto e distribuído sob licença Apache 2.0, o que quer dizer que você tem acesso aos códigos-fonte e também pode contribuir como projeto”* (MONTEIRO, 2012).

Uma das diversas peculiaridades desse sistema operacional, é sua ramificação e suporte para vários aparelhos digitais como câmeras, TVs e até videogames, não apenas *smartphones* e *tablets*.

Com a maior acessibilidade de dispositivos móveis, essa plataforma ganhou cada vez mais força e popularidade, conduzindo muitos desenvolvedores a aprender e produzir aplicações cada vez mais robustas e indispensáveis.

2.3.2 ANDROID STUDIO

Na construção de uma aplicação; seja ela um sistema *web*, *desktop* ou *mobile;* o uso de ferramentas que auxiliem no processo de criação é muito importante, para obter maior performance como a redução de tempo de criação, atuações precisas, uso de *debugger* apontando erros que podem acontecer ao escrever códigos.

Diversas IDE's são pensadas para atender as demandas de desenvolvedores que procuram pelo conforto de uma ferramenta amigável para criar suas aplicações. As mais conhecidas são: Intelij IDEA, Android Studio, NetBeans IDE e o Eclipse com o auxílio de pacotes da API do Android.

A IDE Android Studio oferece diversos benefícios, tais como: a facilidade de interação do usuário, recursos de edição visual, tornando mais fácil a manipulação e criação de telas com funcionalidades arrasta e solta. Criada pela Google, este editor de código, foi baseado no InteliJ IDEA da empresa Jetbrains, sendo mais um ponto positivo para esta ferramenta.

2.3.3 KOTLIN

Outra importante ferramenta para o presente projeto foi a linguagem Kotlin. Kotlin é uma linguagem de programação criada em 2011, pela *Jetbrains*, empresa

[1] https://developer.android.com
https://www.jetbrains.com
https://netbeans.org
https://www.eclipse.org

criadora de diversas IDEs no mercado, porém somente foi lançada sua versão estável em 2016 (KOTLINLAG, 2019).

A jovem linguagem tem ganhado espaço pouco a pouco, abordando uma inteligente estratégia de ser completamente interoperável com o Java, ou seja, é possível inserir código Java a partir de códigos Kotlin e vice-versa.

Durante muito tempo, as aplicações do sistema operacional Android eram desenvolvidas oficialmente pela linguagem de programação Java. Porém, na conferência para desenvolvedores Google IO 2017 foi anunciado que o Kotlin teria suporte como a segunda linguagem oficial de desenvolvimento Android (TITUS, 2017).

Java é uma linguagem que possui uma grande popularidade e sua aplicabilidade se estende em diversos contextos de desenvolvimento por meio do uso de sua JVM. Por outra perspectiva, a linguagem Kotlin teve como objetivo em seu projeto ser uma linguagem produtiva, segura e concisa a fim de se tornar uma alternativa ao Java, aproveitando sua JVM.

Um desenvolvedor Java possui uma menor curva de aprendizado ao tentar desbravar o desafio de aprender essa nova linguagem, pela simplicidade exposta por sua sintaxe (JEMEROV e ISAKOVA, 2017). Na figura 3, é demonstrado um exemplo de código Kotlin.

[2] https://kotlinlang.org

Figura 3 - Exemplo de código Kotlin

Fonte: Página kotlinlang (2019)

2.3.4 BANCO DE DADOS

O homem sempre observou a necessidade de deixar registrado momentos e informações que pudessem ser utilizadas posteriormente. Sendo assim, técnicas foram desenvolvidas para que isso acontecesse ao longo dos tempos, como pinturas pré-históricas, os hieróglifos dos egípcios, e a própria escrita(HEUSER, 2008). Na perspectiva dos sistemas computacionais, é possível observar uma expansão dos bancos de dados, arquivos de armazenamento, etc.

Quando se analisa o desenvolvimento de sistemas móveis, a necessidade de armazenamento é um desafio, pois existe uma limitação física para esta tarefa. Sendo assim, fabricantes e desenvolvedores de aplicações móveis, tendem a criar e aprimorar novos meios para que isso aconteça da forma mais objetiva, segura e rápida.

Date (2004) define um banco de dados como um sistema computadorizado que armazena e administra registros. Sua funcionalidade é guardar dados de modo que usuários possam buscar e utilizar tais dados em momentos oportunos.

> *"Um banco de dados é projetado, construído e populado com dados para uma finalidade específica. Ele possui um grupo definido de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados."*(ELMASRI; NAVATHE, 2011, p. 3)

O uso de um banco de dados traz vários benefícios a usuários e desenvolvedores por proporcionar uma organização na busca e armazenamento de dados, possibilitando acesso e tratamento das informações.

Torna-se essencial o uso de um banco de dados, pois as atividades executadas no cotidiano em sua grande maioria envolvem uma interação com banco de dados.

A forma adotada para persistir dados neste projeto foi o Sqlite, uma biblioteca escrita na linguagem C de código aberto, que possibilita o acesso a um banco de dados relacional sem a necessidade de um SGBD(Sistema de Gerenciamento de Banco de Dados).

Recomendado para aplicações simples, cujo fluxo de dados a serem processados não apresentam um volume grande, possuindo o objetivo de ser integrado a aplicações leves. Seu uso é indicado para sites que não possuirão uma grande demanda de acessos, aplicações *Desktop*, aplicações móveis, dispositivos e sistemas embarcados.

2.3.5 Processo de desenvolvimento de software

Com a preocupação de uma maior qualidade nas entregas produtivas de um *software* como um requisito básico, a engenharia de *software* concentra esforços na elaboração de estratégias, técnicas, métodos e processos de desenvolvimento de *software* para que isso aconteça (PRESSMAN; MAXIM, 2016).

Um processo de desenvolvimento de software, é uma junção de técnicas que visam ser aplicadas durante a construção e manutenção de um projeto, definindo caminhos a serem percorridos para que sejam atingidos os objetivos.

De acordo com Sommerville (2011), o processo de *software* é descrito como um conjunto de atividades relacionadas que possuem como objetivo final a

condução até um produto de *software.* Atividades que devem incluir quatro atividades, sendo: especificação de *software,* implementação, validação e evolução.

O conhecimento e aplicação dessas técnicas, são constantemente utilizadas pelo mercado de trabalho, por facilitar o entendimento, abstraindo para todos os integrantes da equipe as formas de construção de um produto.

Para tal compreensão, foram criadas metodologias que designam papéis e processos para alcançar um alto padrão de qualidade. As metodologias de processos de desenvolvimento de *software* são modelos que abstraem e explicam as diferentes formas de desenvolvimento de *software*.

É possível destacar o modelo cascata como uma forma tradicional, que apresenta todo o planejamento e documentação do projeto, para que seja então implementado, destacando a projeção antes da construção.

Ao contrário da abordagem em cascata, existem formas modernas de processos que são conhecidas como metodologias ágeis, que surgiram com uma proposta de serem métodos menos burocráticos de desenvolver um *software.*

O desenvolvimento ágil de *software* é uma abordagem de desenvolvimento que lida com problemas em ambientes que geralmente ocorrem mudanças repentinas(COCKBURN, 2000).

Um exemplo de metodologia ágil é o *scrum,* descrevendo um modelo incremental de desenvolvimento, aderindo uma forma direta de desenvolver. Baseando-se no conceito de *sprints,* que podem acontecer no intervalo de 2 a 4 semanas e reuniões diárias.

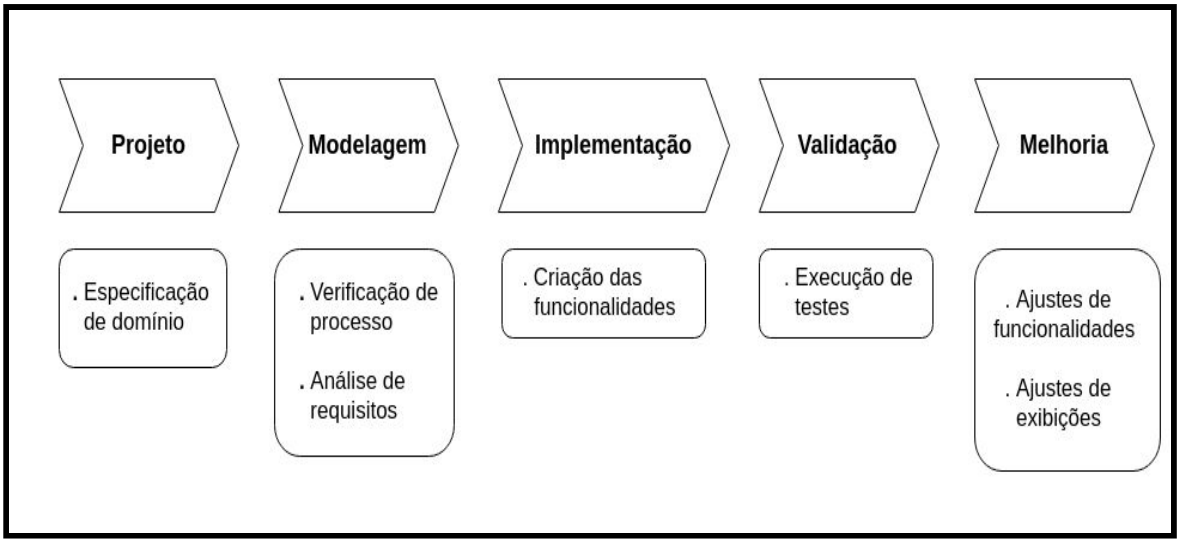
3 METODOLOGIA

A presente seção tem o intuito de discutir e apresentar a metodologia percorrida pelo trabalho discutido.

Inicialmente, este trabalho iniciou suas atividades com a execução de uma análise observacional do mercado para identificar lacunas e dificuldades levantadas pelos gestores referentes às arquiteturas de *softwares* e, com isso, planejar uma solução que auxilie o gestor em alguma etapa da construção de um aplicativo..

Para demonstrar a importância e seriedade do assunto abordado, foi realizado, por meio de um levantamento bibliográfico, questões da complexidade vivenciada por gestores de TI ao ater-se as arquiteturas que atendam os sistemas de modo geral. São contempladas 5 fases principais (Projeto, Modelagem, Implementação, Validação e Melhoria) para a execução do presente estudo e é possível observá-las graficamente na figura 4. Tais fases foram definidas de acordo com os princípios de engenharia de *software* apresentados no capítulo 2.

Figura 4 - Fases de estudo



Fonte: Próprio autor(2019)

- **Projeto:** Nesta fase, foram abordados o domínio do problema a ser estudado e as tecnologias relacionadas, consideradas necessárias para dar continuidade ao projeto.

- **Modelagem:** A fase de modelagem representa a etapa da visualização macro da ferramenta. Nesta fase, é possível abstrair as especificações do produto em conjunto com a realização do levantamento de requisitos funcionais e não funcionais.

- **Implementação:** Essa fase diz respeito à implementação das principais funcionalidades descritas e analisadas para o projeto.

- **Validação:** Esta fase preocupou-se com a qualidade da ferramenta foram elaborados casos de testes para garantir a integridade das funcionalidades planejadas.

- **Melhoria:** A melhoria representa a fase de evolução da ferramenta, realizando ajustes das funcionalidades já implementadas, e evoluções na camada de apresentação.

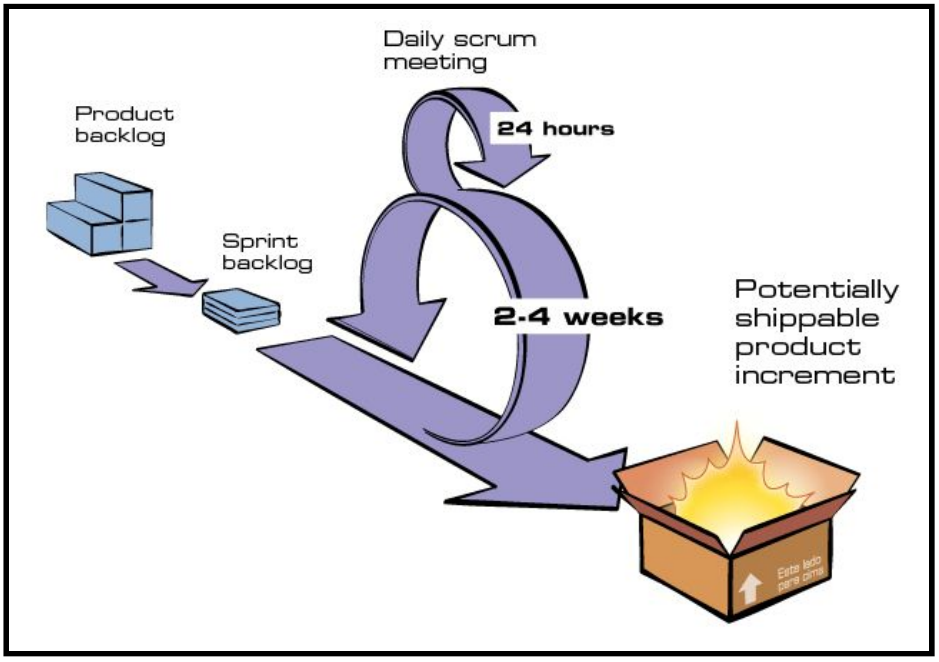
3.1 PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE*

No presente projeto, foi aplicado a metodologia de desenvolvimento baseada no *scrum*, motivado pela autoavaliação executada em cada ponto do projeto. Outra motivação para o uso do *scrum* foi a possibilidade de melhor lidar com as mudanças no projeto, pois dificilmente a elaboração de uma solução se dá antecipadamente, mas pela movimentação passo a passo em direção ao objetivo final, podendo haver recuos ao perceber erros e adaptação às mudanças.

A utilização de uma metodologia ágil baseada em *scrum* no projeto resultou na facilidade de execução de cada fase. Tendo como foco simplificar o escopo, deixando o entendimento para as concretizações das fases planejadas.

O projeto, fluiu de maneira assertiva com o uso de *sprints*, forma descrita pela metodologia que sugere ciclos, que são chamados de *sprints.* Cada ciclo representa um período de tempo fixo*,* obtendo uma série de pequenas atividades por ciclo. A seguir, na figura 5 é exibida o ciclo de vida do *scrum.* As etapas principais deste ciclo são: listagem de funcionalidades a serem implementadas contidas no *backlog*, execução de codificação durante uma *sprint*, revisões diárias durante o andamento da *sprint*, e a entrega das tarefas ao final do ciclo.

Figura 5 - Ciclos de vida *scrum*



Fonte: Pagina Desenvolvimento ágil (2019)

Houveram mudanças significativas para adaptação do *scrum* ao trabalho, por exemplo, não foi possível realizar a *daily* de acordo com a especificação da metodologia. Esta metodologia propõe reuniões diárias que são chamadas de *daily* e por não haver uma equipe composta por desenvolvedores, testadores e dono de produto, não foi necessário o uso da técnica.

O modo iterativo que uma metodologia ágil propõe para o processo de desenvolvimento do projeto foi muito válido para quebrar atividades dentro de cada fase do trabalho em pequenas espécies de histórias em cada bloco de tempo.

Sendo assim, foi possível entender qual o problema de estudo e em seguida modelar o sistema em uma *sprint*. Além disso, nos ciclos seguintes foi possível implementar pequenas funcionalidades seguidas de validações que comprovasse o seu funcionamento e ao término de cada um dos ciclos haviam avaliações para verificar se tudo estava ocorrendo conforme a modelagem e se as tarefas obtiveram sucesso para, se houvesse necessidade, iniciar uma nova iteração.

4 PROJETO E DESENVOLVIMENTO

Esta seção do documento apresenta o *software* EasyArquitetura, sua implementação, finalidades, requisitos, objetivos e público alvo.

O EasyArquitetura, é uma aplicação móvel construída com a linguagem de programação Kotlin, utilizando-se do Android Studio como IDE para codificação, sem o auxílio de qualquer *framework.* O sistema desenvolvido pretende atender a curiosidade de estudantes e auxiliar profissionais que procuram entender e se aprofundar nos conceitos sobre a arquitetura de *software.*

O EasyArquitetura é um sistema de apoio à decisão para o usuário que necessita empregar em sua aplicação uma arquitetura que seja coerente com a necessidade do projeto e ceder informação suficiente para futuros projetos.

A aplicação permite um controle de acesso de usuários por meio de uma tela de *login*, não havendo níveis de usuários, todo acesso ao sistema é irrestrito, livre para que qualquer pessoa obtenha informações sobre o que é o aplicativo, as curiosidades e características das arquiteturas descritas.

Diante das funcionalidades do aplicativo, é possível destacar o apoio às tomadas de decisão do gestor em relação à arquiteturas de *software*. Isso é possível por meio da aplicação de um questionário que provê os dados para uma calibração inicial do perfil do projeto iniciado e das experiências do gestor. Após as análises das respostas, o sistema conseguirá retornar ao gestor qual das arquiteturas pode ser mais adequado ao seu contexto, baseado no sistema de regras do aplicativo.

4.1 MODELAGEM DO SISTEMA

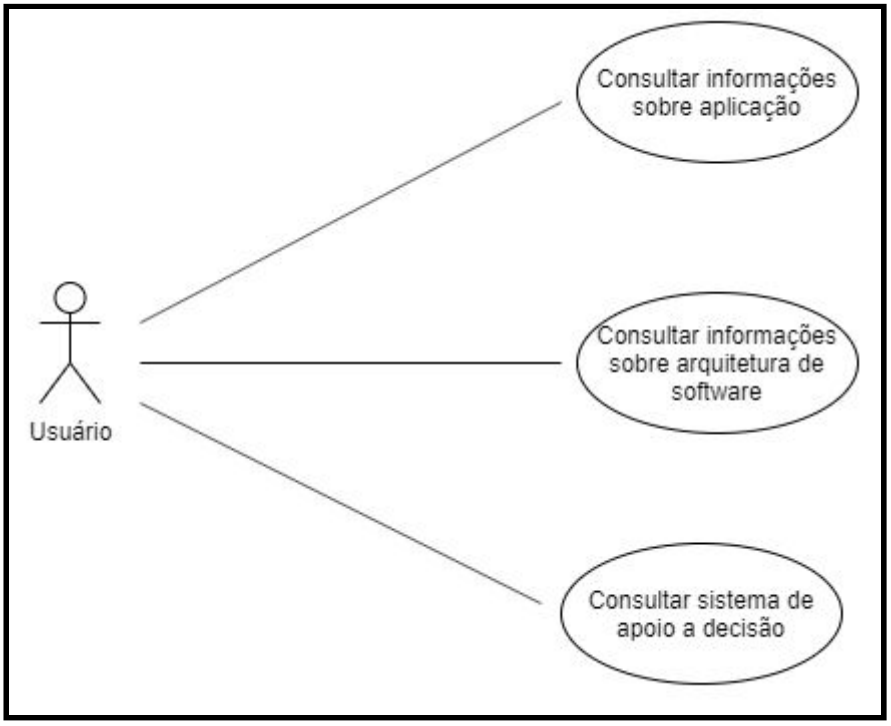
Ao construir qualquer aplicação, é necessário realizar um levantamento de informações que servem como uma importante e indispensável base de apontamento de idéias, para a projeção de como irá se comportar o sistema em questão.

Segundo Pressman (2016) e Maxim (2016), a modelagem é um processo da engenharia de *software* norteada por princípios fundamentais que ajudam a aplicar métodos significativos na construção de *software*.

A modelagem utiliza uma visão genérica de um sistema, usada nos passos da engenharia de *software*, para extrair os requisitos do sistema para que os responsáveis pelo desenvolvimento tenham em mente a ideia coerente do que se trata o projeto.

O diagrama de caso de uso consegue abstrair atores que interagem com o sistema, nomeando as ações provenientes das interações. Na figura 5, o diagrama de casos de uso que contém as funcionalidades do sistema é apresentado.

Figura 5 - Diagrama de caso de uso



Fonte: Próprio autor (2019)

Em um diagrama de caso uso representam-se atores, ou seja, uma definição individual daquele que interage com o sistema, por meio dessa representação é visualizado quais ações um ator pode efetuar no sistema. Na figura 4 exibida anteriormente, é possível identificar o ator sendo o usuário, no qual o mesmo poderá realizar a ação de cadastro, *login* caso esteja devidamente cadastrado, consultar informações sobre a aplicação, verificar informações sobre arquitetura de *software* e utilizar o questionário de apoio à decisão.

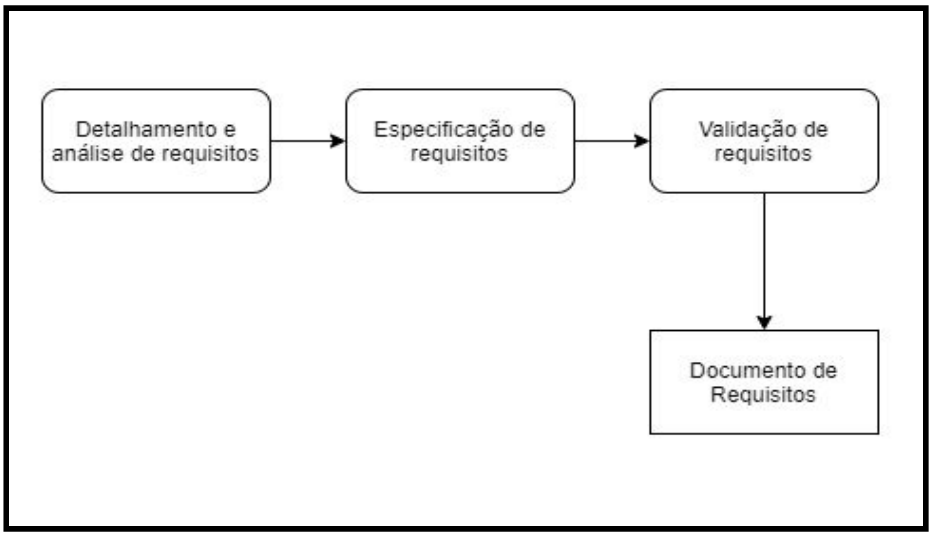
4.2 REQUISITOS DE SISTEMA

De acordo com Sommerville (2011), os requisitos de um sistema, são a formalização descrita do que se espera da aplicação, quais serviços são oferecidos e quais são as restrições apresentadas em seu uso.

Ao se identificar os requisitos do sistema, são tratadas junto com o cliente as exigências, ou seja, quais são as necessidades e desejos solicitados para prover um produto que satisfaça a necessidade do mesmo. Como produto desta etapa tem-se o documento de requisitos do sistema que define com exatidão o que será implementado.

Ao realizar o levantamento das informações a respeito do sistema, os dados em questão devem ser tratados de maneira clara para que haja precisão ao descrever os requisitos. Quanto maior for a clareza nos requisitos, maior será a efetividade da equipe responsável por implementar o sistema. Os requisitos de sistema podem ser identificados como: Requisitos funcionais e Requisitos não funcionais. Na figura 6, é demonstrado como ocorreu o processo de elaboração do documento de requisitos de sistema. O papel do cliente foi fundamental para a execução desta etapa do projeto.

Figura 6 - Processo de requisitos de sistema



Fonte: Próprio autor (2019)

4.2.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais possuem a finalidade de descrever o que o sistema deve fazer ou como o sistema deverá se comportar. Nessa etapa, são fornecidas informações com um nível maior de detalhes, a respeito das funções, entradas e saídas esperadas e possíveis exceções.

> *"Os requisitos funcionais de um sistema descrevem o que ele deve fazer. Eles dependem do tipo de software a ser desenvolvido, de quem são seus possíveis usuários e da abordagem geral adotada pela organização ao escrever os requisitos".*(SOMMERVILLE, 2011)

Esta subseção apresenta os requisitos funcionais idealizados para o produto e a prioridade de cada requisito em questão. A prioridade de cada requisito está definida como: Alta (indispensável para o produto), Média (importante para o produto) e Baixa (desejável para o produto).

O quadro 1 apresenta os requisitos funcionais do projeto, nos quais serão baseados o código de identificação, título, descrição e prioridade de cada requisito.

Quadro 1: Requisitos funcionais do sistema

Código	Título	Descrição	Prioridade
RF01	Consultar questionário	O sistema deve permitir que o usuário utilize o questionário de sistema de apoio à decisão.	Alta
RF02	Consultar arquiteturas	O sistema deverá disponibilizar informações sobre arquiteturas de *software.*	Média
RF03	Tela de boas vindas	O sistema deve possuir uma exibição de boas vindas ao usuário.	Baixa

Fonte: Próprio autor(2019)

4.2.2 REQUISITOS NÃO FUNCIONAIS

Possuindo um nome bem sugestivo, os requisitos não funcionais, são particularidades de um sistema que não necessitam estar diretamente ligados ou relacionados com as especificações ou serviços oferecidos aos usuários finais.

Esses requisitos estão caracterizados por representar em um *software* propriedades como a linguagem de programação a ser utilizada, o desempenho e esquema de cores.

Diferente dos requisitos funcionais, que representam o que o sistema fará, os requisitos não funcionais irão determinar como o sistema fará as ações planejadas, atribuindo-se às qualidades do sistema.

Os requisitos não funcionais podem ir surgindo ao longo da construção de um sistema, mediante as necessidades dos usuários, adequando-se às restrições de orçamentos, interoperabilidade com outros sistemas e políticas organizacionais. Com isso podem haver problemas no decorrer do projeto, pois os requisitos não funcionais, podem conflitar com os requisitos funcionais(SOMMERVILLE, 2011).

Nesta seção, são descritos os requisitos não funcionais pensados para a aplicação, abrangendo questões comportamentais do produto, representados na tabela 2, com código de identificação, título e descrição.

Quadro 2: Requisitos não funcionais do sistema

Código	Título	Descrição
RNF01	Linguagem Kotlin	O sistema deve ser construído utilizando a linguagem de programação kotlin.
RNF02	Plataforma android	O sistema deve ser construído para que seja utilizado na plataforma android.
RNF03	Desempenho	O sistema deve rodar sem que haja travamentos ou fechamentos repentinos.
RNF04	Segurança	O sistema deverá guardar os dados dos usuários de modo que os mesmos sejam criptografados.
RNF05	Facilidade de uso	O sistema deve possuir facilidade e interatividade em seu uso.

Fonte: Próprio autor(2019)

4.4 ASPECTOS VISUAIS

Nesta seção, será demonstrado como o aplicativo irá se comportar, exibindo o fluxo de telas presente na aplicação, assim como o detalhamento das exibições propostas para o projeto sob uma perspectiva de um maior critério técnico.

4.4.1 *Storyboard Mobile*

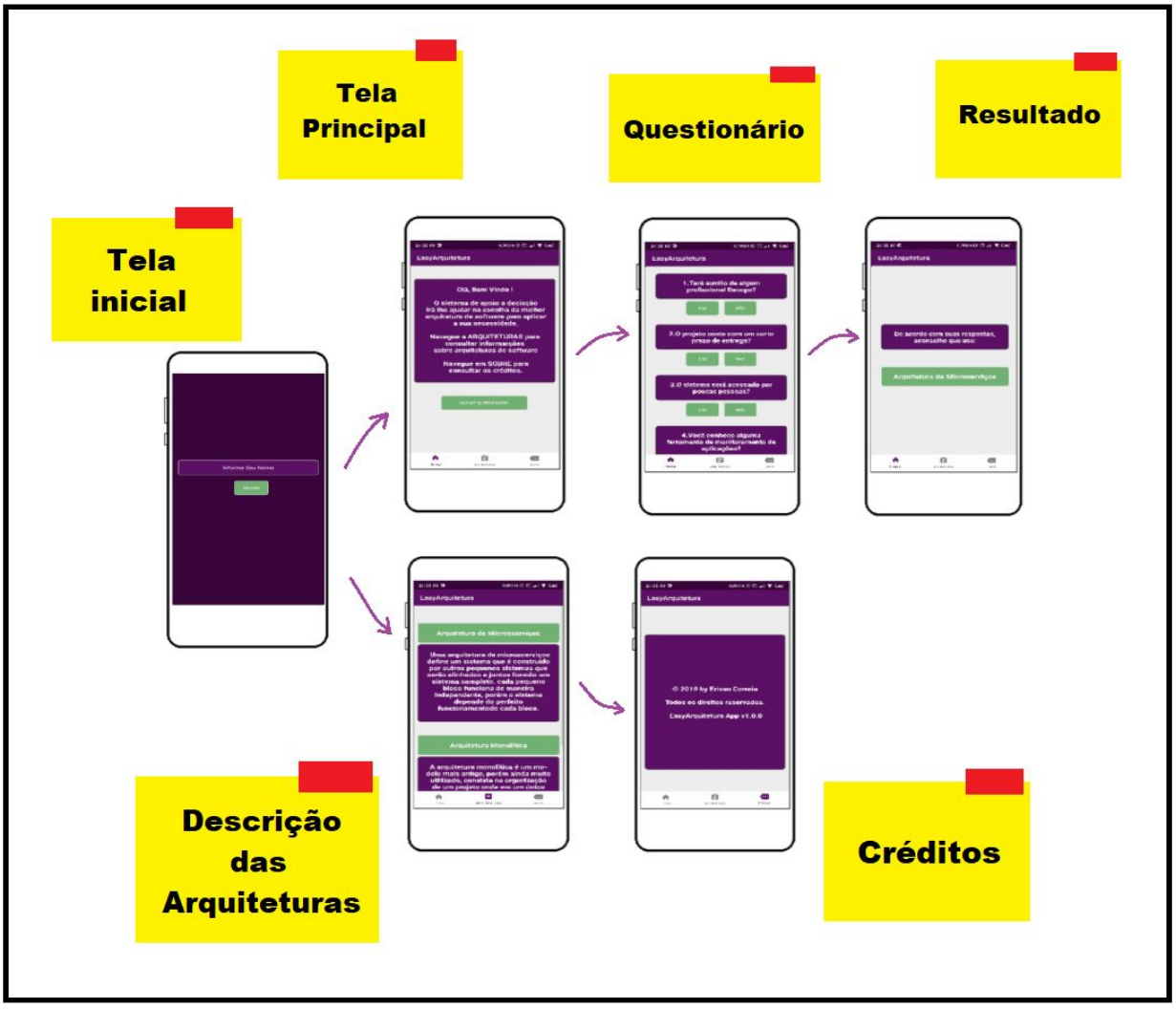
O *storyboard* é uma sequência de representações, geralmente utilizada com desenhos, sendo um esboço gráfico do projeto que representa cenas, acontecimentos e comportamentos do *software*. A intenção de utilizar a técnica de *storyboard* é demonstrar previamente a ideia desenvolvida (ROUSSEAU, 2013).

O uso do *storyboard* permitiu um grande auxílio no desenvolvimento da ideia, como uma espécie de guia de planejamento, fornecendo orientações no processo de desenvolvimento, viabilizando também uma representação global do que foi planejado.

Com esta representação, é possível idealizar, por exemplo, como será a tela inicial para acesso do usuário e como será a tela que exibirá o resultado resultante das perguntas respondidas.

A figura 7 exibe o resultado da elaboração do *storyboard* criado para dar continuidade ao projeto. É possível observar na imagem as telas principais que devem ser criadas no projeto, o fluxo entre elas e detalhes sobre o *design* do sistema.

Figura 7 - Storyboard mobile



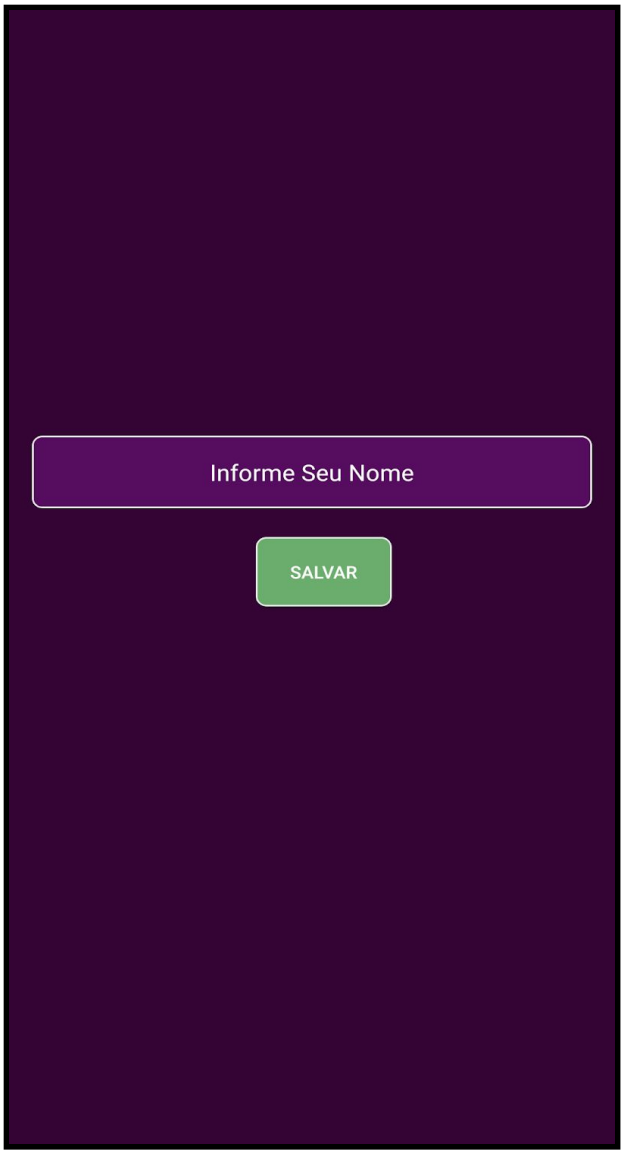
Fonte: Próprio autor (2019)

4.4.2 Tela inicial

A tela inicial do aplicativo é a base interativa com o usuário, nela deve ser informado um nome de acesso por meio de um campo de texto botão de confirmação para continuar as demais funcionalidades presentes no sistema.

Foi construída com um componente independente para representar a interface, esse componente é uma *Activity*. A seguir, na figura 8 é exibido a tela inicial.

Figura 8 - Tela inicial



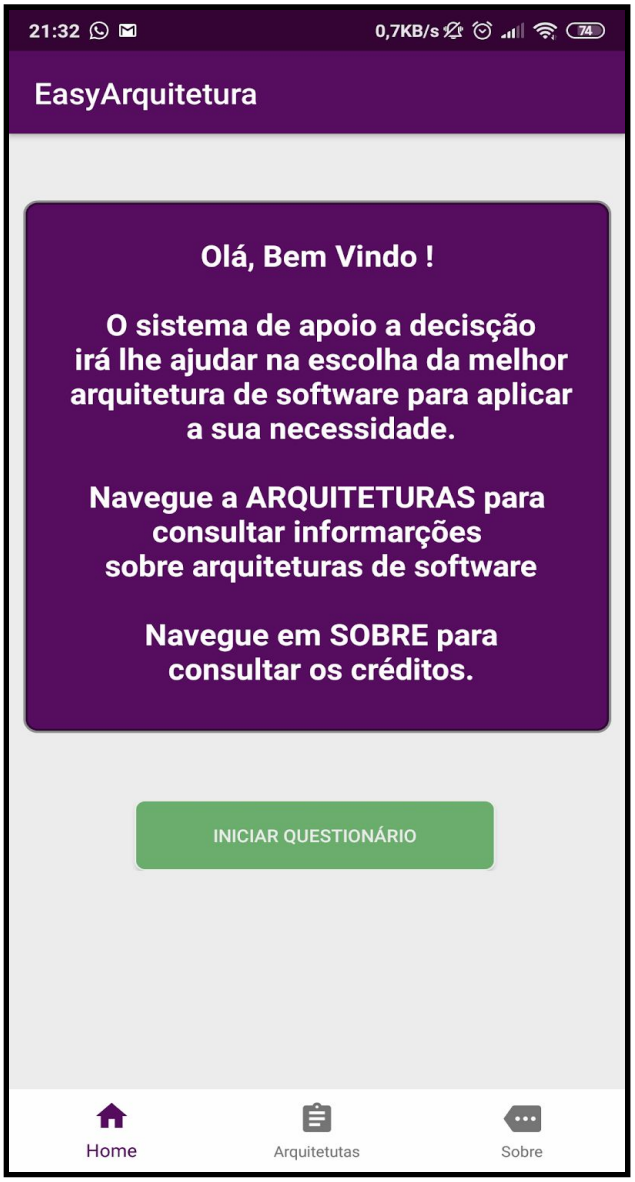
Fonte: Próprio autor (2019)

4.4.3 Tela principal

Esta tela representa a visualização de boas vindas ao sistema, contendo informações pertinentes a respeito da aplicação e uma breve explicação das funcionalidades encontradas na mesma, possuindo também o botão que leva até a funcionalidade de sistema de apoio à decisão.

A construção dessa tela se deu por meio da implementação de um componente Android conhecido como *Fragment. Fragments,* são componentes android parecidos com *Activities,* porém seu uso ocorre em conjunto com uma ou mais *Fragments* sendo lançadas dentro de uma *Activity*. A figura 9 representa a página inicial da aplicação.

Figura 9 - Tela principal



Fonte: Próprio autor (2019)

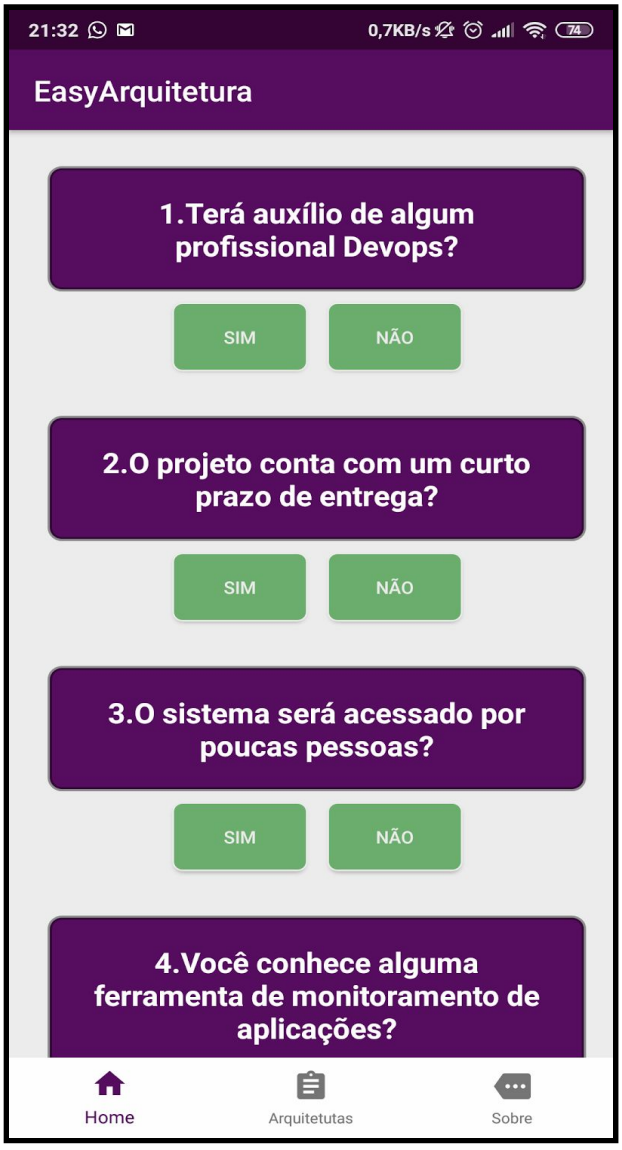
4.4.4 Questionário

Após ser apresentado a *Page Home*, é possível ser redirecionado a *Fragment* que representa a tela de questionário do sistema de apoio à decisão, como mostra a figura 10. Na figura, encontra-se a lista de questões demonstradas em detalhes no Apêndice A, para avaliar a necessidade do usuário.

Tais perguntas foram definidas por meio de conversas com especialistas, que responderam a um questionário inicial, onde cada especialista aceitou assinar um termo de consentimento para que suas respostas pudessem ser aplicadas no projeto. No questionário para o levantamento das regras, os especialistas comentaram sobre os principais pontos para a definição de uma arquitetura de *software*, e demonstrações com a finalidade de exemplificar a usabilidade de cada arquitetura.

As questões foram definidas, tendenciosamente para a avaliação do processo de respostas, nelas estão presentes algumas características de arquitetura monolítica ou de microsserviços.

Figura 10 - Questionário



Fonte: Próprio autor (2019)

4.4.5 Tela de resultado

Após a finalização do questionário, é exibido uma tela, representada pela figura 11, com o resultado obtido a partir das respostas passadas pelo usuário, esta exibição foi definida a partir de uma *Fragment,* cujo acesso só poderá ser alcançado após responder o questionário.

Para a exposição de uma resposta, o sistema avalia a pontuação gerada para cada pergunta respondida, sendo incrementado um valor, partindo do zero, para cada arquitetura. Como as perguntas foram planejadas para serem tendenciosas, a cada questão referente a arquitetura de microsserviço é incrementado um valor para a arquitetura, da mesma forma acontece com as questões que tendem representar a arquitetura monolítica, ao final do questionário o sistema identifica a arquitetura que possui a maior pontuação.

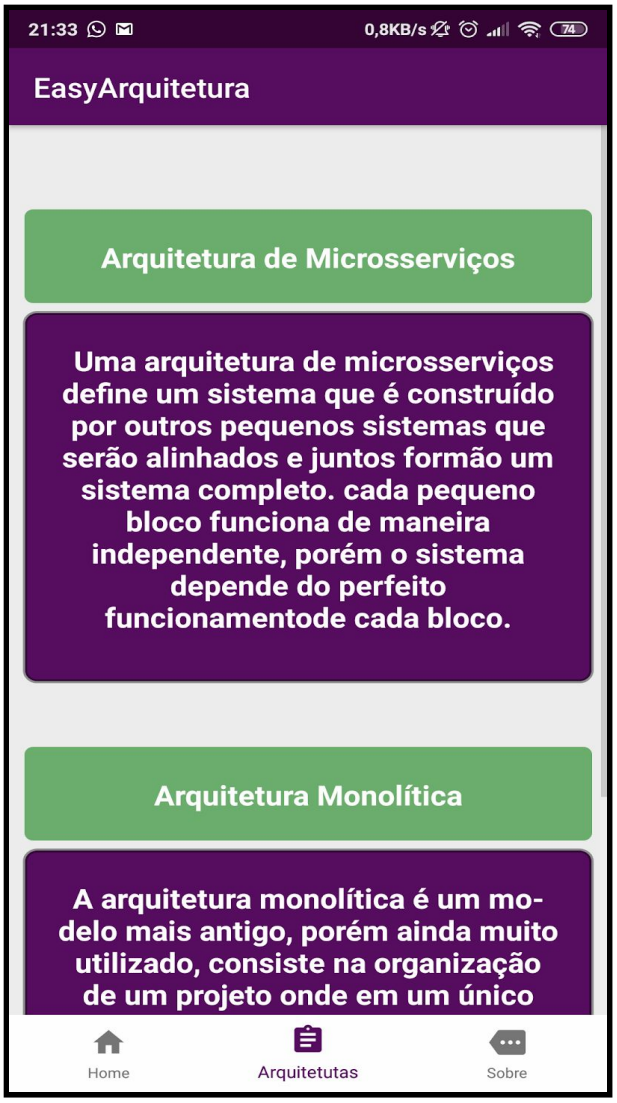
Figura 11 -Tela de Resultado

Fonte: Próprio autor (2019)

4.4.6 Tela de arquiteturas

O EasyArquitetura conta também com uma funcionalidade que descreve as arquiteturas de software, para um maior esclarecimento ao usuário que busca agregar conhecimento ao utilizar-se do serviço da plataforma. Na figura 12 é demonstrado a tela descritiva das arquiteturas de software.

Figura 12 - Tela de arquiteturas

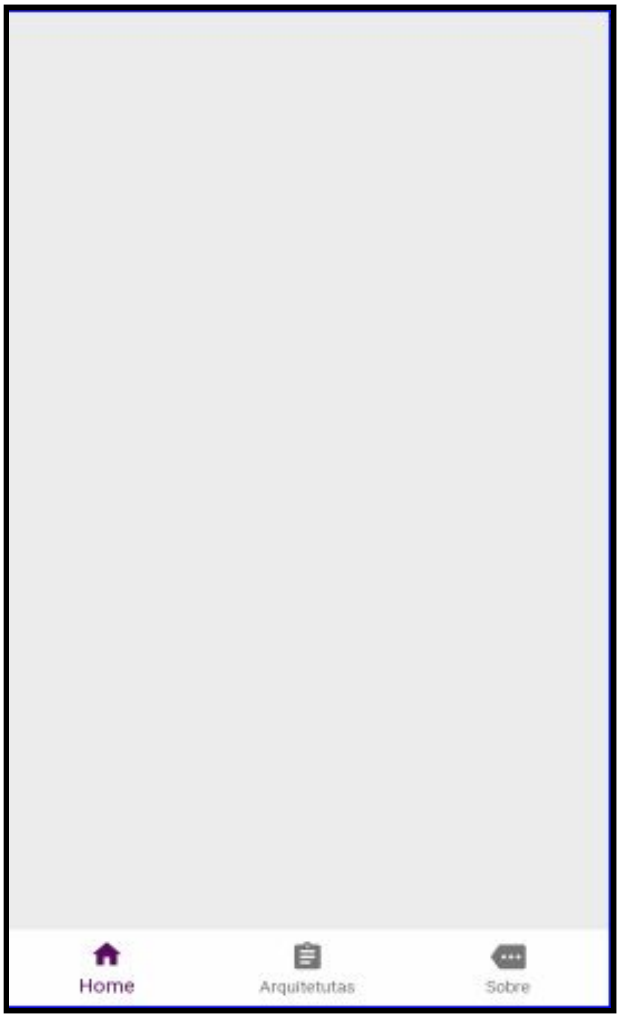


Fonte: Próprio autor (2019)

4.4.7 Menu

O canto inferior do aplicativo conta com botões de navegação, como pode ser observado na figura 13, para que o usuário possua um modo de transição entre as funcionalidades anteriormente descritas.

Figura 13 - Menu



Fonte: Próprio autor (2019)

Por meio dos botões do menu, é possível realizar uma espécie de chamada para as demais telas e, com isso, executar as funcionalidades solicitadas.

A realização das transições de telas ocorre por meio de uma implementação representada pela figura 14.

Figura 14 - Implementação de transição no menu

```
private val navigationListener : BottomNavigationView.OnNavigationItemSelectedListener
        = BottomNavigationView.OnNavigationItemSelectedListener { item ->

    when(item.itemId){
        R.id.navigation_home -> {
            replaceFragment(HomeFragment())
            return@OnNavigationItemSelectedListener true
        }
        R.id.navigation_arquit -> {
            replaceFragment(ArquiteturaFragment())
            return@OnNavigationItemSelectedListener true
        }
        R.id.navigation_about ->{
            replaceFragment(SobreFragment())
            return@OnNavigationItemSelectedListener true
        }
    }

    ^OnNavigationItemSelectedListener false
}

private fun replaceFragment(fragment: Fragment){
    val fragmentTransition : FragmentTransaction = supportFragmentManager.beginTransaction()
    fragmentTransition.replace(R.id.fragment_container, fragment)
    fragmentTransition.commit()
}
```

Fonte: Próprio autor (2019)

Os botões do menu possuem uma identificação única e por meio dessa identificação é realizado um procedimento de comparação no momento em que o usuário clica em um deles. De acordo com o botão escolhido, a função *replaceFragment* é acionada, realizando a mudança de tela.

4.5 TESTE

A prática de teste de *software* tem como objetivo garantir que a aplicação possua qualidade nas execuções das funcionalidades. É uma boa prática analisar e criar casos de testes, para os principais fluxos de atuação no sistema.

Para o processo de qualidade adotado no projeto, foram realizados testes manuais no aplicativo, garantindo a integridade do sistema. Na tabela 3, estão listados e detalhados os casos de testes da aplicação.

Quadro 3: Casos de teste

CT01 Tentar entrar no sistema passando nome válido	
Objetivo do teste	Garantir que o usuário poderá entrar no sistema após informar o nome.
Procedimento	Ao iniciar a aplicação, o usuário deverá informar um nome e em seguida clicar no botão SALVAR.
Resultado esperado	O sistema deverá redirecionar o usuário a tela principal.
CT02 Tentar entrar no sistema não passando nome	
Objetivo do teste	Garantir que o sistema não permitirá que o usuário entre no sistema sem passar um nome.
Procedimento	Ao iniciar a aplicação, o usuário deverá clicar no botão SALVAR sem informar nome.
Resultado esperado	O sistema não deverá redirecionar o usuário a tela principal e alertar a seguinte mensagem: "POR FAVOR INFORME UM NOME".
CT03 Tentar concluir questionário respondendo todas as questões	
Objetivo do teste	Garantir que o sistema irá redirecionar o usuário a tela de resposta.
Procedimento	Iniciar o questionário e responder todas as perguntas em seguida clicar em CONFIRMAR.
Resultado esperado	O sistema deverá redirecionar o usuário a tela de resposta.
CT04 Tentar concluir questionário não respondendo todas as questões	
Objetivo do teste	Garantir que o sistema não irá redirecionar o usuário a tela de resposta.
Procedimento	O usuário deverá responder algumas questões ou nenhuma questão e em seguida clicar em CONFIRMAR.
Resultado esperado	O botão CONFIRMAR deve permanecer travado sem ação, e o sistema não deverá redirecionar para a tela de resultado.

Fonte: Próprio autor (2019)

5 CONCLUSÃO

Para a produção do projeto, houve uma notória ampliação do conhecimento sobre o tema trabalhado, referenciando assuntos presentes na realidade profissional. O uso de técnicas e metodologias ágeis no processo de desenvolvimento, resultou em uma grande produtividade, assim como a escolha desafiadora da linguagem de programação para a construção da aplicação, que serviu perfeitamente para a necessidade exposta.

O projeto foi guiado pelos conhecimentos adquiridos ao decorrer do curso, zelando pelas boas práticas da análise e tratamento das informações, respeitando cada processo percorrido para conclusão do trabalho.

Este trabalho teve como objetivo o desenvolvimento de um sistema de apoio à decisão na escolha de uma arquitetura de *software* a ser utilizada em um projeto, por meio de um questionário e explicações diretas sobre arquiteturas de software.

O sistema em questão atende a profissionais e estudantes que procuram aprender ou firmar conhecimentos sobre a arquitetura de *software*, contribuindo de forma positiva na formação e melhoria técnica de quem utiliza da ferramenta.

Ao final os especialistas que contribuíram na elaboração do questionário mostraram-se animados com a ideia e com a produção da ferramenta, pois o tema abordado pelo aplicativo resulta em um esclarecimento agrupado e de fácil interação.

A primeira versão do aplicativo foi concluída, porém, existem melhorias a serem implementadas, mediante novas conversas com arquitetos de *software*, adição de mais arquiteturas e criação de uma logo que represente a ferramenta.

6 REFERÊNCIAS

ENGHOLM, Hélio. **Engenharia de Software na Prática.** 1. ed. São Paulo: Novatec, 2010.

LAURINDO, F. J. B. **Tecnologia da informação**: Planejamento e gestão de estratégias. 1. ed. Recife: Atlas, 2002.

COULOURIS, G. *et al*. **Sistemas Distribuídos**: Conceitos e Projeto. 1. ed. Porto Alegre: Bookman, 2002.

ZIMMERER, Thomas. **Essentials of small business management.** 1. ed. São Paulo: Macmillan, 1994.

PORTAL ABES. **Investimentos em TI no Brasil aumentam 4,5% em 2017**. Disponível em: <http://www.abessoftware.com.br>. Acesso em: 1 set. 2019.

LEMOS, C. **O que é arquitetura**: subtítulo do livro. 1. ed. São Paulo: Brasiliense, 1980.

SOMMERVILLE, Ian. **Engenharia De Software**: subtítulo do livro. 9. ed. São Paulo: Pearson, 2011.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. 3. ed. São Paulo: Addison-Wesley Professional, 2012.

NEWMAN, Sam; **Building Microservices**. 3. ed. São Paulo: O'Reilly Media, 2015.

FOWLER, Susan J.; **Microsserviços Prontos Para a Produção**: Construindo Sistemas Padronizados em uma Organização de Engenharia de Software. 1. ed. São Paulo: Novatec, 2017.

RED HAT. **O que são os microsserviços?**. Disponível em: <https://www.redhat.com>. Acesso em: 20 set. 2019.

TELLIER, Cristine; BONIN, Odair. **Java de ponta a ponta (6 anos depois)**, Revista MundoJ, Curitiba, Agosto 2011.

MONTEIRO, J. B. **GOOGLE ANDROID CRIE APLICAÇÕES PARA CELULARES E TABLETS**. São Paulo: Casa do Código, 2012.

ANDROID. **História do Android**. Disponível em: <https://www.android.com>.Acesso em: 28 set. 2019.

JETBRAINS. **IntelliJ IDEA**. Disponível em: <https://www.jetbrains.com>. Acesso em: 18 out. 2019.

NETBEANS. **NetBeans IDEA**. Disponível em: <https://netbeans.org>. Acesso em: 18 out. 2019.

ECLIPSE FOUNDATION. **Eclipse IDEA**. Disponível em: <https://www.eclipse.org>. Acesso em: 18 out. 2019.

KOTLIN PROGRAMMING LANGUAGE. **Try Kotlin**. Disponível em: https://kotlinlang.org. Acesso em: 21 out. 2019.

JEMEROV, Dmitry; ISAKOVA, Svetlana. **Kotlin em Ação.** 1. ed. São Paulo: Novatec, 2017.

DATE, C. J. **Introdução a sistemas de bancos de dados.** São Paulo: GEN LTC, 2004.

ELMASRI, Ramez; NAVATHE, Shamkant B. **SISTEMAS DE BANCO DE DADOS**. 6. ed. São Paulo: Addison Wesley, 2011.

RESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software**: Uma Abordagem Profissional. 8. ed. São Paulo: AMGH, 2016.

COCKBURN, Alistair. **Agile Software Development**. Adisson-Wesley, 2002.

ROUSSEAU, David Harland; **Storyboarding Essentials**: SCAD Creative Essentials (How to Translate Your Story to the Screen for Film, TV, and Other Media). Watson-Guptill, 2013.

SQLITE. **What Is SQLite?**. Disponível em: <https://www.sqlite.org>. Acesso em: 20 set. 2019.

Apêndice A - Questionário EasyArquitetura

1. Ao construir o sistema você contará com o auxílio de algum profissional DevOps?

2. O projeto possui um curto prazo de entrega?

3. O sistema planejado será acessado por poucas pessoas?

4. Você conhece alguma ferramenta de monitoramento de aplicações?

5. Possui alguma experiência em construção de API?

6. A regra de negócio explorada pelo sistema possui grande complexidade?

7. Pretende futuramente expandir o sistema?

8. Pretende utilizar algum serviço em nuvem como Amazon AWS ou Microsoft Azure?

9. Uma única linguagem de programação será suficiente para resolver todas funcionalidades em questão?