# Uma Arquitetura de Referência Baseada em Plugins para Sistemas de Informação Mobile

Enoque Joseneas [*]
Instituto Federal da Bahia
Rua Emídio dos Santos, S/N, Barbalho
Salvador-Ba, Brasil
enoquejoseneas@ifba.edu.br

Sandro Andrade [†]
Instituto Federal da Bahia
Rua Emídio dos Santos, S/N, Barbalho
Salvador-Ba, Brasil
sandroandrade@ifba.edu.br

## RESUMO

O desenvolvimento de aplicativos móveis trouxe uma série de desafios para a ciência da computação. Com limitações de recursos como a bateria, armazenamento e memória, o desenvolvimento de software para dispositivos móveis impõe requisitos não-funcionais importantes a serem considerados no projeto de aplicativos. Com a popularização da internet e das redes móveis, os aplicativos móveis tornaram-se popular e projetar aplicativos de forma fácil com componentes de alto nível, baixo acoplamento e bom desempenho não é uma tarefa trivial. Este trabalho apresenta uma arquitetura de referência para o desenvolvimento de aplicativos móveis orientado a plugins no contexto de sistemas de informação, proporcionando baixo acoplamento entre os componentes, escalabilidade de *features* através de plugins, dispõe de componentes genéricos reutilizáveis, além de componentes de alto nível para recursos corriqueiros como requisições HTTP, persistência de dados e comunicação entre os componentes através de eventos.

## Keywords

Aplicativos Móveis; Arquitetura de Software; Sistemas de Informação.

## 1. INTRODUÇÃO

Os dispositivos móveis apresentam a cada dia novas oportunidades e desafios para as tecnologias de informação, tais como o acesso ubíquo, a portabilidade, a democratização do acesso à informação além de novas oportunidades de negócio [15]. Com a expansão da Internet e o grande volume de dados compartilhados nas redes sociais e aplicativos de troca de mensagens, surgiram novos paradigmas (e.g. *Big Data*, *Cloud Computing*, *NoSQL*, etc.), novas tecnologias como o *Push Notification* e também novas oportunidades de trabalho e profissões (e.g. O analista de dados, o desenvolvedor mobile, o *design UX* e etc.), além de pesquisas importantes na ciência da computação que abrange tanto hardware como software.

Os smartphones inovam a cada dia diversas áreas do conhecimento, tais como a engenharia elétrica, no projeto de baterias cada vez mais eficientes, o design, no projeto de interfaces cada vez mais intuitivas e influenciam diretamente na evolução da Internet e dos meios de comunicação como as redes móveis que expandem as áreas de cobertura para atender ao crescente número de aparelhos conectados. Os dispositivos móveis também permitem bons empreendimentos através dos aplicativos. Atualmente, o número de downloads cresce a cada dia na *App Store* e *Google Play*, demonstrando uma certa disponibilidade dos usuários de passarem cada vez mais tempo utilizando os aplicativos do que os próprios navegadores de Internet [24]. Através dos aplicativos, é possível monetizar e gerar receitas via marketing digital e desenvolver soluções para diversos segmentos, tais como o *e-commerce*, redes sociais e sistemas de informação.

O desenvolvimento de aplicativos apesar de contar com inúmeras ferramentas tais como as IDEs (Android Studio, Eclipse e Qt-Creator) e frameworks (Ionic e PhoneGap), ainda apresentam limitações, dentre elas, a falta de soluções arquiteturais de alto nível, ausência de componentes de UI flexíveis e de alto nível. No Android por exemplo, para construir uma interface gráfica utiliza-se arquivos xml incorporados através de classes java. Outra limitação encontrada no desenvolvimento mobile, é a falta de suporte facilitado para comunicação RESTful, visto que os aplicativos móveis utilizam na maioria dos casos algum *webservice*.

Este trabalho teve como objetivo o projeto, implementação e avaliação de uma arquitetura orientada a plugins e reutilizável para o desenvolvimento de sistemas de informação mobile. Dentre os benefícios desenvolvidos destaca-se uma arquitetura de plugins, que permite ao desenvolvedor implementar as funcionalidades do sistema com maior facilidade de extensão e baixo acoplamento entre os componentes que podem se comunicar através de eventos. Esta arquitetura também provê componentes de alto nível para construção de interfaces gráficas através do QML, além de componentes de alto nível para operações rotineiras como ler e salvar dados no dispositivo através de um banco de dados SQLITE, realizar requisições HTTP com suporte a autenticação básica, download e upload de arquivos e também, permite o acesso aos arquivos do sistema que são compartilhados pelo usuário (galeria de arquivos e imagens). Esta arquitetura foi projetada a fim de atender aos seguintes requisitos funcionais: Acesso a rede através de comunicação HTTP com algum serviço REST (suporte a métodos GET, POST, upload e download de arquivos), persistência de dados local via SQLITE com tabelas definidas por cada plugin, notificações de sistema (local, partindo da própria aplicação quando estiver executando), *push notification* através da API do Firebase[1], suportando o registro do token e a exibição de notificações na bandeja do sistema.

[1]Firebase é uma plataforma de desenvolvimento de aplicativos para dispositivos móveis desenvolvida pela Firebase, Inc. em 2011 e adquirida pela Google em 2014.

“”

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Para este trabalho foi utilizado como principal tecnologia o Qt, que provê um mecanismo de comunicação através de eventos, via sinais e slots e possibilita para a aplicação um meio de comunicação assíncrono entre objetos. O Qt também disponibiliza objetos de alto nível que abstrai a plataforma para operações de rede e persistência de dados. Outro recurso que o Qt provê é a instalação e a construção do arquivo executável da aplicação. O Qt também realiza o *deploy* da aplicação em um dispositivo durante a fase de desenvolvimento, agilizando o processo de testes e correção de bugs do aplicativo.

Este trabalho esta organizado como segue. A sessão 2 apresenta o referencial bibliográfico, destacando os assuntos emergentes relacionados a arquitura aqui apresentada. Em seguida, na sessão 3, será apresentado os trabalhos relacionados. As seções 4, 5 e 6 apresentam as tecnologias utilizadas e detalhes da solução desenvolvida. Por fim, na seção 7, é apresentado as perspectivas de trabalhos futuros e a conclusão deste trabalho.

# 2. REFERENCIAL BIBLIOGRÁFICO

Esta seção, apresenta as principais referências que contextualizam este trabalho. A subseção 2.1 descreve sobre Arquitetura de Software. A subseção 2.2 descreve Visão Arquitetural. A subseção 2.3 resume Sistemas de Informação. Já a subseção 2.4 apresenta Arquitetura de Referência para Sistemas de Informação. A subseção 2.5 apresenta de forma geral Arquiteturas de Aplicativos Móveis. A subseção 2.6 faz uma breve revisão sobre Projetos de Aplicativos Móveis. A subseção 2.7 faz uma breve revisão sobre tecnologias suportadas na arquitetura deste trabalho: *web services*, o estilo arquitetural *REST*, o *Push Notification* e o JSON. Para finalizar, a subseção 2.8 destaca sobre Desenvolvimento de Software Orientado a Componentes.

## 2.1 Arquitetura de Software

De acordo com a definição clássica proposta por *Shaw e Garlan* [16], arquitetura de software define o que é sistema em termos de componentes computacionais e os relacionamentos entre eles, os padrões que guiam suas composições e restrições. Arquitetura de software pode ser compreendida como uma especificação abstrata do funcionamento de um sistema e permite especificar, visualizar e documentar a estrutura e o funcionamento de um programa independente da linguagem de programação na qual ele será implementado [14].

Os softwares estão em constante evolução e sofrem mudanças periodicamente, que ocorrem por necessidade de corrigir *bugs* ou de adicionar novas funcionalidades. As mudanças ocorridas no processo de evolução de um software podem torná-lo instável e predisposto a defeitos, além de causar atraso na entrega e custos acima do estimado. Porém, um software que é projetado orientado a arquitetura, possibilita os seguintes benefícios:

1. Melhor escalabilidade;
2. Maior controle intelectual;
3. Menor impacto causado pelas mudanças;
4. Melhor atendimento aos requisitos não-funcionais;
5. Maior agilidade na manutenção do código;
6. Padronização de comunicação entre os componentes e;
7. Suporte a reuso dos componentes e maior controle dos mesmos.

O desenvolvimento de software envolve muitas partes (e.g., levantamento de requisitos, modelagem, implementação, testes, refatoração e etc.). O objetivo de um software é o que motiva a sua construção, e o que fomenta todas as partes que envolve o seu desenvolvimento é o problema que ele tenta solucionar no mundo real e parte do mérito de uma boa solução é devido ao uso de uma boa arquitetura.

Neste trabalho, arquitetura de software pode ser compreendida nas decisões de implementação, nas restrições impostas pelo uso dos recursos disponibilizados e dos componentes reutilizáveis, além dos estilos arquiteturais provenientes das APIs utilizadas, tais como, o *Event-Based*, mecanismo de comunicação orientado a eventos provido pelo Qt/QML e o *Restful* que é utilizado como *web service* suportado pela aplicação. Outro aspecto arquitetural deste trabalho é um estilo de desenvolvimento orientado a plugins que constituem os componentes específicos de cada projeto ou aplicação baseada nesta arquitetura. Os plugins formarão os principais recursos do sistema e são independentes entre si. O principal destaque em uma arquitetura de plugins é o baixo acoplamento entre as funcionalidades do sistema.

## 2.2 Visão Arquitetural

A arquitetura de um software pode ser representada de vários pontos de vista, que podem ser combinados para criar uma visão holística do sistema [2]. As visões arquiteturais são diferentes formas de observar a arquitetura de um software, cada qual ressaltando aspectos específicos e relevantes conforme o papel da pessoa que está definindo a arquitetura e a etapa do processo de desenvolvimento em que ela se encontra [22]. Os requisitos funcionais implementados nesta arquitetura serão apresentados em um tópico seguinte em visões arquiteturais através de imagens e diagramas da UML. O objetivo das visões é facilitar a compreensão das partes que compõe esta arquitetura.

## 2.3 Sistemas de Informação

Um sistema de informação pode ser definido como um conjunto de componentes inter-relacionados trabalhando juntos para coletar, recuperar, processar, armazenar e distribuir informações com a finalidade de facilitar o planejamento, o controle, a coordenação, a análise e o processo decisório em organizações [13].

## 2.4 Arquitetura de Referência para Sistemas de Informação

Uma arquitetura de referência consiste em uma forma de apresentar um padrão genérico para um projeto [26]. Com base nessa arquitetura, o desenvolvedor projeta, desenvolve e configura uma aplicação prototipando-a por meio de componentes reutilizáveis [26].

Para compor uma arquitetura de referência é necessário apresentar os tipos dos elementos envolvidos, como eles interagem e o mapeamento das funcionalidades para estes elementos [10]. De maneira geral, uma arquitetura de referência deve abordar os requisitos para o desenvolvimento de soluções, guiado pelo modelo de referência e por um estilo arquitetural de forma a atender as necessidades do projeto [4]. A concepção de uma arquitetura de referência pode ser entendida neste trabalho como uma forma de disponibilizar um padrão genérico para o desenvolvimento de novos aplicativos no contexto de sistemas de informação.

O domínio de aplicações móveis engloba vários requisitos e restrições que variam entre limitações de hardware tais como energia limitada, baixo poder de processamento e limitação de recursos como armazenamento e realizar comunicação remota quando o dispositivo está em rede móvel. Ao utilizar uma arquitetura de

referência, é possível obter recursos implementados para funcionalidades corriqueiras no contexto da aplicação, como persistir dados, obter informações de um *web service*, exibir uma notificação para o usuário e etc.

## 2.5 Arquiteturas de Aplicativos Mobile

Arquitetura para aplicações móveis abrange quatro camadas: Interação Humana-Computador (*IHC*), Aplicação Móvel, *Middleware* e *Enterprise Backend* [20]. Neste trabalho, a arquitetura foi concentrada apenas nas camadas de interação, aplicação e *Middleware*.

### *2.5.1 Camada de Interação Humana-Computador*

A camada de Interação Humana-Computador (mais conhecida como interface de usuário, ou simplesmente UI) define os elementos de interação entre o usuário e os recursos do aplicativo. De forma abstrata, a camada de interface do usuário descreve o tipo de mídia suportada pelo aplicativo (por exemplo, texto, gráficos, imagens, vídeo ou som), os tipos de mecanismos de entrada (por exemplo, teclado alfa-numérico, ponteiros de caneta ou toques na tela) e os tipos de mecanismos de saída (por exemplo, uma notificação na bandeja do sistema, a tela, os alto-falantes ou algum tipo de *feedback* como vibrar o dispositivo) [20]. Um exemplo de um componente desta camada é o objeto *Image* do QML que corresponde ao carregamento e exibição de uma imagem na tela, além de botões, campos de texto e elementos que suportam cliques.

### *2.5.2 Camada de aplicação*

A camada de aplicação corresponde ao processamento de ações e eventos provenientes da camada de interação, como por exemplo, captando eventos de toque em objetos visuais e realizando processamento em segundo plano como trocar a página atualmente vista pelo usuário, utilizando algum parâmetro lido em um objeto *JSON*. Esta camada, corresponde a componentes não visuais e interagem diretamente com a camada de *middleware*. Objetos da camada de aplicação podem por exemplo, gerenciar e controlar a criação de outros componentes tais como, o objeto *StackView* do QML, que instancia páginas dinamicamente a partir de cliques em um menu de opções.

### *2.5.3 Camada de middleware*

A camada de *middleware* intercala entre a camada de aplicação com a camada de *backend*. O objetivo dessa camada é fornecer de forma abstrata e genérica um meio de comunicação entre o modelo de dados da aplicação com a camada de *backend* [20]. Ela é também responsável por interagir com o meio de comunicação disponível no dispositivo abstraindo para a camada de aplicação qual foi a interface de hardware utilizada. Objetos da camada de *middleware* podem ser executados de forma assíncrona para que não bloqueiem os eventos da tela, enquanto aguardam um feedback da camada de *backend* para permitir melhor desempenho e usabilidade da aplicação. Exemplos de componentes dessa camada são objetos que realizam acesso a rede através de requisições HTTP.

### *2.5.4 Camada de backend*

A camada *backend* consiste de uma outra aplicação que responde pelas requisições do aplicativo através de uma rede via protocolo *HTTP*. Esta camada está associada ao *web service* ou serviço REST. O *web service* pode atender a diferentes requisições e dispositivos, além de abstrair para a aplicação, toda lógica de negócios referente ao armazenamento e processamento dos dados do sistema. A implementação desta camada pode ser desenvolvida sobre uma outra arquitetura, além de implementar regras de negócio inerentes ao seu funcionamento.

## 2.6 Projeto de Aplicativos Móveis

Um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. O termo temporário quer dizer que o projeto possui um ciclo de vida com início e final determinados [9]. O projeto termina quando seus objetivos forem alcançados ou quando existirem motivos para não continuá-lo [9]. Um aplicativo móvel ou aplicação móvel ou simplesmente *app*, é um sistema desenvolvido para ser instalado e executado em um dispositivo eletrônico portátil, como tablets e smartphones [7].

Um aplicativo móvel pode ser baixado diretamente no aparelho eletrônico, desde que o dispositivo possua conexão com a Internet. O mercado de dispositivos móveis é ramificado por diferentes fabricantes, o que inclui uma variação de plataformas de desenvolvimento, sistemas operacionais, versões do SO e configuração variada de hardware. Na construção de um aplicativo para dispositivo móvel, a implementação é um ponto muito importante, pois, além de representar a parte concreta dos requisitos funcionais do aplicativo também refletem diretamente nos requisitos não funcionais e consequentemente na qualidade do software.

O sucesso de aplicativos para dispositivos móveis vai além das medidas de desempenho, portabilidade e usabilidade tradicionais [12]. Os aplicativos devem estar em conformidade com a personalidade, preferências, objetivos, experiências e conhecimento de seus usuários [25]. Além disso, o contexto físico, social e virtual onde ocorrem as interações deve, sempre que possível, ser levado em consideração [17].

Torna-se evidente que são muitos requisitos a serem considerados em um projeto de aplicativo móvel. O esforço dedicado para atender a todos os requisitos pode tornar o projeto enfadonho, além de exigir tempo e mão de obra. O processo de desenvolvimento pode ser otimizado através de ferramentas como *frameworks* ou uma arquitetura de componentes reutilizáveis, a fim de agilizar e auxiliar o processo de desenvolvimento.

## 2.7 Desenvolvimento de Software Orientado a Componentes

O desenvolvimento de software Orientado a componentes é um paradigma da engenharia de software caracterizado pela composição de partes já existentes, ou desenvolvidas independentemente e que são integradas para atingir um objetivo final [6]. Construir novas soluções pela combinação de componentes desenvolvidos aumenta a qualidade e dá suporte ao rápido desenvolvimento, levando à diminuição do tempo de entrega do produto final ao mercado [6]. Os sistemas definidos através da composição de componentes permitem que sejam adicionadas, removidas e substituídas partes do sistema sem a necessidade de sua completa substituição. Com isso, o desenvolvimento baseado em componentes auxilia na manutenção do software, por permitir que o sistema seja atualizado através da integração de novos componentes ou atualização dos objetos já existentes [23].

O reuso de componentes é um recurso extra da arquitetura apresentada neste trabalho, pois dispõe de trinta componentes (visuais e não visuais) reutilizáveis para auxiliar no desenvolvimento de novos aplicativos, seguindo as restrições desta arquitetura que é dedicada a sistemas de informação. Estes componentes são arquivos QML genéricos que podem atender a diferentes customizações através das propriedades disponibilizadas pelos seus elementos internos, que permitem definir ou alterar os valores pre-definidos. Os benefícios da componentização estão ligados a manutenibilidade, reúso, composição, extensibilidade, integração e escalabilidade [5].

## 2.8 Web Services, RESTful, Push Notification e JSON

Os *web services* constituem uma tecnologia emergente da Arquitetura Orientada a Serviços (SOA) [21]. Com a expansão da internet e a necessidade de integração entre aplicações web, tornou-se necessário a centralização de informações para serem acessados por diferentes clientes. Para esse propósito, foi criada a tecnologia de *web services* [19]. Uma característica fundamental dos *web services*, diz respeito à possibilidade de utilização de diferentes formas de transmissão de dados pela rede e o atendimento a diferentes clientes e dispositivos. Logo, a arquitetura de *web services* pode trabalhar com protocolos HTTP, SMTP, FTP, RMI ou protocolos de mensagem proprietários.

o RESTFul é um estilo arquitetural para a construção de sistemas distribuídos [8]. O elemento fundamental da arquitetura RESTful é o *resource* ou recurso. Um recurso pode ser uma página web contendo um documento estruturado, uma imagem ou até mesmo um vídeo. Para localizar os recursos envolvidos em uma interação entre os componentes da arquitetura RESTful é utilizado o chamado identificador de recurso ou *URI*. Com isso, um recurso pode ser representado através de diferentes formatos e o mais comum e utilizado é o *JSON*. O termo REST, é empregado em serviços RESTful que não implementam todos os princípios da especificação do estilo arquitetural definido por *Fielding* [8]. Para que todos os princípios deste estilo sejam respeitados, um conjunto de restrições deve ser seguido, os quais não serão abordados neste trabalho. Para facilitar a compreensão geral de um serviço REST, a imagem à seguir foi adicionada para representar um modelo de comunicação entre um cliente ou dispositivo e um serviço RESTful.

*Push Notification* é descrito por Acer et al. [3] como mensagens pequenas, usadas por aplicações de celular para informar aos usuários sobre novos eventos e atualizações. As notificações na maioria dos casos, estão associadas aos aplicativos instalados no dispositivo. O termo *push* indica que a mensagem parte do servidor para o dispositivo. Os principais provedores de notificações via *push* são o *Apple Push Notification Server* (APN) e o *Firebase* antigo *Google Cloud Messaging*. A imagem à seguir, apresenta o modelo de comunicação que ocorre no envio de um *Push Notification*.

O JSON[2] (*JavaScript Object Notation*) é um conjunto de chaves e valores, que podem ser interpretados por qualquer linguagem. Além de ser um formato de troca de dados largamente utilizado em serviços REST, é fácil de ser entendido e escrito pelos programadores. Estas propriedades fazem do JSON um objeto ideal para o intercâmbio de dados em aplicações web tal como o XML [11].

# 3. TRABALHOS RELACIONADOS

Nesta seção, serão detalhados os trabalhos relacionados com este projeto. O detalhamento será feito com uma descrição geral do trabalho desenvolvido e como ele se relaciona com este projeto. Será considerado também os pontos fracos e fortes identificados exibidos em uma simples tabela.

**Uma Arquitetura de Referência para o Desenvolvimento de Sistemas Colaborativos Móveis Baseados em Componentes**

A arquitetura de referência proposta, denominada CReAMA – *Component-Based Reference Architecture for Collaborative Mobile Applications*, teve como principal objetivo orientar o desenvolvimento de sistemas colaborativos móveis baseados em componentes para a plataforma Android. Sistemas desenvolvidos de acordo com essa arquitetura, devem dar suporte ao desenvolvimento de componentes e à criação de aplicações colaborativas por meio da composição desses componentes. As aplicações e componentes são desenvolvidos para plataformas móveis, facilitando o uso de recursos inerentes a essas plataformas, tais como informações de sensores embarcados. Com base na arquitetura de referência, o desenvolvedor poderá ser guiado para criar componentes e compor novas aplicações seguindo os padrões estabelecidos. Por exemplo, será possível construir *toolkits* que forneçam componentes para um domínio específico. É importante ressaltar que a arquitetura foi definida considerando-se: aspectos da plataforma móvel, de sistemas colaborativos e da própria orientação a componentes. Com relação à plataforma móvel, optou-se por uma plataforma específica, visando-se a definição de uma arquitetura otimizada para as características da respectiva plataforma. A arquitetura proposta dará suporte ao desenvolvimento de novos sistemas baseados em componentes, considerando também aspectos relativos à comunicação com a Web.

O trabalho proposto por Maison Melotti se relaciona com este trabalho pelo fato de terem objetivos semelhantes, que é propor uma arquitetura para facilitar o desenvolvimento de aplicativos móveis, permitindo o reuso facilitado de componentes. Apesar de estarem focados em domínio específico, os trabalhos se relacionam no atendimento de dois requisitos funcionais, que são eles cache de dados, notificações do sistema (local e *push notification*), acesso a rede (requisições HTTP) e o provimento de componentes reutilizáveis pelas aplicações baseadas na arquitetura proposta.

| Recurso | Esta arquitetura | Arquitetura de Melotti |
|---|---|---|
| Provê suporte multiplataforma Desktop e Mobile (Android e iOS) | Sim | Não |
| Provê recursos extensíveis através de plugins | Sim | Não |
| Provê APIs de alto nível (Rede, banco de dados, UI) | Sim | Sim |
| Provê componentes reutilizáveis | Sim | Sim |

Tabela 1: Comparação entre este trabalho e o de Maison Melotti

**MoCA: Uma Arquitetura para o Desenvolvimento de Aplicações Sensíveis ao Contexto para Dispositivos Móveis**

MoCA (*Mobile Collaboration Architecture*) é uma arquitetura que oferece recursos para o desenvolvimento de aplicações distribuídas sensíveis ao contexto que envolvem usuários móveis. Esses recursos incluem um serviço para a coleta, armazenamento e distribuição de informações de contexto e um serviço de inferência de localização de dispositivos móveis. Além disso, a arquitetura provê APIs para o desenvolvimento de aplicações que interagem com estes servicos como consumidores de informações de contexto. Os servicos providos pela MoCA livram o programador da obrigação de implementar serviços específicos para a coleta e tratamento de contexto. O conjunto de APIs oferecidas pela MoCA para desenvolvimento de aplicações compreende tres grupos: as APIs de comunicação, que fornecem interfaces de comunicação sícrona e assícrona (baseada em eventos); as APIs principais que fornecem interfaces de comunicação com os serviços básicos da arquitetura; e as APIs opcionais que facilitam o desenvolvimento de aplicações baseadas na arquitetura cliente-servidor.

A relação deste trabalho com a arquitetura porposta em MoCA pode ser entendida pelo uso dos estilos arquiteturais *Event Based* e *Cliente Servidor*. Além provê APIs de alto nível para operações de rede (HTTP), persistência de dados no dispositivo e notificação do sistema. No entando, MoCa foi construído para trabalhar com um servidor próprio, atendendo requisições específicas de seu domínio, enquanto que esta arquitetura propõe um modelo de comunicação cliente servidor através de serviços RESTFul. A tabela a seguir comparar os principais recursos desta arquitetura com o modelo

[2]https://json.org

proposta em MoCA.

| Recurso | Esta arquitetura | Arquitetura MoCA |
|---|---|---|
| Provê suporte multiplataforma Desktop e Mobile (Android e iOS) | Sim | Não |
| Provê recursos extensíveis através de plugins | Sim | Não |
| Provê APIs de alto nível (Rede, banco de dados, UI) | Sim | Sim |
| Provê componentes reutilizáveis | Sim | Não |

Tabela 2: Comparação entre este trabalho e o de Maison Melotti

# 4. PROJETO DA ARQUITETURA

A arquitetura proposta neste trabalho é baseada em camadas e utiliza os estilos arquiteturais *Client-Server* e *Event-Based*. O modelo em camadas possibilita manter a organização, a separação de conceitos e responsabilidades dos recursos da arquitetura, viabilizando a integração e a comunicação entre os componentes através de conectores que podem ser definidos dinamicamente. No modelo em camadas, a conexão entre os componentes pode ser realizado tanto por eventos como por objetos compartilhados ou, através de leitura e escrita em um arquivo ou em uma base de dados. Porém, nesta arquitetura, foi utilizado somente eventos para comunicação entre componentes. A escolha de eventos como principal conector entre os objetos foi feita principalmente por proporcionar comunicação assíncrona entre emissor e ouvinte e pelo fato de não acoplar os componentes, garantindo maior independência entre os objetos além de permitir que os plugins possam passar dados para outros objetos ou até mesmo outros plugins. Outro motivo da escolha de eventos, é o fato do Qt provê nativamente um mecanismo de comunicação por eventos via sinais e slots.

## 4.1 Processos de desenvolvimento

Esta arquitetura foi desenvolvida sob uma metodologia ágil com destaque para uma programação extrema e teste contínuo. A arquitetura recebeu alterações durante 10 meses e a primeira etapa de desenvolvimento introduziu o suporte aos plugins. O primeiro desafio foi desacoplar os plugins do arquivo QRC[3] e permitir que a aplicação carregasse-os dinamicamente. Também nesta primeira etapa, foi implementado alguns recursos associados aos plugins, como controle de cache dos arquivos QML, ordenação e *parsing* das páginas (definido pelos plugins), além da criação de um componente genérico a ser extendido por todas as páginas do aplicativo. O controle de cache consiste em regenerar o cache de todos os arquivos da aplicação após uma atualização e se faz necessário para garantir o carregamento de mudanças em cada arquivo a cada release. O componente genérico foi definido como *BasePage.qml* e será detalhado posteriormente, ele foi criado para garantir o atendimento de alguns requisitos mínimos de aparência e estrutura da aplicação além de simplificar a criação de páginas. Na segunda etapa, foi implementado uma classe utilitária para que seus métodos fossem utilizados pelos plugins, oferecendo operações de baixo nível ainda não suportados pelo QML, pois nesta versão da arquitetura, ainda não é suportado a implementação de objetos c++ pelos plugins. Na terceira etapa, foi definido os layouts visuais suportados pela arquitetura e dois modelos foram implementados: O layout em pilha (faz uso do container *StackView*) e o layout em linha (faz uso do container *SwipeView*). O suporte ao layout deve ser definido no arquivo de configuração principal (à ser detalhado posteriormente) nomeado *config.json*. Outros componentes serão instanciados para uso em conjunto com cada layout, dentre eles o *TabBar* e o *ToolBar*. Em etapas seguintes foi desenvolvido componentes visuais reutilizados na aplicação além das APIs para requisição HTTP e persistência de dados via QSLITE.

[3]QRC - Qt Resource Collection é um arquivo XML que mapeia os arquivos que serão empacotados no aplicativo

## 4.2 Tecnologias utilizadas

As tecnologias utilizadas consiste de todos os recursos que foram necessários para o desenvolvimento deste trabalho. O Qt e o *QtCreator* foram os artefatos mais importantes, pois, forneceram os recursos e ferramentas para a construção das principais características da arquitetura. Dentre os recursos providos pelo Qt destaca-se os eventos, que permitem interligar objetos através de sinais e slots[4] ou *signal handles*, e as APIs providas em classes C++ que integram os recursos da arquitetura, tais como, persistência de dados (via *QSettings* e *QSqlDatabase*) e rede (via *QNetworkAccessManager*). O *QtCreator* é uma IDE que possui recursos integrados à um projeto Qt e foi essencial para o desenvolvimento deste trabalho. Alguns recursos do *QtCreator* destaca-se, facilidade de *build* do projeto, construção do executável do aplicativo e o *deploy* em um *smartphone*, além de facilitar a realização de testes através de um mecanismo integrado de depuração. O *QtCreator* também foi utilizado como editor de código fonte.

## 4.3 Visão Geral da Arquitetura

A visão a seguir, apresenta um diagrama de componentes da UML e exibe uma visão lógica dos principais pacotes e arquivos da arquitetura e logo a seguir, é descrito o papel e o conteúdo de cada um deles.

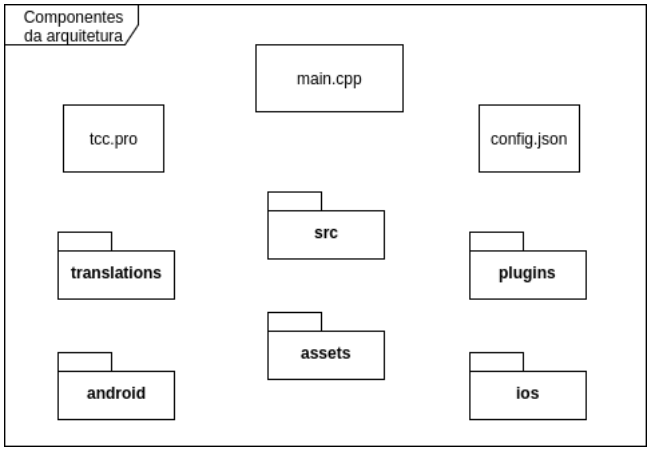


Figura 1: Pacotes principais da arquitetura

**1** *tcc.pro*: Arquivo de configuração de todo projeto Qt. Nele é definido os módulos do Qt a serem utilizados na aplicação, as classes c++ que serão compiladas e linkadas no executável, os aquivos qrc que mapeiam os componentes QML, as imagens e arquivos genéricos a serem empacotados no executável, além de módulos e arquivos de configuração para cada plataforma (linux, osx, android e ios). É neste arquivo que fica definido onde os plugins serão instalados no dispositivo.

[4]funções javascript ou métodos de uma classe c++ invocados quando o sinal o qual estão conectados for emitido, recebendo em seus parâmetros os argumentos enviado pelo sinal.

**2** *main.cpp*: Arquivo que inicializa a aplicação. Esse arquivo é responsável por instanciar as classes do Qt que exibem a janela do aplicativo e o interpretador de código QML, além de classes da camada de aplicação, configuração e utilitários. O *main.cpp* também é responsável por carregar os arquivos de tradução e registrar objetos no contexto da aplicação a serem utilizados pelos plugins.

**3** *config.json*: Arquivo de configuração principal desta arquitetura, pois define propriedades que indicarão alguns comportamentos iniciais e esolha de tipos de objetos a serem instanciados, tais como exibir os termos de uso na primeira inicialização do aplicativo (carregará o arquivo definido em *assets/eula.html* definido pelo usuário da arquitetura), se tem login ou não (caso sim, instanciará um objeto que gerencia o perfil do usuário e a página de login), se usará layout em linha e etc. Os detalhes deste arquivo será apresentado em um tópico mais adiante.

**4** *src*: Diretório de código fonte. É onde está as classes c++ e componentes QML utilizados internamente e dispostos para plugins como componentes reusáveis. Esse diretório é subdividido em outros 5 diretórios que organizam as classes c++ por tipo de API a qual são resposáveis, e são eles: **core** – contém classes do núcleo da aplicação dentre elas *App*, *PluginManager*, *Notification* e *Utils*; **database** – contém classes da API de persistência de dados da aplicação; **extras** – contém classes de customização de estilo no android; **network** que mantém as classes da API de rede da aplicação e **qml** que contém os arquivos qml sub-divididos em *private* e *public*, sendo os componentes "privado"os que são utilizados internamente pela aplicação e os componentens públicos os que são reutilizados pelos plugins.

**5** *plugins*: Diretório de plugins. Cada plugin deve obrigatóriamente estar em um sub-diretório com no mínimo um arquivo de configuração de nome *config.json* e os arquivos QML necessirários para o seu funcionamento. Os detalhes das propriedades requeridas para o carregamento de um plugin serão descritas em um tópico posterior.

**6** *translations*: Diretório contendo os arquivos de tradução. Os arquivos de tradução devem ser gerados ou atualizados antes de cada release do aplicativo. Um arquivo de resources *translations.qrc* existe neste diretório e deve ser utilizado para mapear os arquivos de idioma suportados pelo aplicativo. Cada arquivo de tradução deve ser nomeado seguindo o padrão *language_COUNTRY* com extensão *ts*, por exemplo: *pt_BR.ts*. Ao iniciar a aplicação, no arquivo *main.cpp*, será identificado o *locale* que define o idioma utilizado no dispositivo e o arquivo correspondente será carregado para que os textos visíveis sejam traduzidos para o usuário. Para gerar as traduções, deve-se utilizar o comando *lupdate *.pro* (na raíz do projeto) para criar ou atualizar o arquivo *ts* principal.

**7** *android*: Diretório contendo os arquivos de configuração do aplicativo para a plataforma android. Outros sub-diretórios guardam arquivos do *gradle* utilizados para o *build* do APK, ícones do lançador do aplicativo e classes java, além de uma versão da lib *openssl* compilada para o funcionamento de requisições HTTP.

**8** *assets*: Diretório contendo imagens e arquivos de configuração do *qtquickcontrols2* além de um arquivo html que pode ser usado para exibir os termos de uso do aplicativo quando utilizado necessário (a propriedade *showEula* seja definido para true no *config.json*). Um arquivo de *resources assets.qrc* mapeia todos os arquivos contidos neste diretório e pode ser usado para adicionar outros arquivos a serem empacotados no aplicativo.

**9** *ios*: Diretório contendo os arquivos de configuração do aplicativo para a plataforma ios. Pode conter os ícones do aplicativo, além de imagens diversas requeridas pela plataforma tais como as imagens de *splash-screen*, além do arquivo de configuração *Info.plist* que define nome, versão e recursos do sistema requerido pelo aplicativo.

## 4.4 Arquitetura de plugins

As funcionalidades de um aplicativo baseado nesta arquitetura devem ser implementadas através de plugins, atendendo as necessidades e os requisitos levantados para a aplicação a ser desenvolvida utilizando apenas qml. Os plugins são independentes entre sí e podem incluir arquivos qml, txt, html e imagens em seu diretório. Qualquer componente de um plugin pode reutilizar os componentes públicos usando a diretiva *import "qrc:/src/qml/"*. Os plugins estão desacoplados do núcleo da aplicação e serão conhecidos em tempo de execução. Ao adicionar um novo plugin no diretório *plugins*, ele será carregado no próximo *build* do projeto. Para que um plugin seja reconhecido pelo objeto gerenciador de plugins e carregado pela aplicação, é necessário obedecer a três restrições: estar em um sub-diretório dentro de *plugins*, conter um arquivo *config.json* e pelo menos um arquivo qml visual ou *listener*. Os *listeners* são componentes não visuais que observam eventos da aplicação. O arquivo *config.json* deve ser um objeto contendo as seguintes propriedades:

1. *name* (string): o nome do plugin. Deve ser preenchido para o plugin seja identificado pelo objeto gerenciador de plugins;

2. *description* (string): um texto que descreve o plugin. Deve ser definido, caso contrário o plugin não será adicionado a lista de plugins carregados pela aplicação;

3. *pages* (array): uma lista de objetos que indentifica as páginas do plugin que serão acessadas a partir do menu ativo no aplicativo. Deve ser preenchido caso a propriedade *listeners* esteja vazia;

4. *listeners* (array): uma lista de strings que indentifica os arquivos do plugin (componentes qml não visuais) que serão instanciados como observadores de eventos. O preenchimento dessa propriedade é opcional, porém pode ser preenchida mesmo que a propriedade *pages* tenha sido fornecida.

Cada objeto em *pages* poderá conter as seguintes propriedades que serão lidas pelos componentes que definem os menus durante a inicialização do aplicativo:

- *qml* (string): O nome do arquivo correspondente a página. Se esse valor não for definido, a página não será carregada.

- *title* (string): O título correspondente a página a ser exibido no menu. Esse valor também é requerido, se não for definido, a página não será carregada.

- *icon* (string) (opcional): O nome de um ícone do *Awesome Icons* que será exibido no menu, em conjunto com o título. Se esse valor não for definido, um ícone padrão será utilizado.

- *roles* (array): Uma lista de strings contendo os nomes de perfil de usuário que poderão acessar a página. Esse array será útil somente se a aplicação definir o tipo de perfil do usuário no objeto *UserProfile.profile*. Esse valor é opcional, caso não for definido, será setado um array vazio. Porém, se for definido o perfil do usuário no objeto *profile* e essa string não tiver nesta propriedade (*roles*), a página não será exibida.

- *order* (integer): Um valor numérico que define a ordem em que a página será exibida na lista de itens nos menus. O desenvolvedor deverá definir um valor acima de zero e quanto maior o valor, maior a prioridade na lista de itens.

- *isLoginPage* (boolean): Um valor boleano que indica se a página representa a tela de login do aplicativo e deve ser definido se o valor da propriedade *usesLogin* for definido para true no *config.json* do projeto. Se o aplicativo usa login, o *path* da página definida como login será persistido pois será lido por funções internas do aplicativo na inicialização e quando o usuário fizer *logout*. Se mais de uma página for definida como *loginPage* entre os plugins, será utilizada a última página identificada pelo gerenciador de plugins.

- *isHomePage* (boolean): Um valor boleano que indica se a página corresponde a primeira página exibida para o usuário. Se for definido para *true*, o *path* desta página será persisitido e será carregada após o login. No entanto, se o aplicativo não usa login, o aplicativo carregará a página definida como *homePage* na inicialização e será a primeira página exibida para o usuário. Essa propriedade é requerida no modo de layout em pilha e deve ser definida somente pela página correspondente. Se mais de uma página for definida como *homePage*, será utilizada a última página identificada pelo gerenciador de plugins.

- *showInDrawer* (boolean): Um valor boleano que indica se a página poderá ser exibida no menu de layout em pilha. O menu de layout em pilha será instanciado por padrão no layout em pilha, porém, também pode ser instanciado no layout em linha, basta definir a propriedade *usesDrawer* para *true* no arquivo de configuração. O layout em linha utiliza uma barra de botões como menu e com isso, é possível permitir acesso a páginas diferentes a partir dos dois menus.

- *showInTabBar* (boolean): Um valor boleano que indica se a página poderá ser exibida no menu de layout em linha. O objetivo desse flag é permitir exibir páginas diferentes nos menus quando o *Drawer* menu estiver sendo utilizado, permitindo assim, exibir páginas diferentes nos dois menus.

As páginas serão instanciadas sob demanda quando o aplicativo tiver utilizando o layout em pilha, neste caso quando o usuário clicar em um item da lista, a página correspondente será instanciada e ficará visível para o usuário. No layout em pilha, o menu é exibido pelo componente *Menu.qml* que consiste de uma instância do objeto *Drawer* do *QuickControls* e as páginas serão listadas verticalmente. A figura a seguir apresenta um exemplo deste menu com uma lista simples de páginas.

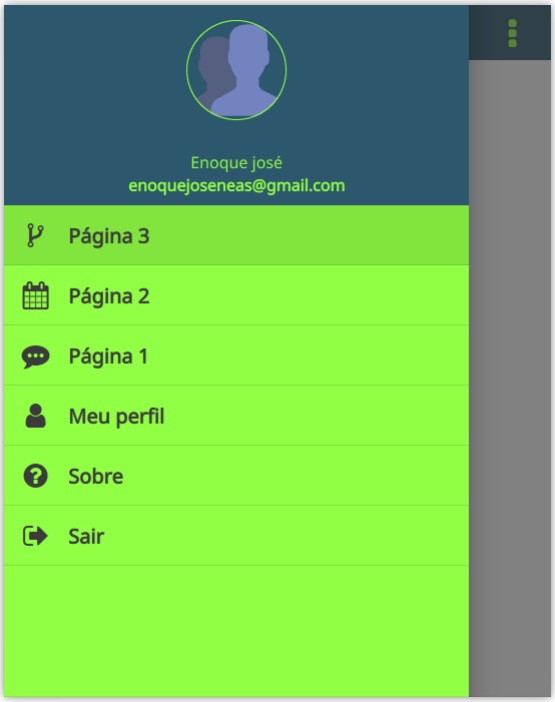


Figura 2: Exemplo de uma lista de páginas no menu de layout em pilha

Quando o layout em linha estiver sendo utilizado, uma lista horizontal de botões será acidionado no rodapé da janela do aplicativo permitindo ao usuário alternar entre as páginas disponíveis. Porém, no layout em linha, todas as páginas serão instanciadas no início da aplicação e terá um botão associado a cada página. No layout em linha, o menu corresponde ao componente *TabBar.qml* também do *QuickControls* com algumas modificações. A figura a seguir apresenta um exemplo deste menu com uma lista simples de páginas. A exibição do título da página pode ser ocultada setando a opção *showTabButtonText* para false no arquivo de configuração do projeto.

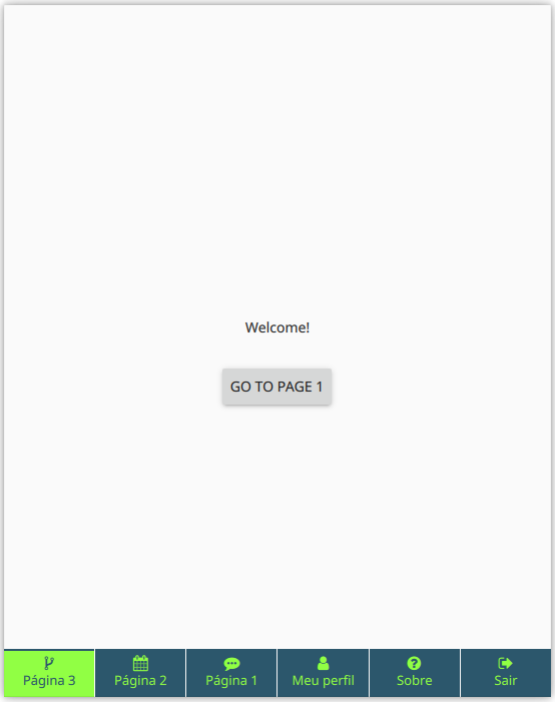


Figura 3: Exemplo de uma lista de páginas no menu de layout em linha

### 4.4.1 Gerenciamento de plugins

A classe *PluginManager* é responsável por carregar os plugins, percorrendo todos os arquivos dentro do diretório *plugins* e analizando as propriedades definida no arquivo *config.json* de cada plugin. Em cada arquivo, é verificado as propriedades definida para cada página adicionando-a como objeto em um array. Após ler todos os plugins, o array de objetos é persistido nas configurações da aplicação para que na próxima inicialização não precise iterar novamente o diretório, lendo as definições dos plugins das configurações, exceto se houver uma atualização da aplicação ou quando a aplicação é executada em modo *debug*. A cada inicialização, é feito uma verificação da versão do aplicativo, que também é persistida nas configurações, se houver diferença entre versão em execução da versão salva na execução anterior (se existir), os plugins serão recarregados. Além disso, esta classe também é reponsável por deletar todos os arquivos de cache contido no diretório de cache da aplicação a cada release. Outra responsabilidade dessa classe, é a criação da tabela de banco de dados do plugin se o plugin dispor de um arquivo *plugin_table.sql* em seu diretório. O banco de dados da aplicação é criado por um objeto que gerencia as operações de persistência e será detalhado em outro tópico. O diagrama a seguir, apresenta as operações e atributos da classe *PluginManager*.

A instância de *PluginManager* é criada pelo objeto *app* na inicialização do aplicativo e o processo de carregamento dos plugins é feito antes de instanciar qualquer componente visual, na invocação do método *loadPlugins*. O objeto *pluginManager* será destruido após o carregamento dos plugins para manter baixo consumo de memória pelo aplicativo.

## 4.5 A classe App

A classe *App* é um componente importante nesta arquitetura, suas responsabilidades consiste de instanciar *PluginManager* que carregará os plugins, carregar o arquivo *config.json* principal que

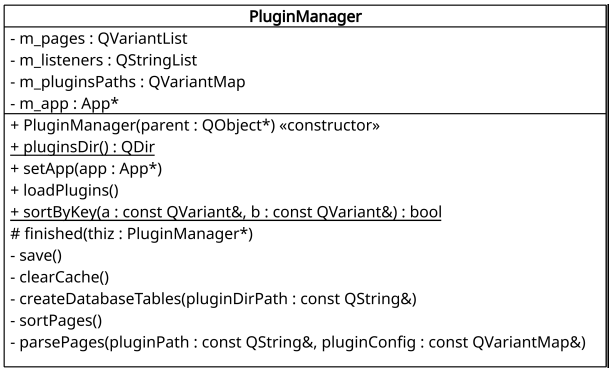


Figura 4: Diagrama da classe PluginManager

contém os parâmetros da aplicação e gerenciar a persistência das configurações da aplicação via *QSettings*[5]. A classe *App* é instanciada na inicialização do aplicativo e é registrada no contexto da aplicação para que os plugins e componentes qml possam invocar seus métodos públicos definidos como *Q_INVOKABLE*, como por exemplo, o método *readSettings* que retorna um tipo genérico de dado, lido das configurações do aplicativo através de algum parâmetro que identifique o dado a ser lido. Outra tarefa que corresponde a esta classe, é a criação de uma conexão com a atividade do aplicativo no android, que corresponde a uma classe. Essa atividade poderá receber parâmetros de eventos como *push notification* ou token de registro no *Firebase*, quando utilizado pela aplicação. O processo de registro do token e o recebimento de notificações via push é feito por objetos java em um processo separado da aplicação, e quando a aplicação estiver em execução, a atividade do aplicativo passará os argumentos recebidos do push ou token para a aplicação através de uma chamada ao método estático *fireEventNotify* da classe *App*.

No iOS, o objeto correspondente a APP também poderá receber argumentos vindos do *QtAppDelegate* objeto que carrega os requisitos da plataforma e inicializa a aplicação. Os argumentos que podem ser recebidos são o *token* de registro no *Firebase* e mensagens de notificações via push.

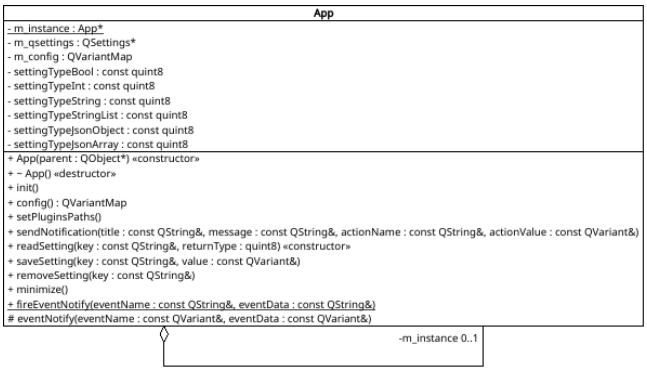


Figura 5: Diagrama da classe App

A classe *App* também é responsável por definir o estilo de widgets utilizado pelos componentes qml. O tipo de estilo deve ser definido pelo usuário no arquivo de configuração *config.json*, na propriedade *applicationStyle* e será passado diretamente para o ob-

[5]Uma classe do Qt que fornece um mecanismo de leitura e escrita de dados no dispositivo através de métodos parametrizados.

jeto *QQuickStyle*. Os possíveis valores para esta propriedade serão detalhados na sessão que descreve o arquivo *config.json*. A instância desta classe adicionará no objeto *Config* a cada inicialização, uma propriedade contendo o path de cada plugin para que os plugins possam acessar os arquivos em seu diretório usando essa propriedade. Exemplo: Considerando que *Session* seria um plugin, os arquivos deste plugin poderiam acessar outros arquivos fazendo uma referência do tipo *Config.plugins.session + "File2.qml"*.

## 4.6 O arquivo config.json

O arquivo *config.json* é um arquivo importante desta arquitetura, ele é utilizado como arquivo de configuração, porém, suas informações não serão persistidas. O conteúdo deste arquivo será repassado para a aplicação como um objeto javascript e o usuário poderá adicionar propriedades a serem lidas pelos plugins. No entanto, as propriedades fornecidas no modelo desta arquitetura não devem ser removidas. Tanto plugins como os componentes internos fazem uso das propriedades definidas neste arquivo. As principais propriedades e seus possíveis valores serão descritos a seguir:

- *appName* (string): O nome do aplicativo. Este valor será setado em *QApplication* na chamada ao método *setApplicationName*. Os plugins poderão ler essa propriedade acessando *Config.appName*;
- *appDescription* (string): Uma descrição sobre o aplicativo. Os plugins poderão ler essa propriedade acessando *Config.appDescription*;
- *organizationName* (string): O nome da organização. Será passado pelo objeto *App* para o objeto *QApplication* e utilizada internamente pelo Qt para definir o nome do diretório de configurações persistentes da aplicação (via *QApplication.setOrganizationName*).
- *organizationDomain* (string): O domínio da organização, como por exemplo *qt.project.org*. É utilizado pelo Qt internamente.
-

O arquivo *config.json* que contém propriedades requeridas pela aplicação será registrado no contexto da aplicação como um objeto javascript identificado por *Config*, apenas como leitura.

## 4.7 Comunicação entre os objetos

Esta seção descreve o processo de comunicação entre objetos ou componentes dentro do aplicativo...

## 4.8 Comunicação com serviço REST

Esta seção descreve o processo de comunicação entre objetos ou componentes dentro do aplicativo...

## 4.9 Gerencimento de Rede (HTTP)

### *4.9.1 O Componente RequestHttp*

O Objeto RequestHttp....

## 4.10 Gerencimento de Banco de Dados

### *4.10.1 A classe Database*

### *4.10.2 O Componente DatabaseComponent*

O Objeto ModelData....

## 4.11 Componentes Visuais

O Objeto ModelData....

## 4.12 O Componente Basepage

O Objeto ModelData....

## 4.13 Fluxo de execução

## 4.14 Métodos para utilização da arquitetura

– descrever o README do projeto no github

OU: Exemplo (a ser editado): Para se utilizar a arquitetura desenvolvida, deve-se seguir uma determinada ordem de atividades (Figura 26), que deve se iniciar no nível arquitetural "escopo" (seção 3.1), passando pelos níveis arquiteturais "modelo de negócios" (seção 3.2) e "modelo de sistema" (seção 3.3), até chegar ao nível arquitetural "modelo tecnológico" que deve ser criado pelo usuário desta arquitetura.

# 5. REFERENCES


[1] Documentação do qt. visão geral e melhores práticas. http://doc.qt.io/qt-5/overviews-main.html. [Online; acessado em 21/08/2017].

[2] Guideline: Architectural view. http://epf.eclipse.org/wikis/openuppt/openup_basic/guidances/guidelines/architectural_view,_T9nygClEEduLGM8dfVsrKg.html. [Online; acessado em 04/10/2017].

[3] U."Acer, A."Mashhadi, C."Forlivesi, and F."Kawsar. Energy efficient scheduling for mobile push notifications. In *Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MOBIQUITOUS'15, pages 100–109, ICST, Brussels, Belgium, Belgium, 2015. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[4] M."C., L."K, and M."F. Reference model for service oriented architecture. https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html, Oct 2006.

[5] D."F. D'Souza and A."C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[6] R."H."B. Feijó. Uma arquitetura de software baseada em componentes para visualização de informações industriais. *Programa de Pós-Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Rio Grande do Norte*, 2007.

[7] M."C. Fernandes. O que é um aplicativo móvel? http://blog.stone.com.br/aplicativo-movel. [Online; acessado em 11/07/2017].

[8] R."T. Fielding. Architectural styles and the design of network-based software architectures. http://dl.acm.org/citation.cfm?id=932295. [Online; acessado em 26/07/2017].

[9] g."GTI. Gerenciando projetos com pmbok. http://www.governancadeti.com/2011/03/gerenciando-projetos-com-pmbok/. [Online; acessado em 14/08/2017].

[10] C."Hofmeister, R."Nord, and D."Soni. *Applied Software Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[11] L."Y."M. Jun, Y.;"Zhishu. Json based decentralized sso security architecture in e-commerce.

http://dx.doi.org/10.1109/ISECS.2008.171, 2008.

[12] A.“H. Kronbauer, C.”A.“S. Santos, and V.”Vieira. Um estudo experimental de avaliaÇÃo da experiÊncia dos usuÁrios de aplicativos mÓveis a partir da captura automÁtica dos dados contextuais e de interaÇÃo. In *Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems*, IHC ’12, pages 305–314, Porto Alegre, Brazil, Brazil, 2012. Brazilian Computer Society.

[13] J.“P. LAUDON, Kenneth C.;”LAUDON. *SISTEMAS DE INFORMAÇÃO: COM INTERNET*, volume Unico. LTC, Rio de Janeiro, Brasil, fourth edition, 1999.

[14] J.“C. Leite. Design da arquitetura de componentes de software. https://www.dimap.ufrn.br/~jair/ES/c7.html, 2000.

[15] P.”LEVY. Cyberdémocratie, 2002.

[16] S.“M. and G.”D. *Software Architecture. Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[17] J.“McCarthy and P.”Wright. Technology as experience. *interactions*, 11(5):42–43, Sept. 2004.

[18] M.“MELOTTI. Creama: Uma arquitetura de referência para o desenvolvimento de sistemas colaborativos móveis baseados em componentes. http://repositorio.ufes.br/handle/10/4270, Aug 2014. http://repositorio.ufes.br/handle/10/4270?mode=full.

[19] E.”C. Microsoft, F.“C. IBM”Research, G.“M. Microsoft, and S.”W. IBM“Research. Web services description language (wsdl) 1.1. https://www.w3.org/TR/wsdl. [Online; acessado em 24/07/2017].

[20] C.”Pabllo, R.“Soto, and J.”Campos. Mobile medication administration system: Application and architecture. In *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*, EATIS ’08, pages 41:1–41:4, New York, NY, USA, 2008. ACM.

[21] C.“Perepletchikov, M.;”Ryan. A controlled experiment for evaluating the impact of coupling on the maintainability of service-oriented software., 2011.

[22] K.“Raymond. *Reference Model of Open Distributed Processing (RM-ODP): Introduction*, pages 3–14. Springer US, Boston, MA, 1995.

[23] C.”Szyperski, J.“Bosch, and W.”Weck. Component-oriented programming. *Object-Oriented Technology ECOOP’99 Workshop Reader Lecture Notes in Computer Science*, page 184–192, 1999.

[24] B.“G. e. L.”G. Valéria“Feijó. Heurística para avaliação de usabilidade em interfaces de aplicativos smartphones: utilidade, produtividade e imersão. *Design e Tecnologia*, 3(06):33–42, 2013. [Online; acessado em 22/07/2017].

[25] A.”P. O.“S. Vermeeren, E.”L.-C. Law, V.“Roto, M.”Obrist, J.“Hoonhout, and K.”Väänänen-Vainio-Mattila. User experience evaluation methods: Current state and development needs. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI ’10, pages 521–530, New York, NY, USA, 2010. ACM.

[26] S.“P. Zambiasi. Uma arquitetura de referência para softwares assistentes pessoais baseada na arquitetura orientada a serviços. *Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas*, (77):331, 2012. https://repositorio.ufsc.br/handle/123456789/99348.