UNIVERSIDADE FEDERAL DE UBERLÂNDIA

João Pedro Galvão Rodrigues

Gestão do canteiro de obras: uma solução de *software* baseada em tarefas

Uberlândia, Brasil

2019

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

João Pedro Galvão Rodrigues

Gestão do canteiro de obras: uma solução de *software* baseada em tarefas

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Maria Adriana Vidigal de Lima

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2019

João Pedro Galvão Rodrigues

Gestão do canteiro de obras: uma solução de *software* baseada em tarefas

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 24 de novembro de 2012:

Maria Adriana Vidigal de Lima
Orientador

Professor

Professor

Uberlândia, Brasil
2019

Dedico esse trabalho a minha mãe, Josiane Guimarães Galvão Rodrigues, que sempre teve como realização da sua própria vida ver os seus dois filhos concluírem uma graduação de ensino superior.

Agradecimentos

Gostaria de agradecer a todos os professores e educadores que tive em minha carreira estudantil até esse momento. Em especial a professora Maria Adriana Vidigal de Lima, que foi uma representante espetacular dessa carreira durante a orientação desse trabalho.

Gostaria de agradecer a minha namorada e companheira, Débora Borges Duarte, que me motivou até o fim na conclusão deste trabalho.

Gostaria de agradecer também ao colega Benjamim Bueno que trouxe o problema que gerou a ideia deste trabalho até mim, além de me acompanhar na execução do mesmo.

The test of the machine is the satisfaction it gives you. There isn't any other test. If the machine produces tranquility it's right. If it disturbs you it's wrong until either the machine or your mind is changed." (Robert M. Pirsig)

Resumo

Os sistemas de gestão e acompanhamento para área de execução da construção civil brasileira, o canteiro de obra, carecem de funções fundamentais que facilitem o dia-a-dia de seus usuários, como: geração de relatórios automáticos para *reports* diários dos contratantes; criação de avisos de pontos fundamentais para realização de atividades futuras e/ou que estão em pausa devido a algum problema, integração com BIM (*Building Information Modeling*), entre outros. Este trabalho tem como objetivo propor uma aplicação para dispositivos móveis e *web* que possa ser, para os mestres de obras e outros cargos de gestão neste meio, uma plataforma na qual todos os dados da obra podem ser encontrados e controlados. Durante o desenvolvimento desse trabalho foi construída a aplicação proposta, utilizando o *framework* React Native, e entende-se que o sistema resultante é um primeiro passo para outros avanços a serem feitos na informatização da gestão da construção civil.

Palavras-chave: Sistema de Gestão. Canteiro de Obras. Tarefas. BIM. Framework Mobile.

Lista de ilustrações

Lista de abreviaturas e siglas

APP	Aplicativo
API	Application Programming Interface
BIM	Building Information Modeling
DB	Data Base
HTTP	Hypertext Transfer Protocol
iOS	iPhone Operationa System
JSON	JavaScript Object Notation
JSX	JavaScript XML
JWT	JSON Web Token
MVC	Model View Controller
MS	Microsoft
PIB	Produto Interno Bruto
SQL	Structured Query Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

Sumário

1 Introdução

A construção civil representou no ano de 2017, 13% do PIB mundial, o que claramente a coloca entre um dos setores mais importantes na economia mundial. Enquanto a construção civil cresce ao ano apenas 1% em média, nos últimos 20 anos, a economia mundial como um todo cresceu 2.8% e setores como a manufatura, cresceram 3.6% (McKinsey Global Institute, 2017). Esse tipo de dado indica que existe uma deficiência no crescimento de produtividade do setor, e uma das preocupações que se acentua é a baixa adoção de novas tecnologias a fim de otimizar sua entregas. Este é o ponto que mais tem impulsionado a produtividade de diversos outros setores, e que é normal que seja pouco abordado num setor já consolidado há tanto tempo.

Pode-se notar, especificamente no setor da construção civil, quanto à adoção de processos e tecnologias, que existem duas grandes áreas focais: planejamento e execução. Na área do planejamento podem ser ressaltados claros avanços sendo feitos, inclusive iniciativas e colaborações por parte do Governo Brasileiro, que busca com a Estratégia BIM BR, tornar obrigatório o uso do BIM a partir de 2021 para novas construções relevantes do setor público (Ministério da Economia, 2019). BIM (*Building Information Modeling*) pode ser traduzido como Modelagem de Informação da Construção, e é um conceito que tem como ideia a criação, de forma precisa, de modelos virtuais de qualquer edificação. Tais modelos virtuais são muito realistas e fornecem um importante suporte às diversas tarefas realizadas em diferentes fases de uma construção civil. Ainda, permitem a realização de análises e controles mais eficientes e propiciam a otimização dos recursos.

Um bom trabalho de planejamento de obra, realizado utilizando-se o conceito BIM, diminui a ocorrência de conflitos e retrabalhos. Porém, o que acaba ocorrendo é que, apesar dos avanços no planejamento, a execução da obra tem visto poucas inovações focadas nela, muitas vezes apenas herdando soluções do planejamento, como o BIM, que é muito bem explorado na elaboração do projeto, mas para a execução acaba se tornando pouco mais que uma planta em três dimensões. Um fato que pode ser destacado em relação à necessidade da criação de novas tecnologias na execução de projetos, é que nesta etapa encontram-se, em média, profissionais com menor nível de instrução educacional, o que torna ainda mais importante o uso de tecnologias que propiciem a diminuição de erros, possibilitem checagens e aprimorem a comunicação entre os profissionais envolvidos (CORDEIRO, 2002).

Considerando-se a necessidade de se inovar tecnologicamente no setor de execução da construção civil, surge este trabalho, que propõe a construção de uma plataforma de gestão e acompanhamento de tarefas, a fim de tornar a execução de uma obra tão eficiente

quanto o planejamento e criar um conjunto de informações suficientes para aproximar ao máximo o planejamento da realidade do canteiro de obra.

1.1 Objetivos

O objetivo principal deste trabalho é realizar a concepção, modelagem e implementação de um sistema de gestão de tarefas de execução de obras da construção civil, focado em canteiros de obras brasileiros. Nesta proposta, pretende-se que o gestor seja capaz de gerenciar as tarefas que estão acontecendo no dia-a-dia da obra, tendo a possibilidade de atrelar à estas tarefas todos os dados que forem relevantes: resultados de verificação de dados, análises diversas (como medições), anotações sobre problemas encontrados e inclusão de fotos. Uma vez que os *inputs* diários estejam armazenados, é importante que a aplicação seja capaz de gerar relatórios automáticos para otimizar o tempo do gestor. Como colaboração na gestão da execução, é importante que o sistema tenha interoperabilidade com ferramentas baseadas no conceito BIM, notadamente as que permitam explorar visualmente o projeto da obra.

1.2 Método

Para que os objetivos apresentados na Seção 1.1 fossem alcançados, os seguintes passos foram propostos para o desenvolvimento deste trabalho:

- Criar um protótipo de solução para a questão apresentada:
  - Fazer entrevistas com profissionais que atuam no gerenciamento de um canteiros de obra em empresas de diferentes tamanhos (dos gerentes de execução, e outros cargos semelhantes de mais alto nível, até os mestres de obra, ou cargos semelhantes que representem o mais baixo nível ainda gerencial) para entender sua metodologia de gestão;
  - Estudar ferramentas de gestão da Engenharia Civil;
  - Fazer reuniões de *brainstorming* com pessoas com conhecimento na área de Engenharia Civil para criar os conceitos da solução;
  - Definir a plataforma computacional para a implantação da solução;
  - Fazer desenhos das telas em um software de modelagem para criação de um protótipo.
- Programar a solução prototipada:
  - Definir a forma de desenvolvimento da aplicação quanto ao seu *front-end*;

- Enumerar as APIs externas que serão necessárias;
- Definir a forma de desenvolvimento da aplicação quanto ao seu *back-end*;
- Escolher o tipo de banco de dados mais adequado;
- Implementar o aplicativo usando a biblioteca *React Native* e arquitetura *Flux*;
- Implementar a API usando a linguagem *JavaScript*.

1.3 Organização do Trabalho

A fim de proporcionar aos leitores uma maior absorção do conteúdo proposto, este trabalho está organizado da seguinte forma: o capítulo dois aborda os principais conteúdos técnicos que serão importantes durante a leitura do restante do material; o capítulo três apresenta a proposta de de software para o tema em questão, tratando tanto da ideia quanto da aplicação prática da mesma, e, por fim, o capítulo quatro aborda qual a aplicabilidade da solução proposta, qual sua relação com soluções parecidas e quais os possíveis desdobramentos futuros desse projeto.

2 Fundamentação Teórica

Neste capítulo são apresentados os diferentes conceitos e tecnologias usadas durante o desenvolvimento deste trabalho. Eles estão agrupados, respectivamente, em metodologias de gestão aplicadas a canteiro de obras, conceitos e ferramentas de construção de protótipos, tecnologias de desenvolvimento *front-end* e tecnologias de desenvolvimento *back-end*.

2.1 Metodologias de gestão aplicadas a canteiros de obras

O estudo da área de gestão de projetos aplicada a canteiro de obras foi feito de forma prática, através de visitas a construtoras da cidade de Uberlândia, em que buscou-se entender quais as metodologias seguidas e as ferramentas utilizadas para apoiar na aplicação da mesma.

As empresas visitadas apontaram a metodologia ágil como sendo adequada ao problema da execução e acompanhamento de obras (FROTA; WEERSMA; WEERSMA, 2016; MENDONÇA et al., 2018). Nesta metodologia, a cada semana retiram-se atividades de um arquivo principal (criado no MS Project ou no Excel) a serem executadas. Este arquivo é considerado o *backlog* da obra, e uma seleção de atividades que devam ser realizadas na semana que está para começar é chamada de *sprint*.

As atividades são, então, comunicadas em uma reunião logo no começo da semana para o mestre de obras e seus encarregados que, utilizando um painel localizado no escritório da obra, fazem a inclusão das atividades utilizando o método Kanban (que permite que cada atividade seja associada à um dos estados: “aguardando”, “sendo realizada” ou “concluída”. Ao fim de cada *sprint* é feita uma reunião de identificação das principais dificuldades encontradas naquela semana e o repasse das atividades realizadas, conforme o Kanban. Após essa reunião, o engenheiro de execução faz a atualização no seu cronograma, no arquivo geral de atividades (MS Project ou Excel), para que possa iniciar-se a idealização do *sprint* seguinte.

Além do uso de metodologia ágil, durante as visitas às construtoras, verificou-se o uso de arquivos no conceito BIM (*Building Information Modeling*), como sendo importantes fontes de informação a serem utilizadas durante o processo da construção. Os projetos baseados no conceito BIM integram a modelagem em 3D à informações do ciclo de vida completo de um empreendimento de construção. A base de um sistema que usa o conceito BIM é o banco de dados que, além de exibir a geometria dos elementos construtivos em três dimensões, armazena seus atributos, diminui conflitos entre elementos construtivos,

facilita revisões e aumenta a produtividade (AZHAR, 2011).

Os arquivos de projeto no conceito BIM permitem navegações em diferentes perspectivas e abrem novas possibilidades para mestres de obras e encarregados compreenderem mais sobre cada objeto do projeto em tempo real, considerando especialidade a que pertencem, características geométricas, físicas entre outras. Porém, vê-se na prática, que o BIM é ainda pouco explorado, conforme abordado em Manzione (2013), pois esse tipo de tecnologia pode ir muito além da modelagem e fornecimento de informações ligadas apenas ao projeto da obra, podendo se estender para o processo de gestão e controle do canteiro de obra.

2.2 Conceitos e Ferramentas de Criação de Protótipos

Para a criação da modelagem do sistema proposto nesse trabalho foi necessária a utilização de uma ferramenta de design de experiência do usuário. A ferramenta escolhida foi a Adobe XD, do inglês *"Experience Design"*. Esta ferramenta disponibilizada pela Adobe, tem foco na criação rápida de protótipos de aplicações, tanto para dispositivos móveis quanto *web*, que são capazes de gerar um entendimento inicial de como será a experiência do usuário e focar seu design em melhorá-la (ADOBE, 2019).

Além da definição e estudo de uma ferramenta como o Adobe XD, foi importante o estudo de uma técnica de design para criação das telas do sistema proposto no trabalho. Como linguagem de design, foi escolhida a Material Design, criada pela Google que condensa bons princípios de *design* e elenca uma série de elementos visuais que são mais sugestivos para o usuário e tendem a causar uma melhor experiência de uso (GOOGLE, 2019).

2.3 Tecnologias de desenvolvimento *front-end*

A linguagem JavaScript foi concebida para ser utilizada na construção de animações e para fornecer interatividade para páginas web. Com a necessidade da comunicação entre páginas web e servidores, a linguagem adaptou-se para possibilitar a programação da comunicação com servidores, e com isso, passou a realizar tarefas mais importantes em aplicações web. Além disso, houve a necessidade de manipulação de modelos de objetos e de permitir melhor interatividade com usuários através de eventos de *browsers*, como cliques de mouse. As funções já existentes não possuíam uma solução eficiente para tais necessidades e, por isso, começaram a surgir bibliotecas de apoio como *Angular*, *Vue* e *React* (EISENMAN, 2015). Desta forma, a linguagem JavaScript vem sofrendo alterações em função de melhores práticas de desenvolvimento e aprimoramento de suas características fundamentais.

Para o desenvolvimento do código da parte da aplicação que será a interface do usuário com o sistema, o *front-end*, foi escolhida a biblioteca *React* e sua versão *React Native* que possibilita ao desenvolver, com a escrita de apenas um código, gerar aplicações que funcionem nas duas principais plataformas móveis: iOS e Android, e também em navegadores (REACTNATIVE, 2019a).

As bibliotecas *React* e *React Native* foram criadas pela empresa Facebook, especificamente para uma linguagem, denominada JSX e usada apenas por estas ferramentas. Esta linguagem (JSX) é uma extensão de sintaxe da linguagem JavaScript contendo algumas características de uma linguagem de *template*, o que torna possível ao programador renderizar elementos *React* dentro do seu código JavaScript (REACT, 2019).

A arquitetura de projeto *Model View Controller* (MVC) define uma separação entre as camadas de programação *Model*, *View* e *Controller*. A camada *Model* representa a camada dos dados da aplicação, a *View* reune os elementos visíveis para o usuário e a *Controller* identifica e propaga as alterações no *Model* causadas pelas *Views* (GAMMA et al., 2008). Nesta arquitetura, a separação de todas as responsabilidades dos componentes do software é bastante clara. Neste contexto, o uso do React permite a fusão das camadas *controller* e *view*. Esta fusão possibilita que, dentro de um componente visto pelo usuário, já estejam definidas quais suas ações e como elas se comunicam com a camada *Model* (dos dados). Assim, pode-se estabelecer o reuso de um mesmo componente em diferentes interfaces.

O único ponto não alterado num MVC padrão quando se usa o *React* sem outras bibliotecas é o *Model*. Porém, para trazer uma maior versatilidade a esta camada, é possível a aplicação da arquitetura *Flux*, um padrão arquitetural criado pelo Facebook para adaptar o que seria a camada *Model* em um MVC com React (FACEBOOK, 2019).

Das implementações da arquitetura *Flux*, optou-se pelo uso da biblioteca *Redux*, por ser uma das implementações mais bem otimizadas, e que garante que o *Model* indique que há alterações apenas quando é realmente necessário, mantendo assim, um menor número de novas renderizações da *View* (REACTREDUX, 2019).

2.4 Tecnologias de desenvolvimento *back-end*

O *back-end* de uma aplicação é a parte do programa que fica invisível ao usuário e que contém a implementação das regras de negócio, sendo vital para o funcionamento da aplicação. Consiste geralmente de servidor, banco de dados, e da codificação do negócio propriamente dito. A aplicação pode ser escrita em muitas linguagens de programação, e é necessário que o *back-end* disponibilize diversos pontos de comunicação com o *front-end*, para que este último possa consumir os dados de um banco de dados e apresentá-los ao usuário de forma adequada e agradável.

Neste trabalho, as tecnologias usadas no *back-end*, que garantem a infraestrutura de armazenamento para os dados manipulados pelos usuários na aplicação foram construídas em JavaScript. Historicamente, a linguagem JavaScript é conhecida como uma linguagem de *front-end*, já que roda em navegadores. Para que se possa utilizar a linguagem JavaScript no servidor é necessária uma tecnologia de suporte, como Node.js. Node.js é um ambiente utilizado para executar aplicações JavaScript. Além de oferecer um ambiente multiplataforma, fornece recursos apoiar a implementação das aplicações. É baseado na *engine V8 JavaScript Engine*, desenvolvida pela Google e usada no Google Chrome. Desta forma, a biblioteca Node.js torna o *V8 JavaScript Engine* um executor para o código JavaScript do lado do *back-end* (SYED, 2014).

No *back-end* também é necessária a definição de um banco de dados e, atualmente, além da tradicional opção de bancos SQL, em que estruturam-se os dados de maneira relacional, existe a opção de armazenamento em bancos NoSQL. Os bancos NoSQL se subdividem em quatro principais categorias de modelagem de dados, chave-valor(Amazon DynamoDB, Redis, Oracle NoSQL), baseado em famílias de colunas (Google BigTable, Cassandra, HBase), baseado em grafos (Neo4J, Arango DB, Amazon Neptune) e, finalmente, orientado a documentos (MongoDB, CouchDB, CrateDB) (SADALAGE, 2012). Para o desenvolvimento desse projeto o MongoDB foi escolhido pela naturalidade com qual ele lida com objetos JavaScript, padronizados segundo a notação JSON (JavaScript Object Notation). O padrão JSON será o modo em que todos os dados serão manipulados de uma ponta a outra da aplicação.

Por fim, para aplicações que exigem segurança nos dados, onde há perfil de acesso e cada usuário tem privilégios de alterações de dados diferentes, como é o caso da aplicação deste trabalho, é importante que a comunicação com o *back-end* ocorra com a certeza que o aplicativo se encontra logado com um determinado usuário, que realmente esteja certificado seu acesso as informações ali sendo transitadas. Para garantir essa segurança foi definido o uso de JSON Web Tokens (JWT) (PEYROTT, 2016). O JWT é um padrão que define como transmitir e armazenar objetos JSON de forma concisa e segura entre aplicações distintas. Os dados podem ser validados quando necessário, pois o *token* é assinado digitalmente.

3 Construção da solução

Este capítulo faz uma descrição geral do aplicativo e suas funcionalidades, explicando detalhadamente o seu funcionamento, como foi realizado o desenvolvimento das funcionalidades, as técnicas utilizadas para solucionar os problemas encontrados, as motivações para uso de APIs externas e os padrões de projeto que foram usados durante a codificação.

A seção 3.1 apresenta conceitos importantes para o entendimento do funcionamento do aplicativo e na seção 3.2 uma visão geral sobre o funcionamento do aplicativo móvel é desenhada, considerando-se o ponto de vista do que é perceptível ao usuário durante a utilização do mesmo. Cada uma das seções do aplicativo é detalhada, com suas telas e funcionalidades. Em seguida, a seção 3.3 retrata a visão geral do funcionamento da parte *web*, explicando brevemente suas telas e funcionalidades. Por fim, a seção 3.4 aborda com detalhes as tecnologias utilizadas no desenvolvimento do trabalho, tanto na modelagem quanto na codificação, assim como APIs externas. Também são explicados os padrões de projeto empregados, a motivação e os benefícios da utilização dos mesmos.

3.1 Conceitos do aplicativo

Este aplicativo tem como objetivo informatizar os processos de gestão da construção civil dentro de um canteiro de obra, utilizando métodos ágeis, que já são, em algumas construtoras usados, mas que carecem de uma ferramenta própria para o setor de construção civil para aplicá-los. A aplicação servirá para gerir as tarefas do dia-a-dia da obra, garantindo um maior acesso à informação por parte de todos os envolvidos nas diversas atividades que acontecem no canteiro.

Nas subseções a seguir são explicados conceitos que são importantes para se entender o funcionamento do aplicativo.

3.1.1 Tarefa

A gestão proposta pelo aplicativo é feita por tarefas, pois é possível nomear tudo que acontece na construção civil como uma tarefa, de uma pintura ou concretagem, a uma preparação de equipamentos de segurança para executar uma outra atividade. Tudo se encaixa no que pode-se chamar de tarefa no aplicativo. Na Figura 1 é retratado um exemplo de uma sequência de tarefas, relacionadas num arquivo em MSProject.

Como tudo que será feito na obra é uma tarefa, há uma série de informações que podem ser ligadas a cada uma delas:

Nome da tarefa	Duration	Start	Finish
MONTAGEM DA ARMADURA DOS PILARES	2.0d	07/02/19	07/03/19
MONTAGEM DOS PILARES E VIGAS (FORMA)	4.0d	07/03/19	07/08/19
ESCORAMENTO DO FORRO DE LAJE	1.0d	07/08/19	07/08/19
MONTAGEM DO FORRO DE LAJE	2.0d	07/09/19	07/10/19
CONCRETAGEM DOS PILARES	1.0d	07/11/19	07/11/19
MONTAGEM DA ARMADURA DA LAJE	2.0d	07/12/19	07/15/19
TUBULAÇÃO EMBUTIDA NA LAJE (ELETRÎCA/HIDRAULICA/GAS/INCËN...	3.0d	07/15/19	07/17/19
CONCRETAGEM DA LAJE	1.0d	07/18/19	07/18/19

Figura 1 – Exemplo de tarefa num arquivo em MSProject.

- **Lembrete**: um aviso que está vinculado a uma atividade e tem uma pessoa ou um grupo de pessoas ou ainda todos os envolvidos da obra, como "alvo". Esse "alvo" receberá, com uma frequência determinada pelo criador do lembrete, uma notificação o "lembrando" de uma determinada ação que deve ser tomada e que tem relação com uma atividade. Entende-se que um lembrete é diferente de uma atividade pois ele não estava no planejamento da obra e a execução do mesmo, ou ainda, a lembrança do mesmo, é importante para execução da atividade com a qual ele está vinculado.

- **Problema**: semelhante a um lembrete, tem uma pessoa ou um grupo de pessoas ou todos os envolvidos da obra, como "alvo", e esse "alvo" também recebe uma notificação configurada pelo criador do problema, assim como no lembrete. Entende-se que um problema é diferente de uma tarefa e de um lembrete, pois ele não estava no planejamento da obra, assim como o lembrete. Mas, diferentemente do lembrete, ele não só é importante como também é um impeditivo para a execução da tarefa, ou seja, caso o problema não seja executado, a tarefa não poderá ser concluída. Por isso, o problema deve ter uma data prevista de conclusão.

- **Datas de início e fim planejadas e real**: essa informação é bem intuitiva e serve para que todos tenham acesso à data planejada de começo e de fim de uma determinada atividade. As datas planejadas são inseridas no momento em que a atividade é criada. Caso a data de fim da atividade seja alterada em algum momento, qualquer atividade que dependa dessa para começar, terá sua data de inicio planejado adiada. A data real de início é preenchida automaticamente quando o usuário identifica que começou a atividade na tela de 'Início de atividades diária'. A data real de fim também é preenchida automaticamente, no momento em que o usuário encerra a atividade na tela do '*Sprint* Semanal'.

- **Dados do projeto**: são informações dos diversos projetos (arquitetônico, estrutural, elétrico, hidráulico, bombeiros, etc) que podem ser incluídas no aplicativo para facilitar o acesso do executor da atividade a essas informações. Por exemplo: se está sendo feita uma atividade de pintura do primeiro pavimento de uma determinada estrutura, o dado do projeto poderia ser a área das paredes pintadas. Esses dados

do projeto são extremamente importantes para a medição, além de facilitar o acesso a informação e fazer com que os executadores tenham que consultar menos vezes os projetos físicos ou, caso não saibam interpretar o projeto físico, consultem menos o engenheiro ou idealizador do projeto.

- **Medições**: uma medição sempre está atrelada a um dado do projeto inserido na tarefa e tem como objetivo identificar o progresso da tarefa e quem fez esse progresso. Esse tipo de informação é importante para facilitar no pagamento de executores terceiros que recebem por trabalho realizado ou ainda para o pagamento de bônus para funcionários internos, uma vez que torna possível acompanhar produtividade.

- **Fotos**: essa também é uma informação bem intuitiva e serve para que seja possível ao usuário do aplicativo inserir fotos junto a uma descrição na atividade. O objetivo dessa informação é trazer para perto do canteiro e do progresso da obra interessados que não podem estar próximas da mesma. Além de gerar a possibilidade de criação automática de relatórios de progresso em fotos, o que torna muito mais tangível o progresso da obra.

- **Qualidade**: no aplicativo, qualidade é um formulário com perguntas e respostas, podendo ter de uma até quantas o usuário achar que é necessário, e é possível que ele identifique se o formulário deve ser preenchido durante, antes, ou no momento de conclusão da tarefa. O objetivo da qualidade é garantir que alguns procedimentos da empresa usuária do aplicativo estão sendo cumpridos. Por exemplo, caso seja necessário, para iniciar uma tarefa, deve-se garantir que existe uma determinada estrutura de segurança instalada. Pode-se criar um formulário com tal pergunta para que antes que a tarefa comece a ser executada, o usuário tenha que responder se essa estrutura já está instalada ou não. O mesmo pode ser feito para iminência da conclusão da tarefa, caso, o usuário queira concluir um assentamento de piso, por exemplo, ele poderia ser perguntado sobre a largura do rejunte instalado, para que a empresa possa garantir que está dentro dos seus padrões. Estes formulários são criados na versão *web* do sistema.

- **Obrigações**: esse é um dado inserido pelo criador da tarefa que tem como objetivo tornar o aplicativo mais engessado, a fim de garantir o preenchimento de algumas das informações supramencionadas em uma tarefa. Ou seja, caso o criador de uma tarefa entenda que para que seja possível a conferência da conclusão da mesma, seja necessária a inclusão de uma foto, ele identifica que este campo é obrigatório. Desta forma, quando a tarefa estiver sendo concluída, é conferido se existe pelo menos uma foto dentro daquela tarefa. Caso exista, o usuário poderá conclui-lá normalmente, caso contrário, será solicitada sua inclusão de foto. O usuário também pode identificar como obrigatórias, além das fotos, as medições.

3.1.2 *Sprint*

Um *sprint* nada mais é que um agregado de tarefas com uma determinada data de começo e uma determinada data de fim. Seguindo portanto, os conceitos padrões de gestão ágil. E para garantir que a metodologia ágil não seja quebrada, é importante lembrar que cada *sprint* deve ter um período não superior a 30 dias, uma vez que cada *sprint* deve garantir uma sensação de "dever cumprido" para os envolvidos no mesmo e deve ter um entregável claro para que os interessados no projeto consigam também enxergar o progresso que aconteceu durante aquele período de tempo.

3.1.3 Obra

Uma obra, no conceito do aplicativo, é um agregado de *sprints* com uma data de início e de fim. É importante que a obra também forneça dados do local onde ela acontece, uma vez que é com esses dados que será possível, no relatório de obra, incluir automaticamente a situação climática da obra. Dados sobre o clima são comumente inseridos nesse tipo de documento, já que esse tipo de fator costuma interferir diretamente na obra.

3.2 Seções do aplicativo móvel

O aplicativo móvel é dividido em seções, algo comum entre os aplicativos de hoje em dia e que torna mais fácil a experiência do usuário, já que fica mais intuitivo de se entender que, em cada uma das diferentes seções, ele estará realizando tipos de atividades diferentes. Além disso, o elemento visual de identificação de qual seção o usuário está, conforme Figura 2, é o mesmo elemento com o qual ele já esta acostumado a interagir em outros apps.

Figura 2 – Navegador de seções.

As funcionalidades de atualização de tarefas e *sprints* estão todas na seção *Home*, explicada na subseção 3.2.1. As outras seções são consideradas acessórias, e derivam, de alguma forma, do uso da seção '*Home*', são elas:

- Seção Projeto, explicada na subseção 3.2.2
- Seção Relatórios, explicada na subseção 3.2.3
- Seção Perfil, explicada na subseção 3.2.4

A subseção 3.2.5 deste trabalho foi reservada para se falar das telas que não foram incluídas em nenhuma seção uma vez que antecedem a interação do usuário com as seções.

3.2.1 *Home*

Esta subseção ganha esse nome exatamente por se tratar da principal seção do aplicativo, pelo menos para quem o usará com muita frequência, uma vez que é nela que são feitas todas as interações com os *sprints*, tarefas, problemas e lembretes. Caso o usuário queira manipular qualquer uma dessas informações, é nessa seção que o usuário será capaz de fazê-lo. E como essas interações devem ser feitas diariamente, pode-se afirmar com certeza, que, se considerado o uso efetivo do aplicativo por uma empresa contratante, essa será a seção mais acessada, pelo menos em números absolutos de acesso. É lógico que para um perfil de usuário ou outro, isso pode mudar, mas de forma geral, é nessa seção que é determinado o fluxo principal do aplicativo.

Essa seção se divide nas seguintes telas e funcionalidades:

- Tela do *Sprint* (Figura 3): tela onde o usuário é capaz de checar as informações do *Sprint*, como: progresso, data de começo e fim, quantidade de problemas e lembretes ativos e um pequeno resumo de cada uma das suas tarefas, com o nome, data em que começou.Se já tiver começado, data em que foi concluído, se já tiver sido concluído, se a tarefa tem algum problema e/ou lembrete e se ela está ou não atrasada. No canto superior também é possível encontrar o nome da obra a qual este *sprint* pertence, informação que está presente em todas as telas dessa seção.

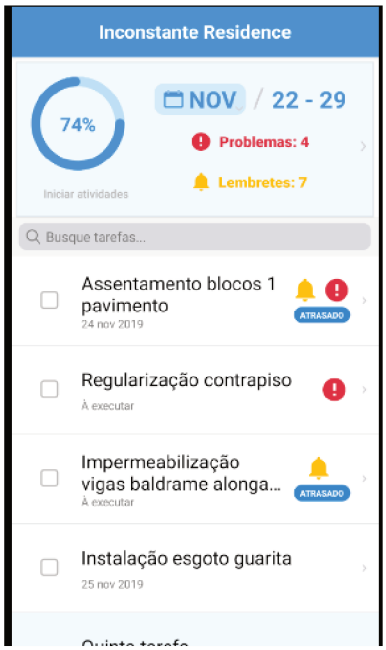

Figura 3 – Tela *Sprint*.

Conforme Figura 4, o usuário pode interagir com esta tela clicando em uma das atividades (item [1]) para ir para a tela daquela atividade especifica. Clicar no pequeno retângulo presente em cada atividade, item [2], pode ser um mecanismo para finalizar aquela atividade. Clicar na quantidade de problemas (item [3]) leva a uma tela com a listagem dos problemas. Clicar na quantidade de lembretes (item [4]) mostra uma tela com a listagem de lembretes. Clicar em iniciar atividades (item [5]) abre a tela de início de atividades. Clicar no mês do *sprint* (item [6]) permite ir para uma tela de *sprints*. Clicar em qualquer outro lugar do cabeçalho que não seja um dos mencionados até agora (item [7]) mostra uma tela com apenas tarefas concluídas. Ou, por fim, inserir algum texto na busca (item [8]) é uma opção para buscar tarefas pelo nome.

- Tela da Tarefa (Figura 5): tela onde o usuário pode conferir todas as informações vinculadas à uma determinada tarefa. Também permite ao usuário vincular novas informações. As informações presentes nessa tela são as explicadas na subseção 3.1.1.

  Conforme Figura 6, o usuário pode interagir com essa tela clicando nos diversos símbolos de adição (item [1]) pra adicionar mais informações daquele tipo à tarefa. Pode clicar em algum dos dados do projeto (item [2]) onde surgirá um *pop-up* para alterá-lo. Pode clicar na medição para adicionar dados da medição ou alterar a medição já criada. Pode clicar na imagem (item [3]) para abrir um *pop-up* onde ela pode ser baixada. Pode clicar em uma das bolinhas embaixo da imagem (item [4]) ou fazer o movimento de *swipe* para trocar de imagem. Pode clicar em um problema (item [5]) para editá-lo. Pode clicar em 'problemas resolvidos' (item [6]) para ir para uma tela com os problemas resolvidos desta atividade. Pode clicar em um lembrete (item [7]) para editá-lo. Pode clicar em 'lembretes resolvidos'(item [8]) para ir para uma tela com os lembretes resolvidos desta atividade. Ou ainda, clicar no botão *Share*(item [9]) para compartilhar um *link* via outros aplicativos que dão acesso direto a esta atividade.

- Tela de Problemas (Figura 7): tela onde o usuário tem acesso a todos os problemas daquela obra, cada problema é apresentado com seu nome e de qual tarefa ele pertence, além de uma *checkbox* para identificar se o mesmo está concluído ou não.

  Conforme Figura 8, o usuário pode interagir com essa tela clicando em um dos problemas (item [1]) para que seja redirecionado à tela de atividade que gerou este problema. Pode clicar no *checkbox* (item [2]) para concluir o problema. Pode clicar no título da tela (item [3]) para ser redirecionado a uma tela de problemas com apenas os problemas já concluídos. Ou, por fim, inserir algum texto na busca (item [4]) para buscar problemas pelo nome ou pelo nome da tarefa que gerou este problema.

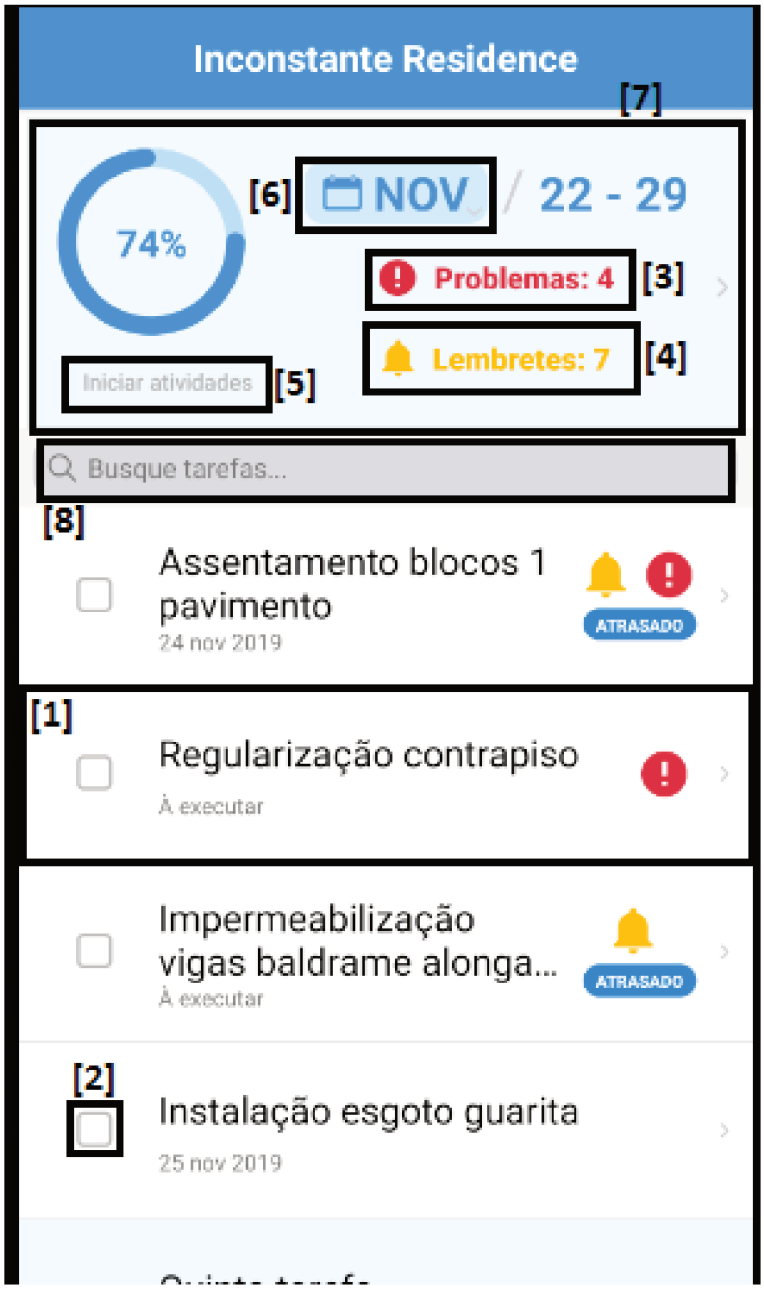

Figura 4 – Tela *Sprint* com marcações.

- Tela de Lembretes (Figura 7): Esta tela em nada se difere da tela de problemas, apenas o conteúdo, ao invés de ser sobre problemas, é sobre lembretes.
- Tela de Tarefas Concluídas (Figura 9): Esta tela se difere pouco das telas de problemas e lembretes. Ao invés de trazer informações sobre estes, ela traz informações de tarefas já concluídas e ao invés do nome da tarefa vir subscrito, é a data de começo e fim daquela tarefa que está abaixo de seu nome. As interações são as mesmas das encontradas nas telas de lembretes e problemas.
- Tela de início de tarefas (Figura 10): Esta tela é sempre a primeira tela a aparecer no aplicativo, caso o usuário esteja abrindo o aplicativo pela primeira vez no dia

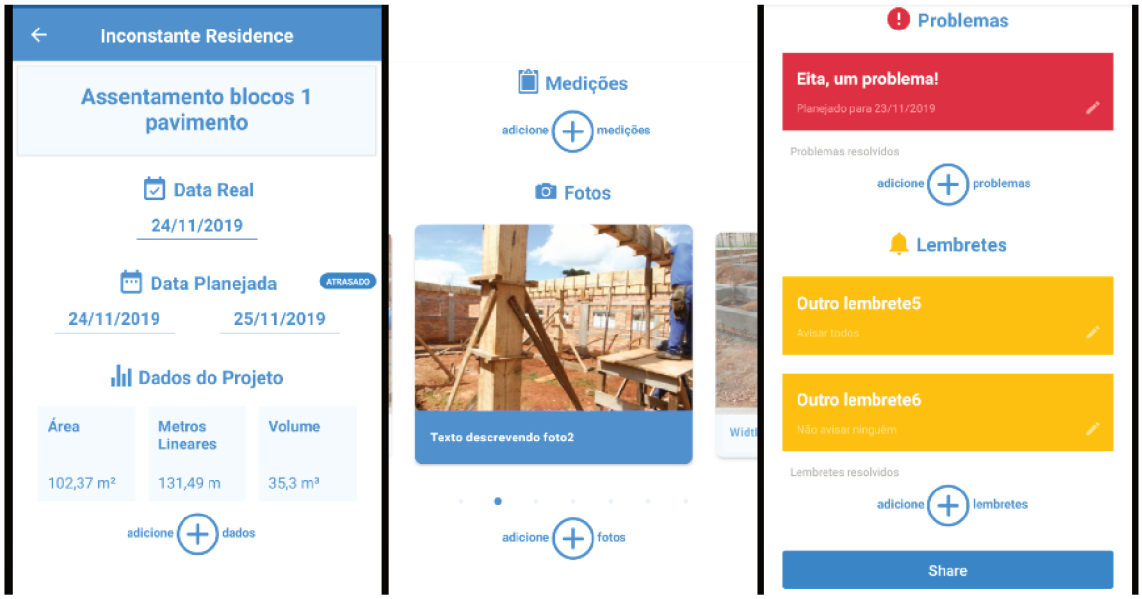

Figura 5 – Tela Tarefa.

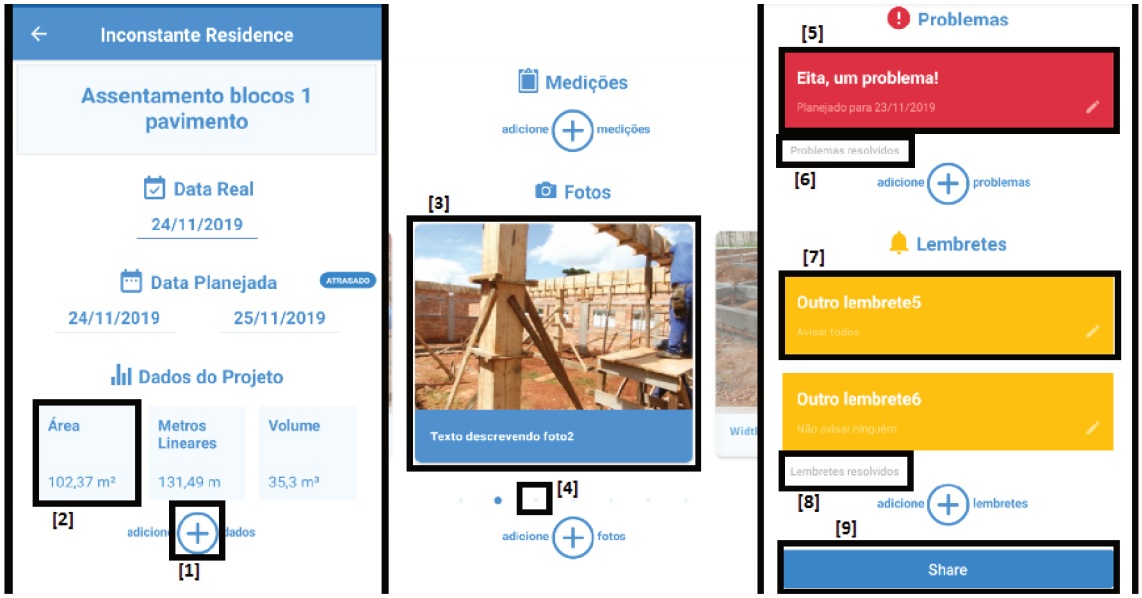

Figura 6 – Tela Tarefa com marcações.

(este fluxo é explicado melhor na subseção 3.2.5). Ela tem como objetivo estimular o usuário a identificar quais tarefas, ainda não iniciadas, do *sprint* atual, estão sendo iniciadas naquele dia. Esta tela possui apenas uma lista com todas as tarefas do *sprint* atual que ainda não foram iniciadas e dois botões, um de pular/cancelar, e outro de concluído, que identifica que todas as tarefas que estão começando hoje, já foram selecionadas. Além de um campo de busca para buscar as tarefas por nome.

- Tela de *Sprints*: Tela que lista os *sprints* existentes para aquela obra em ordem do mais antigo para o mais atual, além de permitir ao usuário a criação de um novo *sprint*. É no momento de criação do *sprint* que o usuário identifica quais tarefas

Figura 7 – Tela Problemas.

terão quais informações obrigatórias, conforme definição da seção 3.1.1.

3.2.2 Projeto

Uma das preocupações do aplicativo móvel é se tornar a ferramenta de bolso dos mestres de obras e dos engenheiros civis de execução que atuam mais no campo. Por isso, foi criada a seção Projeto, que garante, a quem está no canteiro de obra, um rápido acesso a arquivos dos mais diversos tipos que estão envolvidos na execução de uma obra, ou seja, acesso a plantas ou até modelos no conceito BIM. O usuário do aplicativo consegue consultar uma informação mais simples ou, até mesmo, usando as ferramentas de medição, extrair uma área ou uma metragem de determinado modelo.

Os arquivos ao qual essa seção dá acesso devem ser anteriormente importados na versão *web* do sistema (seção 3.3) e, como foi usada a API Forge da Autodesk (subseção 3.4.2.2), são suportados arquivos nos formatos: 3dm, 3ds, a, asm, brd, catpart, catproduct, cgr, collaboration, dae , dgn, dlv3, dwf, dwfx, dwg, dwt, dxf, emode, exp, f3d, fbx, g, gbxml, glb, gltf, iam, idw, ifc, ige, iges, igs ipt, iww, jt, max, model, mpf, msr, neu, nwc, nwd, obj, pdf, pmlprj, pmlprjz, prt, psmodel, rcp, rvt, sab, sat, sch, session, skp, sldasm, sldprt, ste, step, stl, stla, stlb, stp, stpz, vue, wire, xas, xpr.

Essa seção possui apenas duas telas (Figura 11), sendo uma delas (a da esquerda) apenas uma lista com todos os arquivos disponíveis para a obra em questão, onde é possível fazer uma busca pelo nome do arquivo ou selecionar um dos arquivos para ser aberto na segunda tela, que é, de fato, o visualizador de arquivos do aplicativo (a da direita).

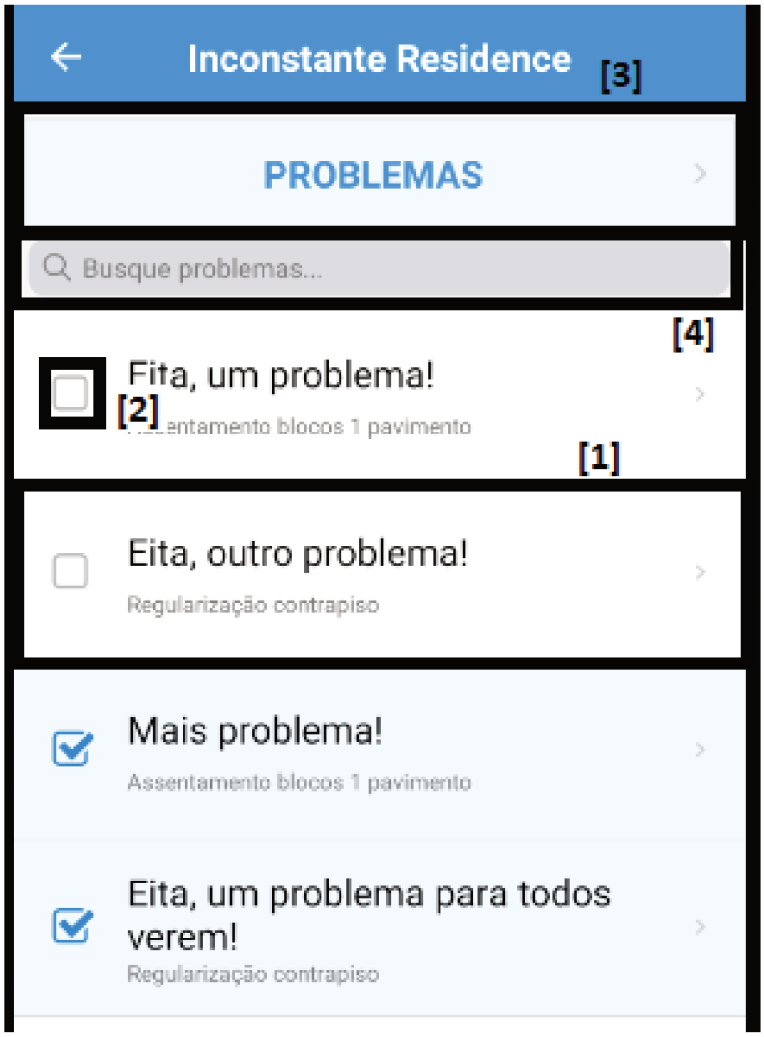

Figura 8 – Tela Problemas com marcações.

3.2.3 Relatórios

A seção Relatórios foi criada no aplicativo para que os diversos interessados em uma obra tenham um rápido acesso a informações condensadas, sem ter que entrar em cada uma das tarefas para extraí-las. Abaixo é listado cada um dos relatórios e quais as informações contidas:

- Relatório diário de obra: Dado um dia selecionado, identifica quais eram as condições climáticas naquele dia e reporta todas as mudanças consideradas relevantes naquele dia em qualquer atividade, são elas:
  - Conclusão de alguma atividade
  - Adição de foto em alguma atividade
  - Adição de alguma medição dentro de uma atividade
  - Adição de algum problema dentro de uma atividade

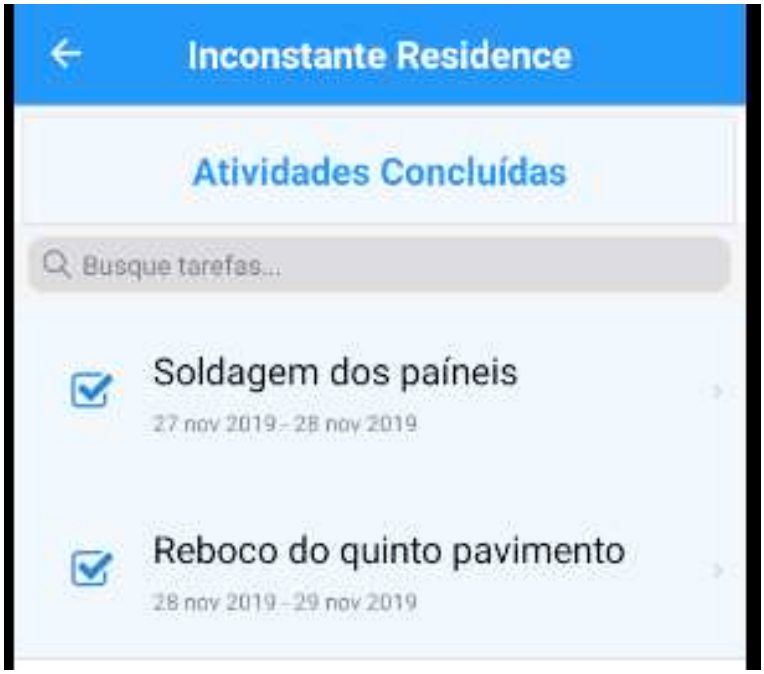

Figura 9 – Tela de Tarefas concluídas.

  - Conclusão de algum problema dentro de uma atividade
  - Conclusão de algum lembrete dentro de uma atividade

- Relatório de desvio da obra: Através de um gráfico de linhas onde uma linha representa o cronograma planejado da obra e a outra representa o cronograma executado até o momento, tendo a projeção das atividades que ainda não foram executadas seguindo o tempo de execução do planejamento, permite comparar o planejado contra o que está sendo executado, e se a obra deve ou não acabar dentro do prazo.

As telas da seção 'Relatórios' se assemelham muito as telas da seção 'Projeto', a diferença é que ao invés da listagem ser de arquivos disponíveis, a primeira tela possui uma lista de tipos de relatórios disponíveis onde é possível clicar em cada um deles, e clicando, o usuário é levado a segunda tela dessa seção que é um visualizador de relatórios, com o relatório selecionado aberto.

3.2.4 Perfil

A seção Perfil possui apenas uma tela, conforme pode ser visto na Figura 11, e serve para que o usuário possa fazer algumas configurações como troca de *e-mail* e telefone vinculados ao usuário logado. E ainda, caso o usuário possua mais de uma obra atrelada a esta conta, fazer a troca de obras. Ou seja, caso ele esteja na obra X, clicando no nome desta obra X surge uma lista de outras obras atreladas àquele perfil, onde ele, selecionando outra obra, altera todas as outras seções do aplicativo, que passam a ter seu conteúdo referente a nova obra selecionada, ao invés da anterior.

Figura 10 – Tela de início de tarefas.

Além disso, é nessa seção que o usuário encontra o botão de sair, importante para desvincular o *token* de segurança JWT criado no momento do *login* e passado para este celular. O *token*, enquanto estiver vinculado a um celular, dará acesso ao sistema, para esse aparelho, sem que seja necessário um novo login. Portanto, garantir que o usuário faça o *logout* é especialmente relevante quando o uso do sistema é feito em um aparelho compartilhado com outras pessoas, já que, caso o *logout* não seja feito, outras pessoas podem alterar o conteúdo das tarefas usando uma conta que não as pertence.

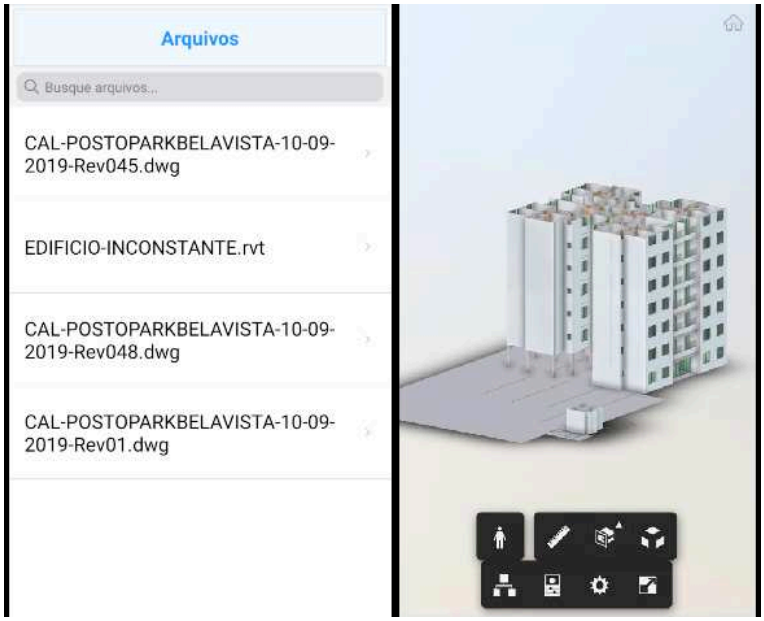

Figura 11 – Telas da seção projeto.

3.2.5 Telas sem seção

Para que nenhuma das telas ficasse ser sem contemplada nesse trabalho, criou-se esta subseção que trata das duas telas que ficam fora de toda a lógica de seções proposta pelo aplicativo móvel. O não pertencimento destas telas a nenhuma das seções explicadas nas subseções anteriores ocorre devido ao fato de antecederem o uso do aplicativo em si.

Essas duas telas sem seção, estão ambas atreladas ao momento do usuário se identificar no aplicativo, isso é de extrema importância para que seja possível saber exatamente quem tomou cada uma das ações dentro do sistema, e para possibilitar a criação de uma hierarquização de usuários, na qual determinadas ações apenas usuários de nível mais elevado na hierarquia conseguem agir.

A tela à esquerda na Figura 13 é a tela de carregamento no aplicativo, e seu intuito é destacar visualmente para o usuário que o aplicativo está sendo aberto. Esta tela é importante, pois, antes que a primeira tela seja mostrada ao usuário é necessário fazer uma série de considerações e, na maioria das vezes, carregar dados, conforme mostra o fluxograma da Figura 14. Portanto, é mais agradável e "um padrão da indústria", esse primeiro carregamento acontecer com a logo da empresa em evidência, até mesmo para que a marca fique exposta, uma vez que ela aparecerá apenas nesse carregamento e na tela de *login*.

A tela à direita na Figura 11 é a tela de *login*, e nela o usuário irá fornecer seu *e-mail* e senha que serão passados para autenticação no servidor. Caso seja bem-sucedido, o usuário receberá um *token* de segurança JWT que permitirá que ele não precise passar por esse processo novamente, até que faça o *logout* na seção 'Perfil'. Uma vez com o *token*,

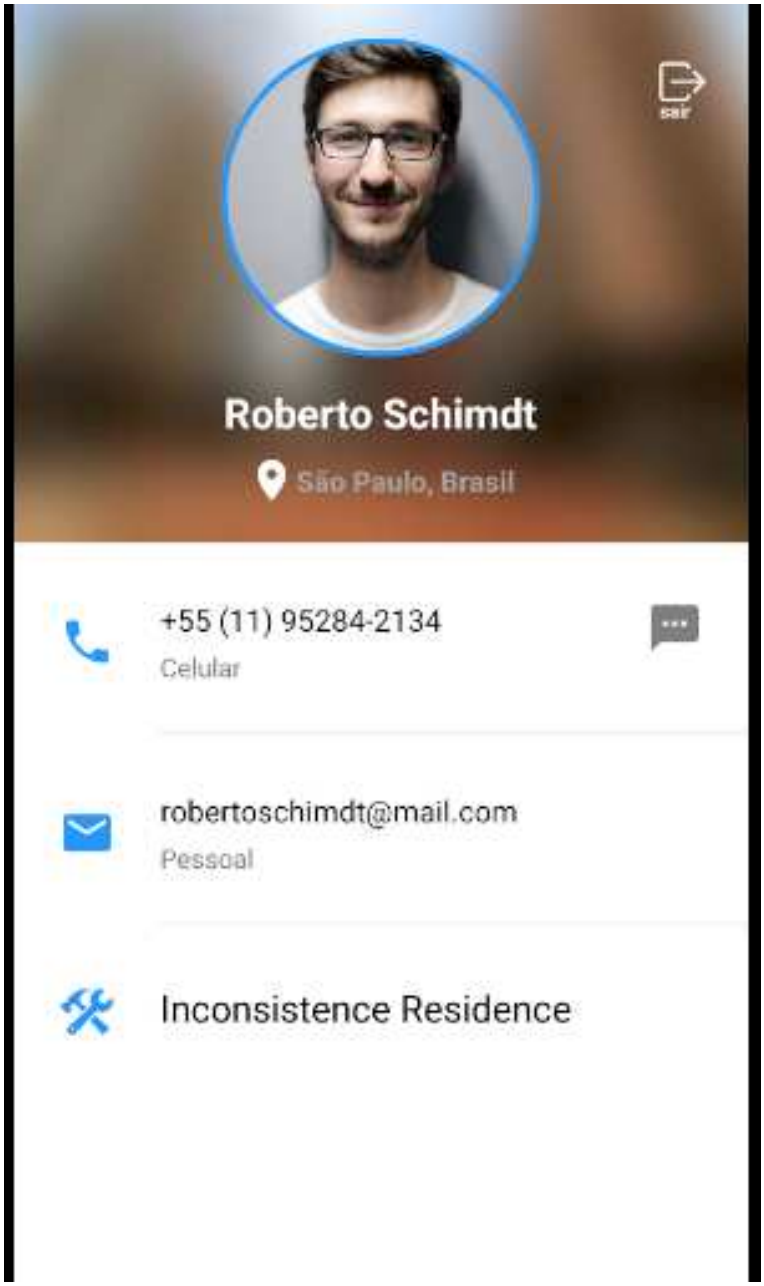

Figura 12 – Tela da seção perfil.

a tela de *login* segue para a bifurcação dois do fluxograma da Figura 14.

3.3 Telas da versão *Web*

A versão *web* do sistema proposto neste trabalho tem como principal papel ser a ferramenta pela qual o usuário faz o *setup* inicial de cada obra, uma vez que pode ser necessário uma série de configurações e *uploads* de arquivos que não gerariam uma experiência de usuário agradável se realizadas numa versão móvel. Ou seja, o aplicativo para celulares é a parte principal do sistema proposto, uma vez que será uma ferramenta de uso diário, sendo a versão *web* importante apenas no inicio da obra, para que sejam

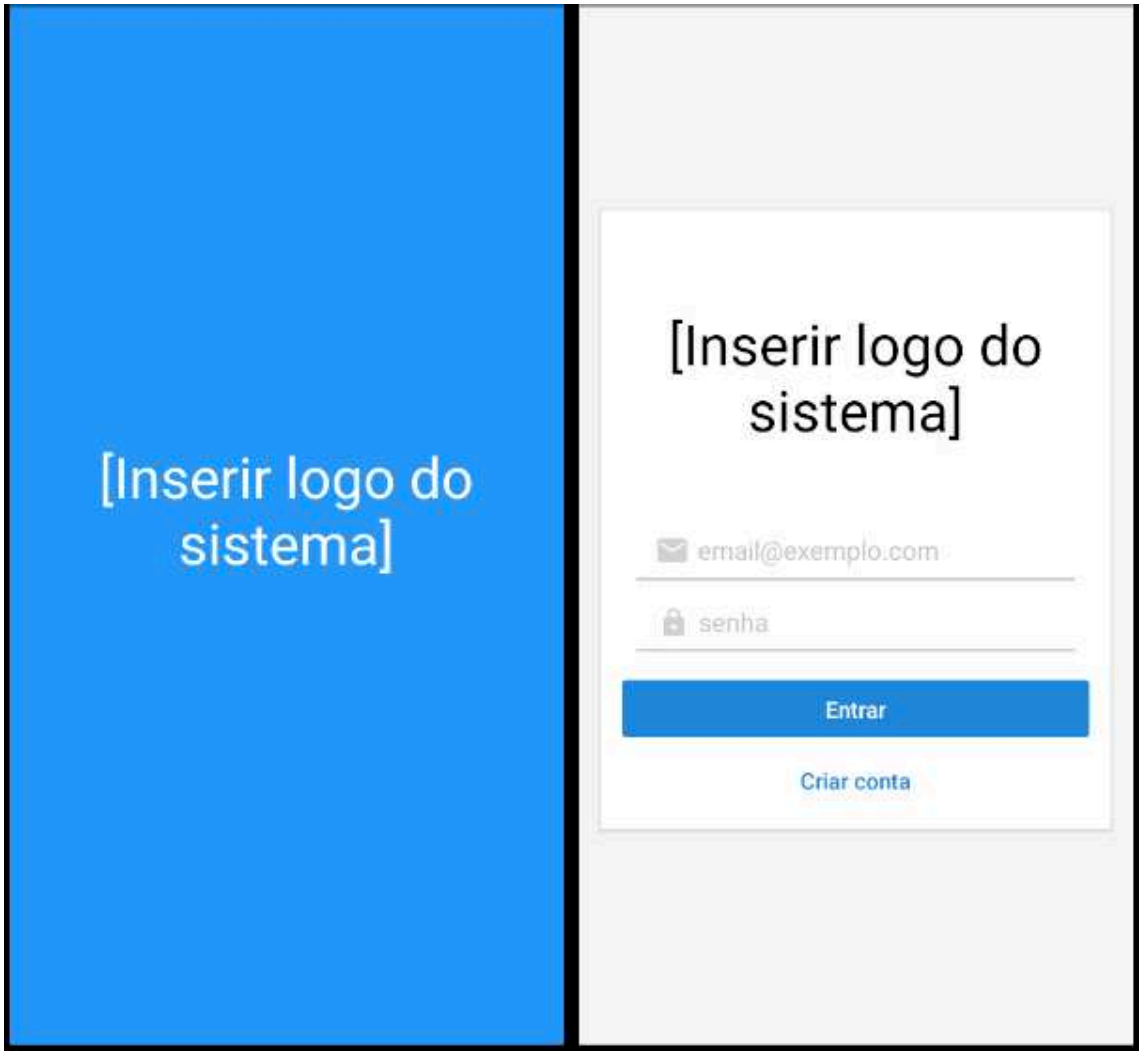

Figura 13 – Telas sem seção. A esquerda a tela de carregamento do aplicativo. A direita a tela de *Login*.

feitas algumas configurações.

Na versão *Web* o usuário pode criar e remover obras. Quando uma obra é criada o usuário tem a opção de, após a inserção dos dados mencionados na subseção 3.1.3, incluir um arquivo feito no MSProject com uma listagem de atividades da obra, seus vínculos de dependência e datas de começo e fim, ou, ainda, ele pode criar as tarefas manualmente uma-a-uma nesta versão do sistema.

Além disso, é na versão *web* que são incluídos arquivos a serem abertos no visualizador de cada uma de suas obras. Para realizar este tipo de ação, basta que o usuário esteja na tela de *upload* de arquivos e arraste o arquivo para dentro do seu navegador ou selecione o arquivo navegando até onde o mesmo se encontra dentro do seu computador.

Também é nesta versão do sistema que o usuário faz a criação de formulários para a qualidade, conforme mencionado na subseção 3.1.1.

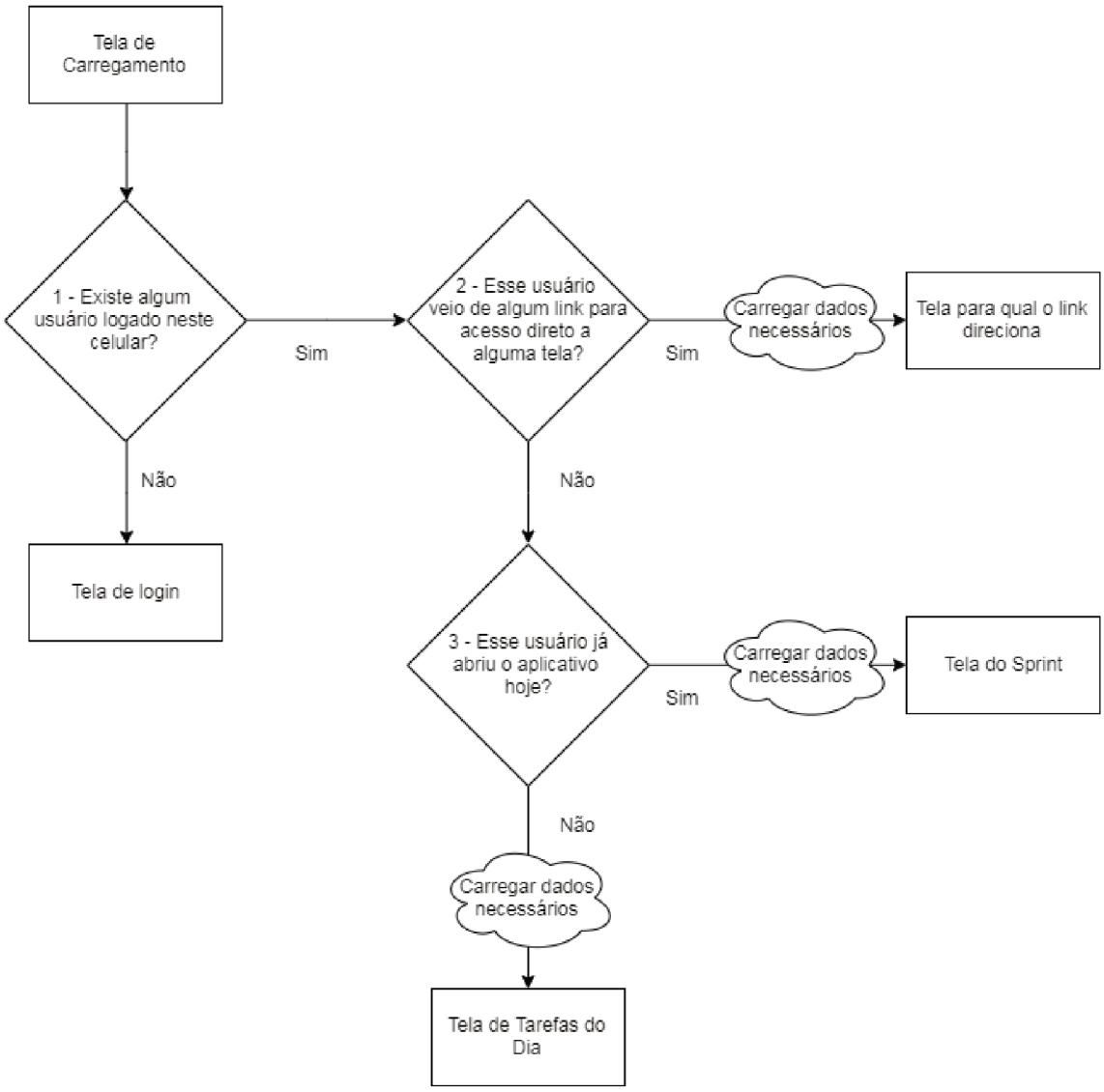

Figura 14 – Fluxograma da tela de carregamento.

3.4 Desenvolvimento do aplicativo

É importante fazer a separação do desenvolvimento da aplicação proposta em duas partes, igualmente trabalhosas, uma sendo a modelagem, que envolve identificar como serão separadas as funcionalidades do app e da versão *Web* em telas, onde e como posicionar cada um dos elementos dessas telas, enfim, toda a parte de *design* e experiência do usuário para as telas é abordado na subseção 3.4.1 deste trabalho.

A outra parte, também muito importante, pois é ela que transforma, de fato o sistema em uma aplicação capaz de ser usada pelo usuário para efetuar todas as funcionalidades até aqui especificadas, é a programação. É neste ponto, abordado na subseção 3.4.2, que se juntam todas as telas e fluxos do protótipo às regras de negócio definidas durante a concepção.

3.4.1 Modelagem

A modelagem desta aplicação, assim como a criação de vários dos conceitos utilizados foi feita em parceria com o aluno Benjamim Bueno, do bacharelado em Engenharia Civil da Universidade Federal de Uberlândia. A modelagem das telas deu origem ao estudo inicial da presente aplicação, com o anseio de cumprir os objetivos desta proposta e aprimorar o uso de novas tecnologias nos canteiros de obra.

As telas de modelagem tanto da versão móvel (Figura 15 e 16) quanto da versão *web* (Figura 17) foram concebidas usando um conceito mais *clean*, em que apenas os dados necessários são apresentados para o usuário a cada tela, buscando algo semelhante às interfaces criadas usando Material Design, linguagem de design criada pela empresa Google para desenvolvimento de aplicações.

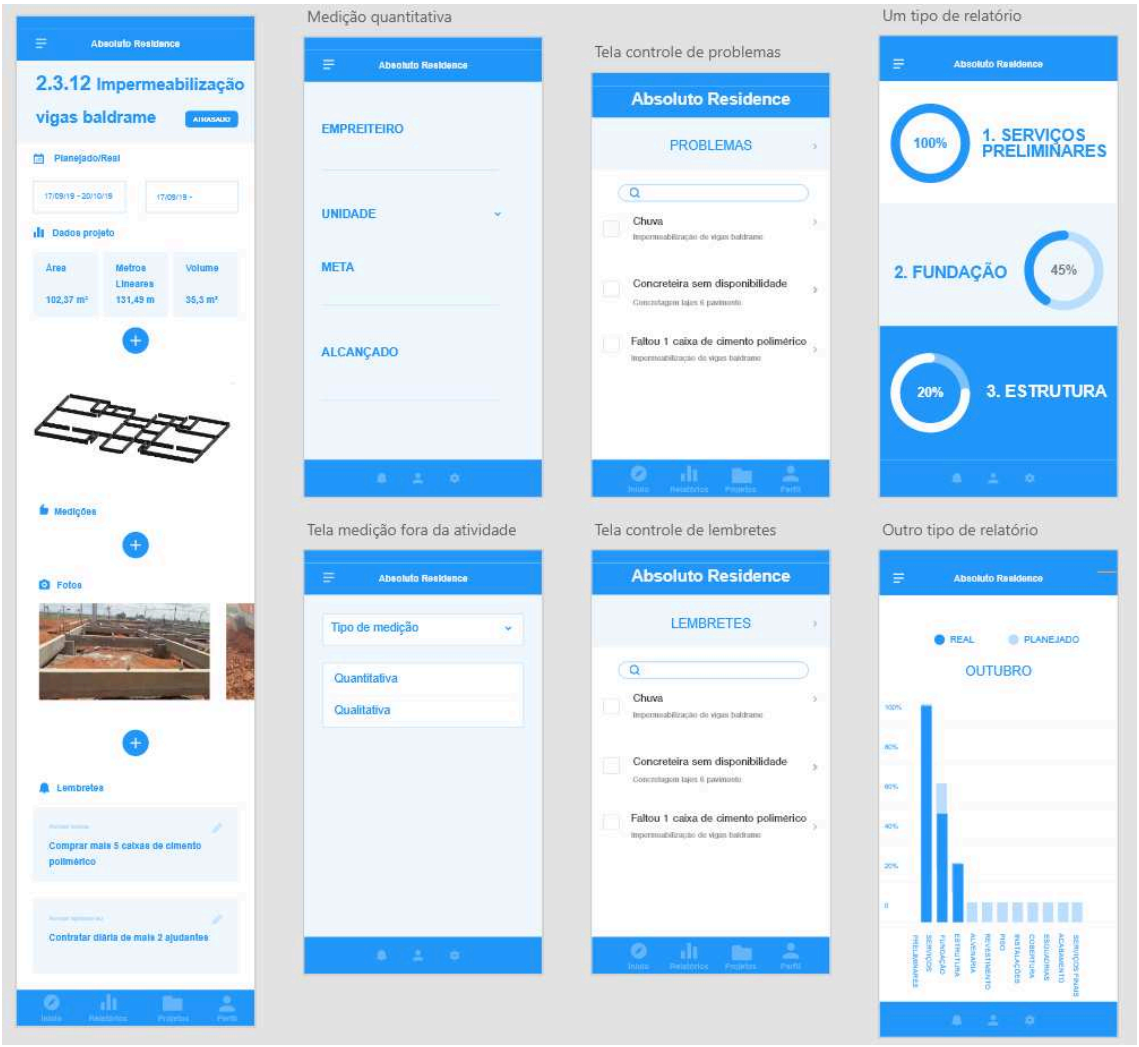

Figura 15 – Telas do protótipo *mobile*.(1/2)

Além das telas, algo que foi realizado ainda durante a modelagem para a versão *mobile* foi uma versão protótipo, que auxiliou trazendo algo palpável de como seria a navegação dentro do aplicativo, ou seja, um vínculo entre as telas feitas no design, conforme

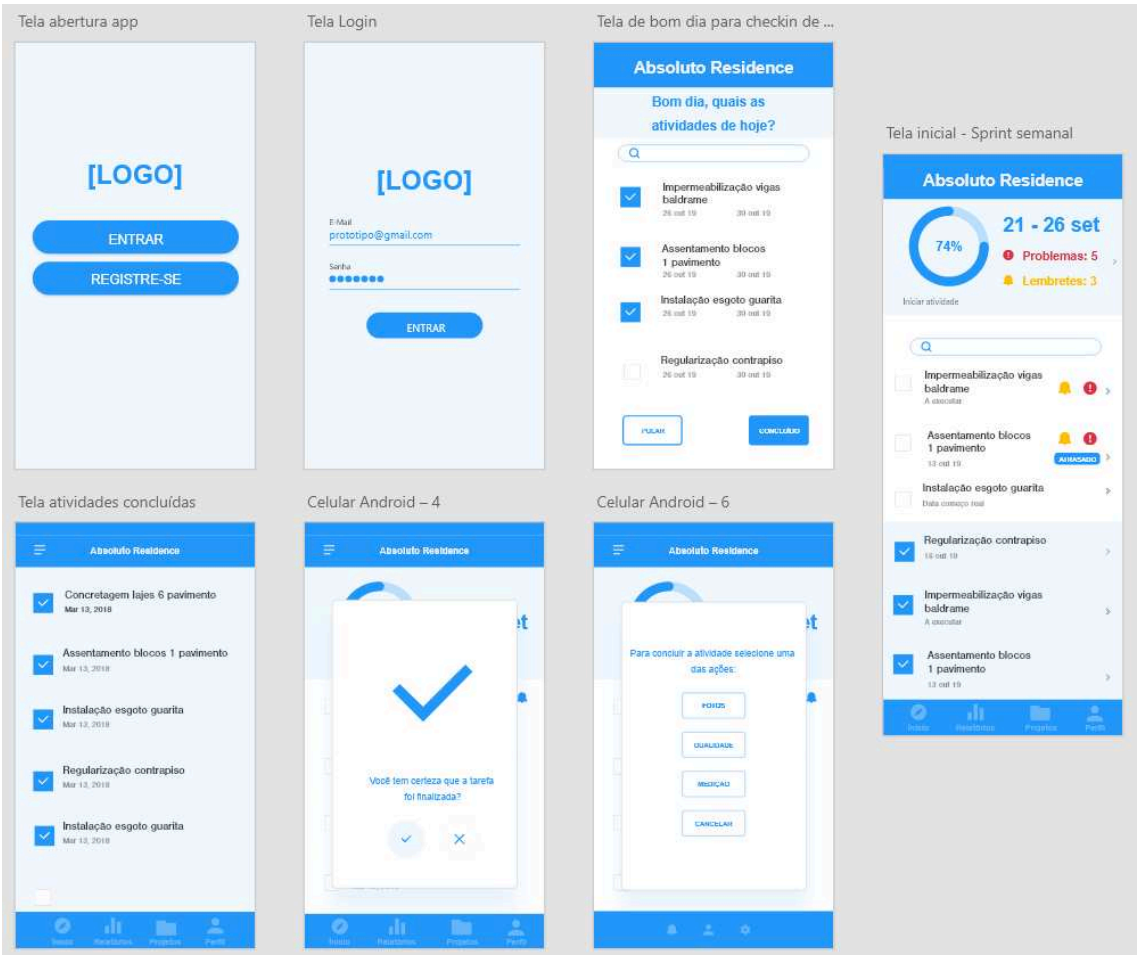

Figura 16 – Telas do protótipo *mobile*.(2/2)

demonstra a Figura 18, em que fosse possível já se ter uma primeira impressão de qual seria a experiência do usuário.

Essa fase de modelagem ocorreu toda dentro da ferramente Adobe XD e ela foi essencial para acelerar a parte de programação da aplicação, uma vez que, apesar de ainda deixar brechas, garante que grande parte das decisões visuais já estejam tomadas na hora da codificação, deixando como responsabilidade do desenvolvedor construir algo o mais próximo possível daquilo que está feito no protótipo.

3.4.2 Programação

3.4.2.1 *Frameworks* Utilizados

Como esse projeto envolve a criação de um aplicativo móvel que tem de acessar funcionalidades internas simples de um celular, como a possibilidade de se tirar fotos e de gerar notificação para os usuários do app, a rota escolhida foi o desenvolvimento através de um *framework mobile*. Para entender o motivo dessa escolha é importante entender quais eram as outras opções e quais suas vantagens e desvantagens.

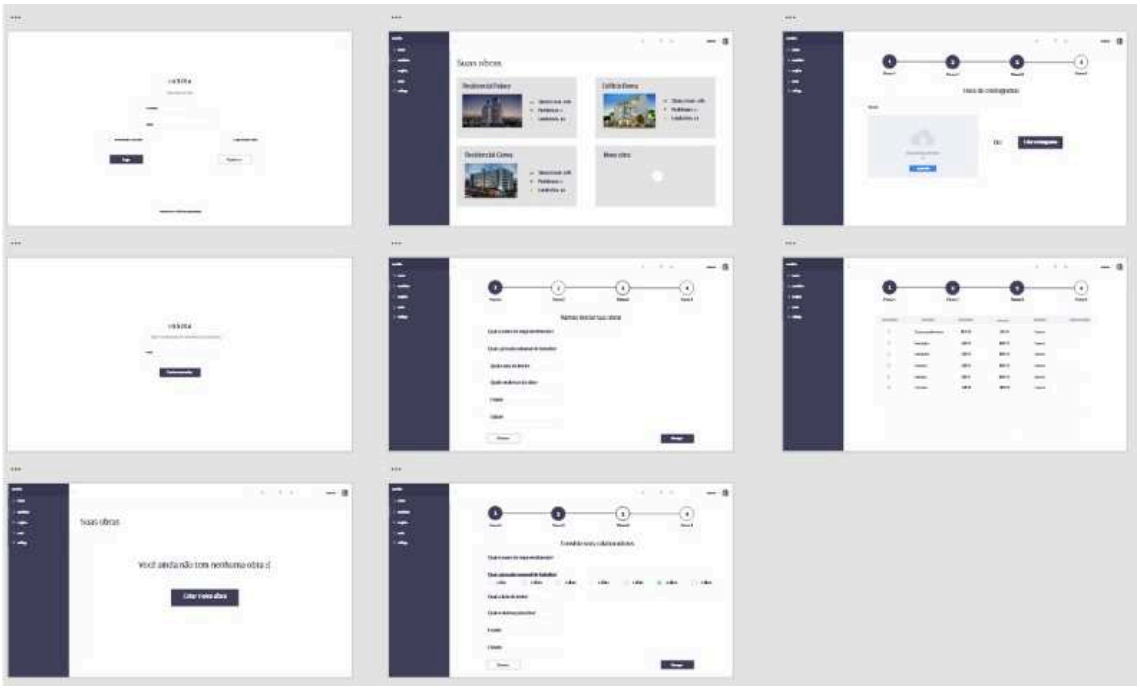

Figura 17 – Telas do protótipo *web*.

A opção mais óbvia seria desenvolver o aplicativo especificamente para cada plataforma onde ele deve funcionar, ou seja, Android e iOS, já que conforme o StatCounter[1] estes dois sistemas operacionais estão em 98,72% dos celulares do mundo. Com isso tem-se um código em Objective-C ou Swift para iOS, sistema operacional dos aparelhos da empresa Apple e outro código em Java ou Kotlin para os aparelhos que funcionam com o sistema operacional Android. Essa opção oferece a vantagem ao desenvolvedor de se ter acesso completo a tudo que é possível um aplicativo fazer dentro dos respectivos sistemas operacionais, ou seja, é o maior nível de liberdade possível quando se fala em programação de um aplicativo móvel. Mas ela carrega com si dois grandes desafios, ter o domínio de, ao menos, duas linguagens distintas e disponibilidade para manutenção de dois códigos diferentes. Ou seja, qualquer alteração feita numa versão do aplicativo que funciona em uma das plataformas, deve ser programa para que ocorra na outra. Isso pode ser positivo em casos de um *bug* que esteja acontecendo especificamente em uma versão especifica de um dos celulares, mas de maneira geral, gera trabalho dobrado.

Outra opção, para a versão móvel, é a criação de um aplicativo *web* que seria possível ser acessado de um navegador de qualquer dispositivo móvel. Essa opção traz a facilidade de ser necessário apenas um código para qualquer plataforma que tenha acesso a um navegador, mas é inviável para as necessidades da aplicação descrita aqui, uma vez que com uma aplicação rodando em um navegador não é possível ter acesso a funcionalidades do celular como câmera fotográfica ou o envio de notificações, funcionalidades que são essenciais para o funcionamento do aplicativo.

[1] https://gs.statcounter.com/os-market-share/mobile/worldwide

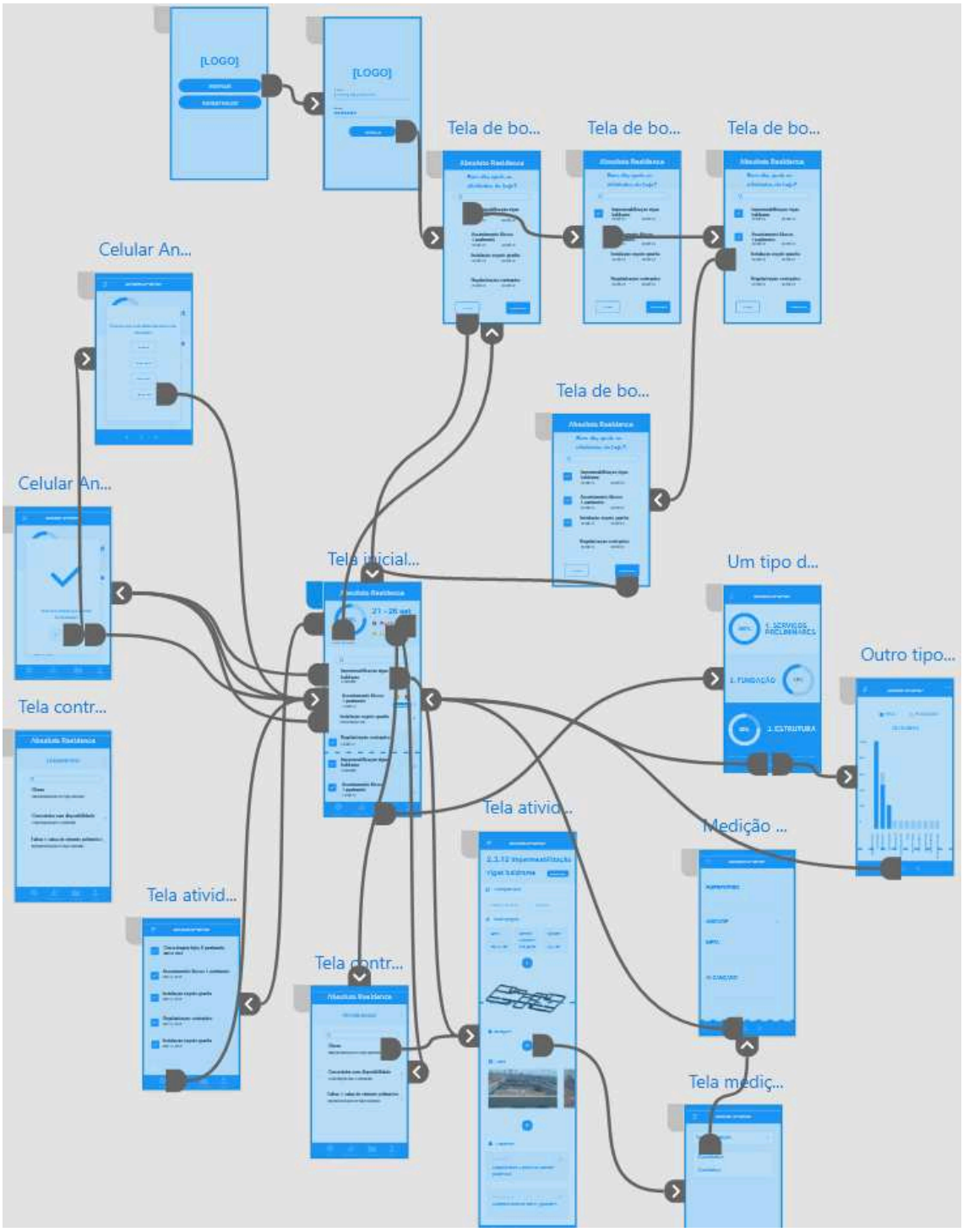

Figura 18 – Telas vinculadas no protótipo.

Portanto, a opção de codificação do aplicativo em um *framework mobile*, como React Native, Cordova, PhoneGap, Ionic e outros, garante ao desenvolvedor, de maneira geral, a manutenção de apenas um código para ambos os sistemas operacionais supracitados, mas, diferente da opção *web*, fornece o acesso a funcionalidades internas do celular, algo que já foi identificado como essencial para este aplicativo.

Dada a decisão de se desenvolver em um *framework mobile*, era necessário decidir

em qual desses, exatamente, a aplicação seria feita. A escolha foi pelo React Native, que traz como seu grande diferencial, em relação aos outros *frameworks*, a possibilidade do desenvolvedor, caso achar que seja necessário, criar trechos de códigos específicos para Android ou iOS para acessar alguma funcionalidade nativa do celular a qual o React Native possa não dar acesso. Entende-se que inicialmente essa demanda não existe, mas como possibilidade de deixar esse projeto preparado para aprimorações futuras, o escritor deste trabalho tomou essa decisão, além do que, esse diferencial do React Native não é criado as custas de nenhuma desvantagem, uma vez que deixando esse ponto de lado, a decisão entre qualquer outro *framework mobile* ocorre por questões pessoais e não técnicas. (AVDIC, 2019)

3.4.2.2 APIs Externas Utilizadas

Como um dos objetivos do aplicativo era que a seção Projeto oferecesse a possibilidade do usuário poder acessar seus arquivos da palma da sua mão e de maneira prática, optou-se pelo uso de uma API externa de uma empresa referência no setor de tecnologia para construção civil, Autodesk, que além de fornecer a API de um visualizador (AUTODESK, 2019b), fornece também uma API de tradução de arquivos (AUTODESK, 2019a) para o formato lido pelo seu visualizador. O que garante a compatibilidade com uma enorme gama de tipos de arquivos.

O uso da API não foi exatamente trivial, e um único trabalho abordando React Native e estas APIs encontrado foi o de um *blog* de funcionários da própria Autodesk[2], que usava versões ultrapassadas do React Native e da própria API da Autodesk. Porém, depois de entender alterações que haviam ocorrido nas versões mais recentes do *Viewer* (AUTODESK, 2019c), que requereram uma série de alterações no código e ainda o uso de *cookies*, bastou, então, entender como desligar as funcionalidades atreladas aos *cookies* para que as versões mais recentes da API funcionassem nos aparelhos com Android, tendo em vista a dificuldade no funcionamento dos *cookies* numa *WebView* do Android. Já no iPhone, foi necessário setar que o tipo de navegador que deveria ser usado dentro da *WebView* seria o *WKWebView*, ao invés do padrão do React Native que ainda usa, no iPhone, o *UIWebView*, que causa conflitos com a API e já não é mais recomendado pela Apple (REACTNATIVE, 2019b).

O código do visualizador para funcionamento em ambas plataformas usando React Native está no Apêndice A deste trabalho.

3.4.2.3 Padrões de projeto

Com o uso do *framework mobile* React Native, conforme explicado na seção 3.4.2.1, o aplicativo baseia-se na arquitetura de projeto *Model View Controller (MVC)*, com al-

[2] https://forge.autodesk.com/blog/forge-react-native-au-talk

gumas adaptações que o React traz em si.

No React, o MVC é um pouco diferente, uma vez que ao invés de se ter o *View* e o *Controller* separados, tem-se uma entidade, chamada Componente, que recebe tanto a tarefa de *View* quanto a de *Controller*, podendo ficar a cargo do programador a tentativa de desacoplar os dois o máximo possível com a criação de *Higher Order Components*, que são componentes que fazem o papel de *Controller* de vários outros componentes que cuidarão apenas da renderização de conteúdo, ou seja, o papel da *View*. Mas, ainda nesses casos, é necessário que os *Higher Order Components* identifiquem de quais componentes visualmente eles serão compostos, o que faz com que o React, mesmo num código onde todo o controle é feito em *Higher Order Components*, implemente um MVC não "puro".

A parte do *Model* do MVC no React pode sim ser implementada da forma que essa arquitetura sugere. Porém, a medida que aumenta a complexidade do aplicativo e sua quantidade de *Views* e *Models*, os vínculos entre um e outro escalam de tal forma, que existe o risco de se chegar em um *loop* infinito, onde uma *View* atualizou um *Model*, que gera uma atualização de uma outra *View*, que por sua vez atualiza um outro *Model*, que por fim atualiza aquela primeira *View*. O problema não é só a existência dos *loops*, que podem ser retirados uma vez encontrados, o problema é, exatamente, encontrá-los, o que acontece muitas vezes é que a quantidade de *Views* e *Models* e vínculo entre eles é tamanha que essa tarefa se torna muito difícil (Figura 19).

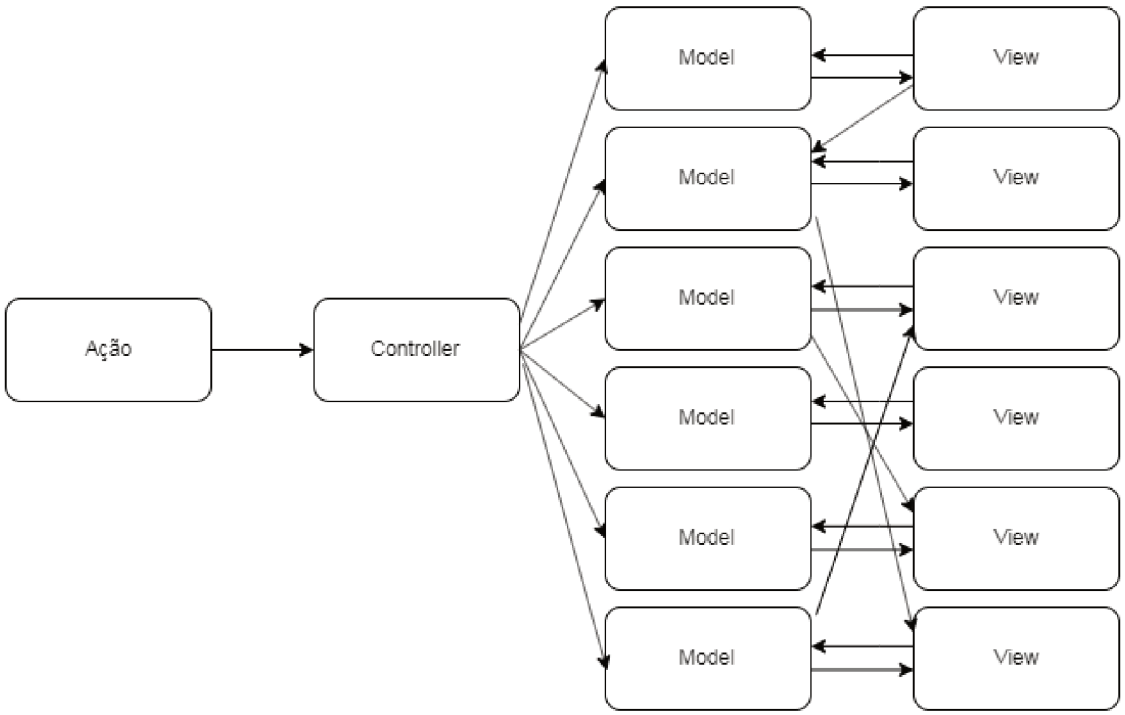

Figura 19 – Múltiplos *Views* e *Models* e seus vínculos.

A fim de evitar esse problema, garantindo assim uma maior facilidade na adição de novas funcionalidades futuras no aplicativo, optou-se neste trabalho por implementar

uma arquitetura chamada Flux, arquitetura proposta pelo Facebook onde o fluxo de dados acontece de maneira unidirecional. Conforme o diagrama da Figura 20, no Flux todas as ações chegam para um *Dispatcher*, que dispara quais ações devem ser tomadas pelas diversas *Stores*. As *Stores*, por sua vez, guardam os dados e, uma vez recebida uma ação do *Dispatcher*, identifica se essa ação deve afetar, de alguma forma, seu estado atual. Em caso positivo, ela notifica todos os Componentes (que fazem o papel de *Controller* e *View*, conforme último parágrafo) que se atualizam conforme são avisados pelas *Stores* importantes para cada um deles.

Dessa forma, garante-se um fluxo unidirecional de dados, com todo o estado dos dados da aplicação mantidos nas *Stores*, fazendo com que não haja nem inconsistência entre uma *View* e outra, e ficando longe dos *loops* infinitos causados pelos vínculos de diferentes *Views* e *Models*, uma vez que todos os *Models* conversam entre si, gerenciados dentro do *Dispatcher*, para depois serem terem suas modificações disparadas para os componentes(*Views* e *Controllers*).

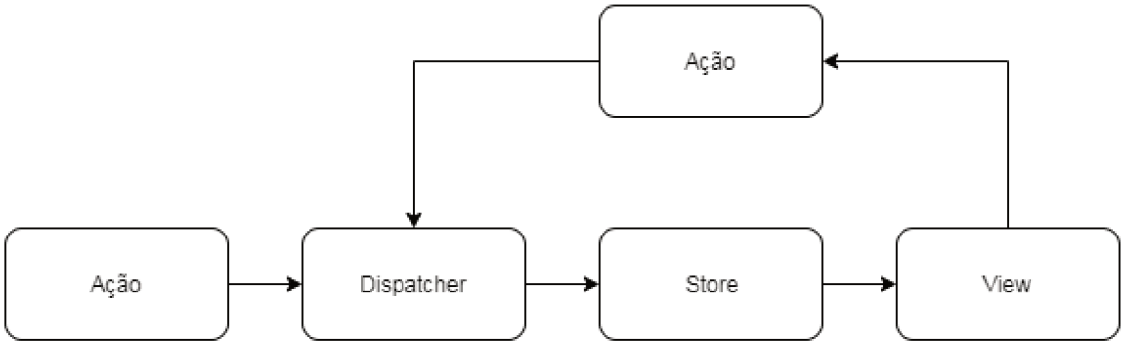

Figura 20 – Diagrama da arquitetura Flux.

Para implementação do Flux nesse projeto, a biblioteca Redux foi usada, ela é uma implementação da arquitetura Flux que tem algumas diferenças do conceito inicial.

A primeira grande diferença é que no Redux, diferentemente do Flux, em que a ideia é se ter diferentes *Stores*, tem-se apenas uma, todo o estado dos dados da aplicação, com Redux, é guardado em um objeto *Store*. Isso torna possível entender a segunda diferença do Redux, nele o *Dispatcher* não é uma entidade separada da *Store*, ele está dentro da *Store* e dispara as ações que chegam até ele para os *Reducers*. Os *Reducers* são quem manipulam, de fato, o estado dos dados na *Store*, eles vão receber o estado anterior e a ação disparada pelo *Dispatcher*, e, dado esses dois dados, vão entregar um novo estado dos dados para a *Store*. A *Store* por sua vez, da mesma forma que ocorre no Flux, avisa as *Views* do novo estado dos dados. Essa nova forma de fazer o fluxo unidirecional dos dados está mostrada no fluxograma da Figura 21.

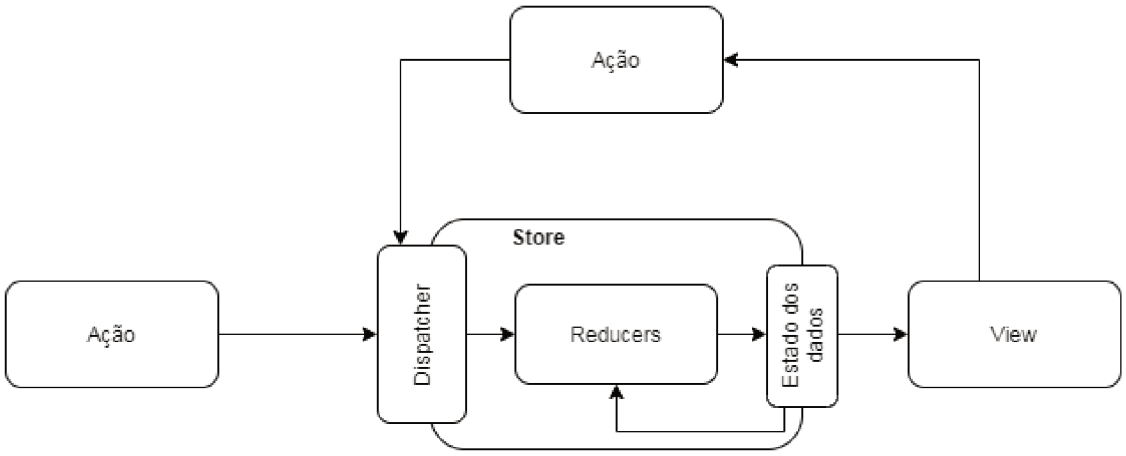

Figura 21 – Diagrama de funcionamento da biblioteca Redux.

3.4.2.4 *Back-end* da Aplicação

Por se tratar de uma aplicação móvel feita no *framework* React Native no qual o acesso ao *back-end* seria apenas para leitura e escrita dos dados, optou-se por usar um *back-end* também em JavaScript, afim de garantir uma fácil manutenção de código, já que toda a pilha de desenvolvimento seria em JavaScript.

Através do uso do node.js foi construída uma API de acesso ao banco de dados, ao qual o aplicativo faz o *login* a cada acesso de um usuário sem um JWT e repassa os dados relevantes, em sua última versão, àquele usuário.

A API, construída sob o protocolo HTTP, tem uma estrutura simples, onde para ter acesso a qualquer dado ou fazer alguma alteração nestes, a aplicação deve passar um JWT via cabeçalho da requisição. Esse *token* tem sua validade conferida, e caso válido, a aplicação pode realizar a requisição solicitada.

Para resgatar os dados a aplicação deve enviar, além do *token* no cabeçalho, via HTTP, um método GET, passando, na própria URL, um dos seguintes termos:

- *sprints*
- tarefas
- lembretes
- problemas
- obras
- usuários
- arquivos
- relatórios

Caso não queira receber uma lista com todos os dados para aquele termo aos quais o *token* passado garante o acesso, a aplicação tem a opção de passar um identificador único especifico de algum dado daquela lista que esteja buscando. Dessa forma a API retornará apenas aquele dado, caso o *token* enviado garanta o acesso a ele e exista de fato um dado com aquele identificador.

Além de dar acesso aos dados no banco, a API garante a possibilidade de alterá-los. Para cada ação no aplicativo que altera um dado, existe um termo equivalente a ser colocado na URL para dispará-la, como 'concluirlembrete', 'concluirproblema', 'comecar-tarefas', entre outros, todos com nomes bem intuitivos quanto a qual funcionalidade do app eles correspondem. Para esse tipo de solicitação o método POST deve ser usado, também deve ser enviado o dado necessário para aquela alteração no "corpo"da requisição, que também ocorre via HTTP. No caso das alterações mencionadas acima, por exemplo, a aplicação tem de enviar, respectivamente, o identificador único do lembrete que está sendo concluído, o identificador único do problema que está sendo concluído, um *array* com os identificadores únicos das tarefas que estão sendo iniciadas.

O banco de dados usado da aplicação é o MongoDB e foi escolhido pelo fato de toda a pilha de desenvolvimento trabalhar com JavaScript. Assim, um banco feito para se trabalhar com objetos JSONs agilizaria o desenvolvimento da aplicação, que também não possui consultas tão complexas, o que poderia ameaçar o benefício da agilidade no uso de banco um banco NoSQL. Na Figura 22 pode-se encontrar a estruturação dos dados proposta para este trabalho, na forma de documentos, modelo de dados usado pelo MongoDB.

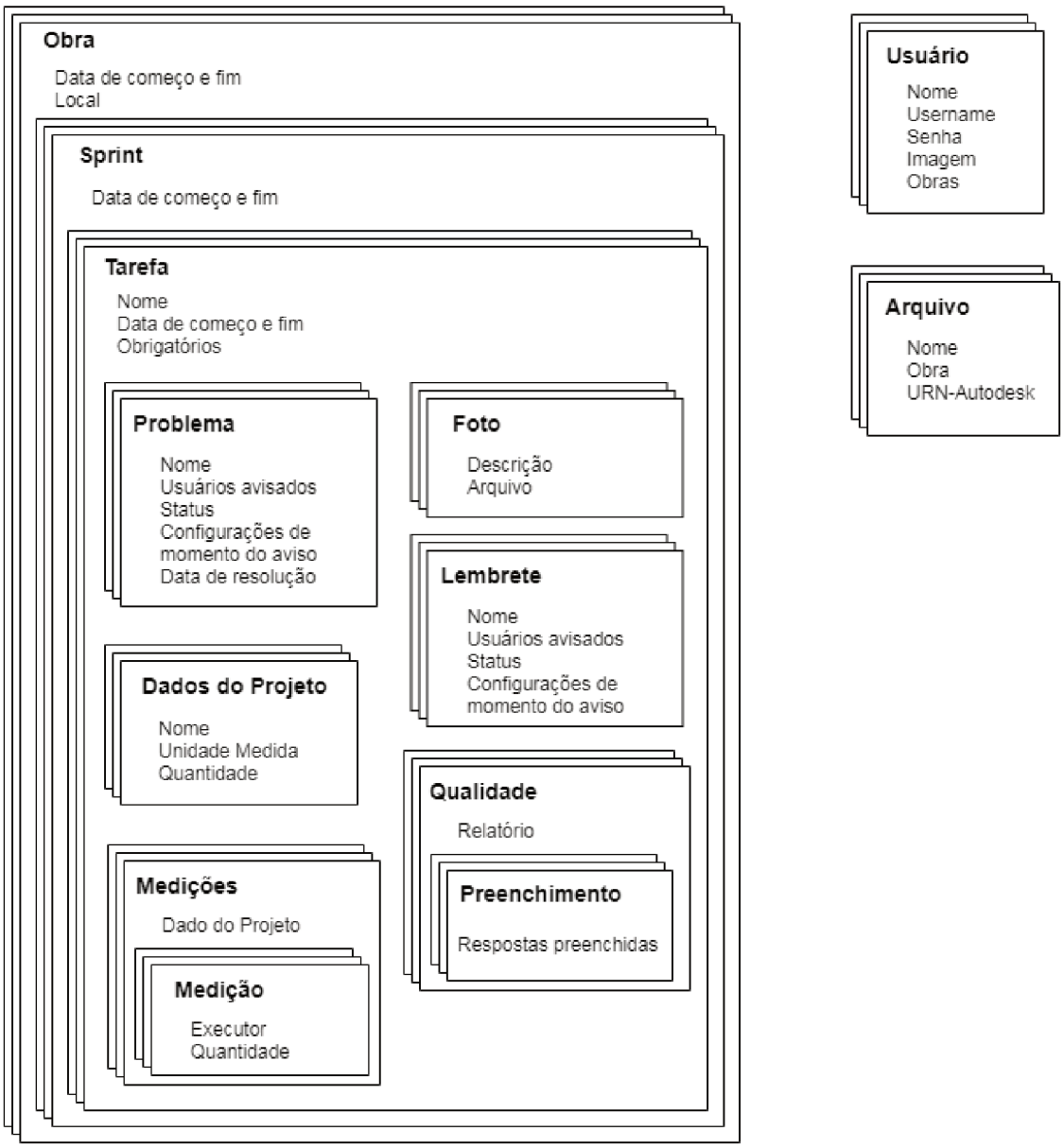

Figura 22 – Modelagem de dados proposta na forma de documentos.

4 Conclusão

O setor de construção civil brasileiro, mais especificamente, o canteiro de obra, carece muito de sistemas que o conheçam e busquem ajudar na sua dinâmica do dia-a-dia. Diversos sistemas já foram criados que facilitam o trabalho dos projetistas e até mesmo dos planejadores de obras. Mas o canteiro de obra, onde os profissionais costumam ter um menor nível de instrução educacional é o que mais fica longe dos avanços tecnológicos. Portanto, se enxerga a conceituação, modelagem e desenvolvimento desse sistema como um passo importante no aumento de produtividade deste meio.

A construção deste sistema para que o usuário consiga interagir de maneira intuitiva e na forma de um aplicativo para aparelhos móveis, entende-se como sendo um importante passo para sua viabilidade no dia-a-dia, uma vez que independentemente de condição financeira e escolaridade, a maioria da população brasileira tem um *smartphone* e seria capaz de interagir com o app. (MEIRELLES, 2019)

Outro ponto positivo demonstrado durante a confecção desse trabalho é que é possível o desenvolvimento de um aplicativo, que não tenha necessidades muito pesadas de acesso ao *hardware* do celular, em um *framework mobile* e, ainda, que é possível fazê-lo para que o mesmo mantenha semelhanças visuais importantes com aplicativos desenvolvidos com código nativo do seu sistema operacional.

De forma a criar um comparativo entre a solução apresentada neste trabalho com outras presentes no mercado, foram identificadas as seguintes diferenças:

- ConstructApp[1]: Tem uma gestão baseada em tarefas, assim como a proposta na solução deste trabalho, porém não faz o uso do conceito de *sprint*, nem de problemas e lembretes, apresentados neste trabalho, nem algo semelhante que crie o senso de urgência e ajude o gestor a não esquecer de tarefas importantes do dia-a-dia da obra.
- Diário de Obra[2]: Tem como objetivo apenas substituir o diário de obra feito no papel, assim como um dos relatórios propostos na subseção 3.2.3 deste trabalho, mas não oferece nenhuma outra funcionalidade para apoiar o administrador da obra em sua rotina, assim como o Construct.
- Procore[3] e BIM 360[4]: São duas ótimas soluções e com mais funcionalidades do que as propostas por esse próprio trabalho, porém, além de não terem uma versão

[1] https://constructapp.io/pt/
[2] https://diariodeobras.net/
[3] https://www.procore.com/
[4] https://www.autodesk.com/bim-360/

em português, o que já inviabilizaria sua aplicação para funcionários com menos instrução educacional, são construídas para a realidade do canteiro de obras de outros países, que são diferentes da encontrada no Brasil.

4.1 Trabalhos Futuros

Entende-se que esse sistema é um passo inicial na informatização do canteiro de obras e, que através da sistematização do controle de tarefas, é possível avançar em diversas outras áreas que a computação já atua em outros segmentos, e que ainda não são explorados na construção civil. Abaixo algumas das possibilidade que o escritor desse trabalho enxerga como possíveis após a sistematização do controle de atividades num canteiro de obra:

- *Mineração de dados* para acompanhar a performance de cada empreiteiro, afim de entender qual empreiteiro produz mais e porque.
- Através da adição de gasto de materiais como um dado a ser preenchido no fim de cada tarefa, é possível realizar *mineração de dados* para entender o gasto de material em cada tarefa da obra, afim de otimizar orçamentos pré-obra. Ainda com a adição dessa funcionalidade, seria possível automatizar o processo de solicitação de materiais.
- Uma vez bem estabelecidos processos como medição e qualidade, via este sistema proposto, e uma obra bem mapeada no conceito BIM, é possível o uso de *drones* para se fazer os processos de medição e qualidade com maior exatidão, menos erros e sem dependência de *inputs* manuais.
- *Mineração de dados* para o entendimento de quais tarefas são dependentes entre si para o aperfeiçoamento automático e em tempo real de quais atividades devem ser as próximas a serem realizadas, com possibilidade de, vinculado a sistemas de previsão meteorológica e sistemas de controles financeiros, se adaptar as situações climáticas e a disponibilidade dos materiais na obra.
- Através de adaptações ao sistema para uso por parte dos empreiteiros, é possível realizar a automatização da delegação de tarefas, dando a possibilidade de um coordenador conseguir cuidar de mais frentes de trabalho concomitantemente.
- Através de um maior entendimento do funcionamento de arquivos no conceito BIM é possível atrelar cada tarefa a apenas objetos que estiverem no modelo que forem relevantes a ela, afim de facilitar a usabilidade desse tipo de modelagem no dia-a-dia da obra.

Referências

ADOBE. *Adobe XD*. 2019. Disponível em: <https://www.adobe.com/br/creativecloud/business/enterprise/xd.html>. Acesso em: Dez de 2019. Citado na página 15.

AUTODESK. *AutoDesk Forge Model Derivative*. 2019. Disponível em: <https://forge.autodesk.com/en/docs/model-derivative/v2/developers_guide/overview/>. Acesso em: Dez de 2019. Citado na página 38.

AUTODESK. *AutoDesk Forge Viewer*. 2019. Disponível em: <https://forge.autodesk.com/en/docs/viewer/v2/developers_guide/overview/>. Acesso em: Dez de 2019. Citado na página 38.

AUTODESK. *AutoDesk Forge Viewer ChangeLog*. 2019. Disponível em: <https://forge.autodesk.com/en/docs/viewer/v7/change_history/changelog_v7/migration_guide_v6_to_v7/>. Acesso em: Dez de 2019. Citado na página 38.

AVDIC, D. *React Native vs Xamarin – Mobile for industry*. Dissertação (Dissertação de Mestrado) — Lund University, 2019. Citado na página 38.

AZHAR, S. Building information modeling (bim): Trends, benefits, risks, and challenges for the aec industry. *Leadership and management in engineering*, American Society of Civil Engineers, v. 11, n. 3, p. 241–252, 2011. Citado na página 15.

CORDEIRO, M. I. G. M. C. C. C. O perfil do operário da indústria da construção civil de feira de santana: Requisitos para uma qualificação profissional. 2002. Citado na página 11.

EISENMAN, B. *Learning React Native: Building Native Mobile Apps with JavaScript*. O'Reilly Media, 2015. ISBN 9781491929070. Disponível em: <https://books.google.com.br/books?id=274fCwAAQBAJ>. Citado na página 15.

FACEBOOK. *Flux - In Depth Overview*. 2019. Disponível em: <https://facebook.github.io/flux/docs/in-depth-overview>. Acesso em: Dez de 2019. Citado na página 16.

FROTA, F. R. D.; WEERSMA, M. R.; WEERSMA, L. A. Método de projetos Ágeis aplicado ao setor de construção civil: Caso comparativo entre construtoras de médio porte. *Anais do V SINGEP, São Paulo, Brasil*, 2016. Disponível em: <https://singep.org.br/5singep/resultado/700.pdf>. Acesso em: 8 dez. 2019. Citado na página 14.

GAMMA, E. et al. *Padrões de Projetos - Soluções Reutilizáveis de software orientado a objeto*. [S.l.]: bookman, 2008. 20-22 p. Citado na página 16.

GOOGLE. *Material Design*. 2019. Disponível em: <https://material.io/design/introduction/>. Acesso em: Dez de 2019. Citado na página 15.

MANZIONE, L. *Proposição de uma Estrutura Conceitual de Gestão do Processo de Projeto Colaborativo com o uso do BIM*. Tese (Doutorado) — Escola Politécnica, Universidade de São Paulo, São Paulo, Brasil, 2013. Disponível em: <https:

//www.teses.usp.br/teses/disponiveis/3/3146/tde-08072014-124306/pt-br.php>. Acesso em: 8 dez. 2019. Citado na página 15.

McKinsey Global Institute. *Reinventing Construction: A route to higher productivity* . 2017. Disponível em: <https://www.mckinsey.com/~/media/McKinsey/Industries/Capital%20Projects%20and%20Infrastructure/Our%20Insights/Reinventing%20construction%20through%20a%20productivity%20revolution/MGI-Reinventing-Construction-Executive-summary.ashx>. Acesso em: Dez de 2019. Citado na página 11.

MEIRELLES, F. S. 30ª pesquisa anual do uso de ti nas empresas. 2019. Disponível em: <https://eaesp.fgv.br/sites/eaesp.fgv.br/files/pesti2019fgvciappt_2019.pdf>. Acesso em: Dez de 2019. Citado na página 44.

MENDONÇA, J. et al. Implantação de gestão Ágil na engenharia civil. 2018. Disponível em: <http://www.computacao.unitri.edu.br/erac/index.php/e-rac/article/view/1242>. Acesso em: 8 dez. 2019. Citado na página 14.

Ministério da Economia. *Estratégia BIM BR*. 2019. Disponível em: <http://www.mdic.gov.br/images/REPOSITORIO/sdci/CGMO/Livreto_Estratgia_BIM_BR-6.pdf>. Acesso em: Dez de 2019. Citado na página 11.

PEYROTT, S. *The JWT Handbook*. [S.l.]: Auth0, 2016. Citado na página 17.

REACT. *React - Introduzindo o JSX*. 2019. Disponível em: <https://pt-br.reactjs.org/docs/introducing-jsx.html>. Acesso em: Dez de 2019. Citado na página 16.

REACTNATIVE. *React Native*. 2019. Disponível em: <https://facebook.github.io/react-native/>. Acesso em: Dez de 2019. Citado na página 16.

REACTNATIVE. *React Native - Introducing new iOS WebViews*. 2019. Disponível em: <https://facebook.github.io/react-native/blog/2018/08/27/wkwebview>. Acesso em: Dez de 2019. Citado na página 38.

REACTREDUX. *React Redux - Why Use React Redux?* 2019. Disponível em: <https://react-redux.js.org/introduction/why-use-react-redux>. Acesso em: Dez de 2019. Citado na página 16.

SADALAGE, M. F. P. J. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. [S.l.]: Addison-Wesley Professional, 2012. Citado na página 17.

SYED, B. A. *Beginning Node.js*. [S.l.]: apress, 2014. Citado na página 17.

Apêndices

APÊNDICE A – Código do componente do visualizador de arquivos

```
1   import React, { useState, useEffect } from 'react';
2   import { WebView, StyleSheet, View, ActivityIndicator, Text,
        Image } from 'react-native';
3   import qs from 'qs';
4   import base64 from 'react-native-base64';
5   import COLORS from '../../colors';
6
7   export default AutodeskViewerWrapper = (props) => {
8
9       const urn = base64.encode(props.navigation.getParam('urn')
            ) || '';
10
11      const [urnReady,setUrnReady] = useState(false);
12
13      const [urnLoading,setUrnLoading] = useState(true);
14
15      let viewer;
16
17      let token;
18
19      const loginAutodesk = () => {
20          return fetch('https://developer.api.autodesk.com/
                authentication/v1/authenticate',
21              {
22                  method: 'POST',
23                  headers: {
24                      'Content-Type': 'application/
                            x-www-form-urlencoded'
25                  },
26                  body: qs.stringify({
27                      client_id: 'SUBSTITUIR-COM-ID-AUTODESK',
28                      client_secret: '
                            SUBSTITUIR-COM-SECRET-AUTODESK',
29                      grant_type: 'client_credentials',
```

```
30                      scope: 'data:read'
31                  })
32              })
33              .then(res => res.json())
34              .then(res => res.access_token)
35              .catch(function(res){ console.log(res) });
36      };
37
38      const testFile = (urn, token) => {
39          return fetch(`https://developer.api.autodesk.com/
               modelderivative/v2/designdata/${urn}/manifest`,
40              {
41                  headers: {
42                      'Authorization': `Bearer ${token}`
43                  }
44              })
45              .then(r =>  {
46                  if(!r.ok) {
47                      return r.status;
48                  }
49                  else {
50                      return r.json();
51                  }
52              })
53              .catch(function(res){ console.log('erro testfile '
                   ,res) });
54      };
55
56      const loadUrn = async () => {
57          if(token===undefined) {
58              token = await loginAutodesk();
59          }
60          var js = `initializeViewer("urn:${urn}","${token}")`;
61          viewer.injectJavaScript(js);
62      };
63
64      useEffect(() => {
65
66          (async function testIfReadyFile() {
67              token = await loginAutodesk();
68              const respTest = await testFile(urn,token);
69
```

```
70              if(respTest instanceof Object &&
                   respTest.hasOwnProperty('status') &&
                   respTest.status === 'success') {
71                  setUrnReady(true);
72                  setUrnLoading(false);
73              }
74              else {
75                  setUrnLoading(false);
76                  console.log(respTest);
77              }
78          })();
79      },[])
80
81      if(urnLoading || !urnReady) {
82          return (
83              <View style={{flex:1,justifyContent:"center",
                   alignItems:'center'}}>
84                  {urnLoading ?
85                      <View style={{flex:1,justifyContent:"
                           center",alignItems:'center'}}>
86                          <ActivityIndicator size="large" color=
                               {COLORS.darkblue} />
87                      </View>
88                  :
89                      <View style={{flex:1,justifyContent:"
                           center",alignItems:'center'}}>
90                          <View style={{flex:0.15,
                               justifyContent:"center",alignItems:
                               'center'}}>
91                              <Text style={{fontSize:18,
                                   color:COLORS.red,fontWeight:"
                                   bold",textAlign:'center'}}>
92                                  Nao conseguimos carregar esse
                                       arquivo
93                              </Text>
94                          </View>
95                          <Image
96                              source={require('../../../images/
                                   sad-worker-cannot-load.png')}
97                              style={{flex:0.7,resizeMode:'
                                   contain'}}
98                          />
```

```
99                          <View style={{marginTop:20,flex:0.15,
                               justifyContent:"center",alignItems:
                               'center'}}>
100                             <Text style={{
                                    color:COLORS.lightgrey}}>Entre
                                    em contato com o time da InSite
                                    !</Text>
101                         </View>
102                     </View>
103                 }
104             </View>
105         );
106     }
107
108     return (
109         <View style={styles.webContainer}>
110
111
112             <WebView
113                 useWebKit={true}
114                 source={{ html: latestVersion }}
115                 style={styles.webview}
116                 javaScriptEnabled={true}
117                 scrollEnabled={false}
118                 ref = {webview => { viewer = webview; }}
119                 onLoadEnd={loadUrn}
120             />
121
122         </View>
123     );
124 }
125
126 const styles = StyleSheet.create({
127     webContainer: {
128         flex: 1,
129         backgroundColor: '#eee',
130         justifyContent: 'flex-start',
131         alignItems: 'stretch'
132     },
133     webview: {
134         flex: 1
135     }
```

```
136 });
137
138 const latestVersion = `<head>
139 <meta name="viewport" content="width=device-width,
    minimum-scale=1.0, initial-scale=1, user-scalable=no" />
140 <meta charset="utf-8">
141 <link rel="stylesheet" href="https://
    developer.api.autodesk.com/modelderivative/v2/viewers/7.*/
    style.min.css" type="text/css">
142 <script src="https://developer.api.autodesk.com/
    modelderivative/v2/viewers/7.*/viewer3D.min.js"></script>
143 <style>
144     body {
145         margin: 0;
146     }
147     #forgeViewer {
148         width: 100%;
149         height: 100%;
150         margin: 0;
151         background-color: #fff;
152     }
153 </style>
154 </head>
155 <body>
156 <div id="forgeViewer"></div>
157 </body>
158
159 <script>
160 var viewer;
161 function initializeViewer(urn, token) {
162     var options = {
163         env: 'AutodeskProduction',
164         api: 'derivativeV2',
165         accessToken: token
166     };
167     Autodesk.Viewing.i18n.options.useCookie=false;
168     Autodesk.Viewing.Initializer(options, function() {
169         var htmlDiv = document.getElementById('forgeViewer');
170         viewer = new Autodesk.Viewing.GuiViewer3D(htmlDiv);
171         var startedCode = viewer.start();
172         if (startedCode > 0) {
```

```
173             console.error('Failed to create a Viewer: WebGL
                    not supported.');
174             return;
175         }
176         console.log('Initialization complete, loading a model
                next...');
177         viewer.setBackgroundColor(255,255,255,255,255,255);
178     });
179     var documentId = urn;
180     Autodesk.Viewing.Document.load(documentId,
            onDocumentLoadSuccess, onDocumentLoadFailure);
181     function onDocumentLoadSuccess(viewerDocument) {
182         // viewerDocument is an instance of
                Autodesk.Viewing.Document
183         var defaultModel = viewerDocument.getRoot().
                getDefaultGeometry();
184         viewer.loadDocumentNode(viewerDocument, defaultModel);
185     }
186     function onDocumentLoadFailure() {
187         console.error('Failed fetching Forge manifest');
188     }
189 }
190 </script>
```