UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Bruno Sérgio Cardoso Vieira

Aplicativo para marcação de pontos de interesse em modelos 3D de projetos de engenharia civil

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Bruno Sérgio Cardoso Vieira

Aplicativo para marcação de pontos de interesse em modelos 3D de projetos de engenharia civil

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

2021

Resumo

O processo de acompanhamento de obras de construção civil produz uma grande quantidade de dados que são, posteriormente, agrupados em relatórios que trafegam dentro das empresas. A organização e armazenamento desses dados é de fundamental importância para garantir o acesso rápido e assertivo das informações necessárias, aumentando a produtividade e a eficiência dos profissionais envolvidos e, consequentemente, a qualidade do projeto. Este trabalho propõe o desenvolvimento de uma aplicação para dispositivos móveis que auxilie o dia-a-dia desses profissionais por permitir a gestão visual de pontos de interesse nos modelos 3D dos projetos por meio de marcações que contenham registros importantes para as atividades realizadas no canteiro de obras. Na etapa de desenvolvimento, a aplicação proposta foi desenvolvida para a plataforma iOS e entende-se que os conceitos utilizados e as experiências práticas adquiridas neste trabalho sejam relevantes para outros estudos e pesquisas que busquem evoluir o envolvimento da tecnologia no processo de gestão de obras de construção civil.

Palavras-chave: Construção Civil, Marcador, Modelo 3D, Aplicativo iOS.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas

iOS iPhone Operating System

GPS Global Positioning System

MVC Model-View-Controller

MVP Model-View-Presenter

MVVM Model-View-ViewModel

VIPER View-Interactor-Presenter-Entity-Router

DS Design System

FBX FilmBox

IDE Integrated Development Environment

USDZ Universal Scene Description (Zip-compressed)

JSON JavaScript Object Notation

BIM Building Information Modeling

PDF Portable Document Format

Sumário

1 Introdução

Em 2019, o setor de construção civil movimentou 273,8 milhões de reais no Brasil, com as atividades de 1,9 milhões de pessoas em 125,1 mil empresas de construção ativas, atuando em diversos tipos de obra, como construção de edifícios e projetos de infraestrutura (IBGE, 2020). Esses números apontam a importância do setor para o desenvolvimento e produção do país.

A concorrência do setor destaca a relevância da especialização e das inovações tecnológicas para a execução dos serviços com maior produtividade e qualidade. Com isso, o processo de planejamento e gestão da obra é de fundamental importância para as empresas (SILVA, 2011).

De acordo com Costa (2016), o controle de execução de um projeto, na maioria das vezes, ocorre informalmente, com a experiência e a qualificação dos profissionais envolvidos sendo extremamente variável, afetando a qualidade das informações coletadas.

Também, com base em relatos de experiências de colaboradores deste trabalho, empresas de construção civil de pequeno e médio porte realizam o processo de acompanhamento de obra em formato de relatórios redigidos pela equipe que realiza visitas à obra, registrando fotos e notas do progresso da obra e de possíveis pontos de atenção que devem ser solucionados para o prosseguimento das atividades. Encontram-se, porém, muitas dificuldades no momento da unificação desses registros nos relatórios pois, geralmente, os registros não são organizados e estruturados de forma a facilitar essa busca.

Considerando a natureza visual do acompanhamento das atividades, a ligação de um registro à um ponto visual no modelo 3D tende a contribuir na localização das informações relevantes e na organização dos dados colhidos.

Este trabalho propõe uma aplicação para dispositivos móveis que possa ser utilizada no acompanhamento de uma obra, com o objetivo de destacar pontos de relevância em uma representação visual do projeto em execução. A etapa de desenvolvimento deste trabalho construiu a aplicação proposta para a plataforma iOS, sendo o princípio de um sistema completo e útil, e que também seja importante para os estudos do envolvimento da tecnologia na gestão de uma obra.

1.1 Objetivo

O objetivo deste trabalho é idealizar, modelar e desenvolver um sistema de registros de pontos de interesse em modelos 3D de projetos de construção civil, utilizando aplicativos móveis como plataforma de interação dos profissionais envolvidos no processo

de acompanhamento de obra para gerenciamento das informações importantes. O aplicativo deve ser capaz de associar modelos 3D ao gerenciamento de uma obra, considerando suas etapas, e prover controles de navegação, permitindo que o usuário consiga adicionar as marcações nos locais desejados.

1.2 Método

Para alcançar o objetivo descrito na seção 1.1, propõem-se as seguintes etapas:

- Desenvolvimento de um protótipo:
  - Implementar um protótipo para validação técnica da solução;
  - Apresentar o protótipo para profissionais da área de Engenharia Civil que atuem no acompanhamento de obras para estudo das necessidades e detalhamento dos requisitos;
  - Especificar o escopo da aplicação.
- Modelagem da solução:
  - Definir as plataformas do aplicativo;
  - Refinar detalhes técnicos de desenvolvimento;
  - Definir a interface e a usabilidade.
- Implementação:
  - Desenvolver o aplicativo;
  - Validar o resultado final.

1.3 Organização do trabalho

Este trabalho está organizado da seguinte forma: o capítulo dois detalha os fundamentos e as ferramentas utilizadas para a execução do trabalho. O capítulo três apresenta a etapa de desenvolvimento do aplicativo, desde o processo de prototipação à construção do aplicativo final. Por último, o capítulo quatro aborda a análise do resultado, possíveis evoluções deste trabalho e aplicações de fundamentos deste projeto em outras áreas.

2 Fundamentação Teórica

Neste capítulo, são apresentados os conceitos e as ferramentas utilizadas neste trabalho. A seção 2.1 discorre sobre o que são aplicativos móveis e os conceitos de cada plataforma de desenvolvimento. A seção 2.2 detalha os conceitos utilizados na etapa de desenho da solução. A seção 2.3 lista as ferramentas importantes para a implementação do projeto. Por último, a seção 2.4 apresenta conceitos de desenvolvimento ágil que foram seguidos durante o desenvolvimento.

2.1 Aplicativos Móveis

Os aplicativos móveis, geralmente, fazem parte de um sistema dividido em duas seções principais: *frontend* e *backend*. A Figura 1 exemplifica um sistema que contém peças dessas duas seções. O *backend* de um sistema de *software* é a parte do sistema que não é visível ao usuário, sendo responsável pelo armazenamento e manipulação das informações necessárias para o contexto do sistema, como acesso à bancos de dados e conexões à outros sistemas de *software*.

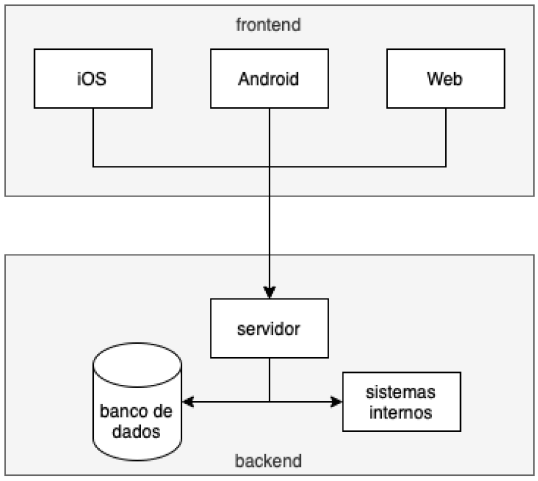

Figura 1 – Exemplo de um sistema contendo aplicativos móveis

Por outro lado, o *frontend* de um sistema de *software* é a parte com a qual o usuário interage, ou seja, é a parte visível. Aplicativos móveis são peças de *frontend* que se integram ao sistema com o papel de interface para dispositivos móveis, como celulares e *tablets*. Diversas tecnologias são utilizadas para suprir a necessidade de desenvolvimento desses aplicativos. O mercado de dispositivos móveis é composto, em sua grande maioria, por dispositivos *Android* e *iPhone Operating System (iOS)*, com 72.45% e 26.74%, res-

pectivamente, em setembro de 2021, segundo o StatCounter[1]. Por consequência, essas são as plataformas mais suportadas no processo de criação de um novo sistema. O primeiro ponto a ser levantado na etapa de idealização do projeto é a escolha entre desenvolvimento nativo, híbrido, ou multiplataforma.

2.1.1 Nativo, híbrido ou multiplataforma

Um aplicativo nativo é desenvolvido utilizando a linguagem de programação específica para a plataforma. Aplicativos *Android* e *iOS* são desenvolvidos em *Kotlin*/*Java* e *Swift*/*Objective-C*, respectivamente. Essas linguagens, por serem adotadas oficialmente para o desenvolvimento dos aplicativos, costumam ser otimizadas para integração com a plataforma, resultando em melhores resultados de desempenho e confiabilidade. Por outro lado, o custo de um time com desenvolvedores das duas plataformas tende a ser mais alto (SHARMA, 2020).

Um aplicativo híbrido é um aplicativo construido na plataforma desejada mas que, internamente, utiliza os componentes nativos de exibição de páginas *Web* (conhecidos como *WebView*) para exibir o conteúdo *Web* customizado para dispositivos móveis. Embora existam ferramentas, como *PhoneGap* e *Ionic*, que auxiliem na utilização das funcionalidades nativas do dispositivo, como câmera e Global Positioning System (GPS), ajustar o funcionamento dessa integração para cada plataforma suportada ainda é um problema encontrado no momento da implementação de funções mais complexas.

Um aplicativo multiplataforma é desenvolvido em uma linguagem de programação específica da ferramenta utilizada, porém é capaz de ser exportado para as diferentes plataformas desejadas. Essa característica, entretanto, costuma resultar em menor desempenho e dificuldades de integração com algumas funcionalidades dos dispositivos (SHARMA, 2020). Com o passar dos anos, diversas ferramentas foram criadas com esse intuito, como *Xamarin*, *ReactiveNative* e *Flutter*.

Um dos desafios dos desenvolvedores de aplicativos móveis, atualmente, é a tomada de decisão de qual abordagem seguir, considerando alguns aspectos relevantes para os resultados do projeto. A seguir, as abordagens são comparadas em relação à algumas características que podem influenciar essa escolha, com base em uma análise disponível em Saccomani (2019):

- Desempenho: os aplicativos nativos possuem melhor desempenho pois o código tende a ser mais simples e o desenvolvimento é realizado diretamente com as ferramentas otimizadas para cada plataforma. Os aplicativos híbridos dependem do desempenho do componente de visualização fornecido pelas plataformas nativas e que ainda não entregam o mesmo desempenho. Por fim, os aplicativos multiplataforma possuem

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

uma camada intermediária que é responsável ou por converter o código genérico em código nativo, ou por fornecer um ambiente para execução do aplicativo, o que também afeta o desempenho;

- Reaproveitamento de código: aplicações nativas possuem o menor reaproveitamento de código pois o desenvolvimento ocorre separadamente em cada plataforma e não são compartilháveis. Por outro lado, aplicações híbridas e multiplataforma possuem apenas um código-fonte, que é compartilhado entre as duas plataformas, com exceção da integração com funcionalidades do dispositivo, como câmera, GPS, microfone e giroscópio;

- Quantidade de desenvolvedores: como consequência do reaproveitamento de código, os aplicativos nativos tendem a exigir uma maior quantidade de desenvolvedores em comparação às demais abordagens por exigir atuação em duas bases de código ao invés de uma. Essa variável impacta diretamente os custos de um projeto uma vez que a contratação de mais pessoas aumenta os gastos com a folha salarial do time;

- Velocidade de desenvolvimento: também como resultado da base de código duplicada, um projeto nativo tende a levar mais tempo para ser concluído, enquanto que projetos híbridos e multiplataforma são desenvolvidos mais rapidamente, desde que a aplicação não exija muitas integrações complexas com funcionalidades do dispositivo uma vez que essas conexões dificultam o desenvolvimento;

- Obrigatoriedade de conexão com a internet: um aplicativo híbrido, por ser executado em uma *WebView*, exige conexão contínua com a internet para a realização de qualquer operação. As demais abordagens não possuem essa limitação;

Com base nessas e outras comparações pertinentes ao projeto, pode-se mensurar o impacto da escolha da abordagem para o desenvolvimento do aplicativo. Neste trabalho, a importância de um bom desempenho da aplicação no momento da manipulação dos modelos 3D e a integração com funcionalidades do dispositivo como câmera e gerenciamento de gestos, além do conhecimento e a experiência prévia, direcionaram a escolha da abordagem como desenvolvimento nativo em *iOS* utilizando a linguagem *Swift*.

2.2 Arquitetura de uma aplicação

De acordo com Robert (2008), a qualidade do código de uma aplicação afeta, diretamente, a produtividade do time que trabalha nela. Um código mal escrito pode parecer suficiente em um primeiro momento. Porém, conforme novas funcionalidades surgem e melhorias e correções se fazem necessárias, o código ruim precisa ser compreendido, ao menos

parcialmente, para que tais novos códigos, consequentemente ruins, sejam adicionados e as demandas, cumpridas.

Essa situação resulta, comumente, em mais pessoas envolvidas no desenvolvimento dessas evoluções. Como resultado, o baixo conhecimento dos novos desenvolvedores sobre o sistema tende a acelerar, ainda mais, a queda de produtividade da equipe até o ponto em que qualquer alteração no código se torna impossível e a única forma de retomar o desenvolvimento é por meio do replanejamento da aplicação, gerando custos indesejados. Entretanto, caso não haja uma mudança significativa no processo de desenho e codificação, não existem garantias de que o novo código desenvolvido não sofrerá das mesmas dificuldades do código anterior, iniciando um novo ciclo problemático.

Uma forma de prover essa mudança é por meio de padrões e boas práticas de desenvolvimento que fomentem a boa qualidade de código, como o conceito de arquitetura limpa. A arquitetura de uma aplicação define a forma como o sistema é dividido em componentes e como esses componentes se comunicam (ROBERT, 2017). O termo "limpa" se refere à responsabilidade de um componente, que deve ser preservada sem "sujeiras" de responsabilidades que deveriam ser de outros componentes. Por exemplo, um componente de banco de dados não deve conter a responsabilidade de apresentar dados em uma interface de usuário, pois sua responsabilidade é armazenar e organizar os dados utilizados no sistema.

Tal abstração estimula a elaboração de uma aplicação mais concisa e coesa, resultando em alguns benefícios: a aplicação requer uma fração dos recursos humanos para ser criada e mantida; as alterações são simples e rápidas; os defeitos são poucos e esporádicos, o esforço é minimizado, e a funcionalidade e a flexibilidade são maximizadas.

Para o desenvolvimento de aplicativos *iOS*, diversos padrões de arquitetura buscam seguir o conceito de arquitetura limpa para herdar esses benefícios descritos. Cada arquitetura possui pontos positivos e negativos que devem ser medidos durante a etapa de planejamento da aplicação para aumentar a assertividade da decisão de adotar uma delas. Complexidade das operações, experiência do time de desenvolvimento e estimativa de prazo de entrega são exemplos de características que impactam esse processo decisório. Algumas arquiteturas comumente utilizadas em projetos *iOS* são:

- Model-View-Controller (MVC): o *Model* contém a modelagem dos dados, a *View* contém os elementos visuais de exibição na interface, e o *Controller* é responsável por realizar conexões com servidores, pelas regras de renderização e de negócio, e por requisitar e formatar os dados. Nessa arquitetura, *View* e *Controller* são fortemente conectados a ponto de serem considerados uma única entidade. Os relacionamentos entre as entidades são de acordo com a Figura 2.

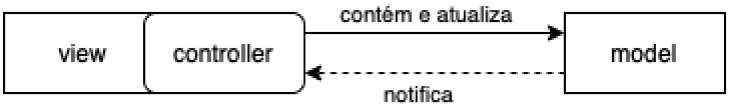

Figura 2 – Relacionamento das entidades da arquitetura MVC

- Model-View-Presenter (MVP): o *Model* tem o mesmo papel da arquitetura anterior, o *ViewController* contém os elementos visuais de exibição na interface e regras de renderização, e o *Presenter* é responsável por realizar conexões com servidores, pelas regras de negócio, e por requisitar e formatar dados. Em comparação ao MVC, mover algumas responsabilidades do *Controller* para o *Presenter* simplificam a responsabilidade da junção entre *View* e *Controller*, desacomplando-as do restante da arquitetura. Os relacionamentos entre as entidades são de acordo com a Figura 3.

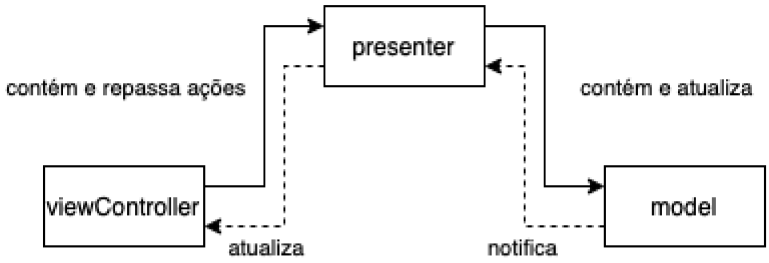

Figura 3 – Relacionamento das entidades da arquitetura MVP

- Model-View-ViewModel (MVVM): *Model* e *ViewController* contém as mesmas responsabilidades mencionadas na arquietura MVP. A diferença para essa arquitetura é o *ViewModel*, que possui as mesmas responsabilidades do *Presenter* em MVP, porém com foco em ligação de dados, criando um mecanismo de atualização direta entre o *ViewModel* e a *View* para que as atualizações de estados de qualquer uma das duas entidades atualize a outra. Os relacionamentos entre as entidades são de acordo com a Figura 4.

- View-Interactor-Presenter-Entity-Router (VIPER): o *ViewController* possui a mesma responsabilidade das arquiteturas anteriores, o *Interactor* contém as regras de negócio, as conexões com servidores e a requisição de dados, o *Presenter* contém as regras de renderização e a formatação dos dados, a *Entity* implementa a modelagem dos dados e o *Router* é responsável pelas transições nos fluxos de cenas. Essa é a

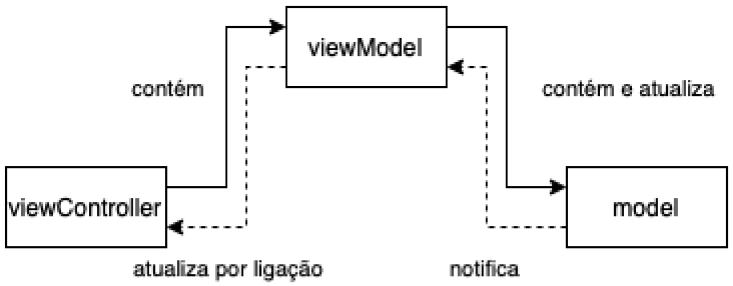

Figura 4 – Relacionamento das entidades da arquitetura MVVM

única arquitetura, dentre as descritas aqui, que define como as operações de roteamento são implementadas. Os relacionamentos entre as entidades são de acordo com a Figura 5.

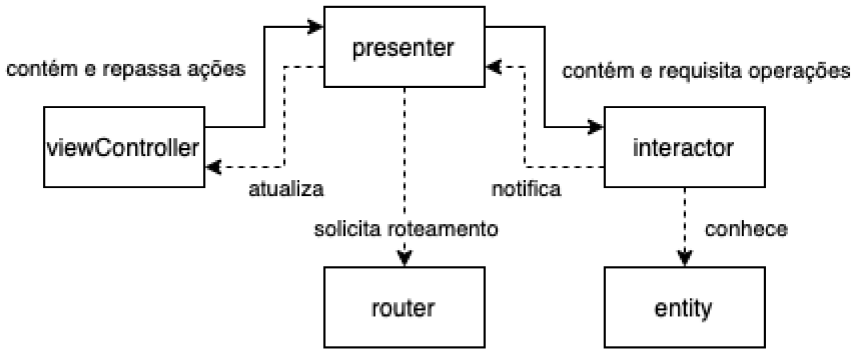

Figura 5 – Relacionamento das entidades da arquitetura VIPER

A tabela 1 apresenta uma comparação dessas arquiteturas em relação aos seguintes parâmetros:

1. Separação de responsabilidades: é a aplicação do conceito de componentes limpos, com responsabilidades bem definidas e isoladas;
2. Testabilidade: indica a possibilidade de adicionar estruturas de testes para validação de cada componente de forma unitária para garantir a qualidade do código implementado e antecipar problemas e erros nas fases de desenvolvimento e manutenção;
3. Complexidade da arquitetura: mensura a quantidade de desenvolvimento necessário para executar uma operação, considerando o número de componentes envolvidos na sequência de passos realizados até a conclusão da tarefa;

4. Facilidade de manutenção: qualifica a simplicidade e a agilidade em realizar manutenções no código desenvolvido, seja para correção de problemas ou evolução das funcionalidades.

	MVC	MVP	MVVM	VIPER
Separação de responsabilidades	baixa	baixa	média	alta
Testabilidade	baixa	média	média	alta
Complexidade da arquitetura	baixa	baixa	média	alta
Facilidade de manutenção	alta	alta	média	baixa

Tabela 1 – Comparativo de arquiteturas iOS

Essas comparações podem direcionar a escolha da arquitetura para o projeto ou a base a partir da qual uma nova arquitetura é criada aplicando mudanças que incrementem ou simplifiquem sua utilização, adequando-a ao contexto da aplicação.

2.2.1 Design System

De acordo com Kholmatova (2017), um Design System (DS) pode ser definido como um conjunto de padrões e práticas organizados, coerentemente, para servir ao propósito de um produto digital. Padrões são elementos recorrentes que são combinados para criar uma interface, como botões, campos de texto, ícones, cores e tipografia. Práticas são as formas de criar, compartilhar e utilizar tais padrões. Já Hassu (2021), define o mesmo conceito como "um ecossistema de bibliotecas com componentes desenvolvidos a partir de semânticas de design e gestão de estilo centralizada", destacando a importância do desenvolvimento (código) na existência do DS. Não existe uma definição estabelecida do termo *Design System* na comunidade de experiência do usuário e, consequentemente, este termo é utilizado de formas diferentes para definir seu objetivo. E existem outras além destas duas em destaque.

As diversas interpretações, entretanto, convergem para o mesmo objetivo: aumentar a consistência e a agilidade de criar e desenvolver as diferentes experiências de usuário que são necessárias para suprir as demandas em plataformas e produtos distintos. Por exemplo, uma empresa pode ter seu aplicativo disponível nas plataformas *iOS*, *Android* e *Web*. Cada uma dessas plataformas tem particularidades que variam a usabilidade da aplicação em seções isoladas, mas grande parte do sistema costuma ser semelhante e isto implica ser necessário construir as interfaces de cada plataforma com certa similaridade. Além disso, é importante manter a consistência da marca da empresa, utilizando as cores corretas e uma linguagem adequada de comunicação com o usuário.

Tal detalhamento está contido na documentação do *Design System*, que é constantemente evoluída para centralizar as informações relevantes para o alinhamento desses

pontos entre as partes envolvidas: o cliente encontra a linguagem geral da solução, o *designer* encontra os elementos disponíveis para uso, o desenvolvedor encontra os detalhes técnicos do que deve ser desenvolvido em sua plataforma, o *tester* encontra a fonte de comparação para as validações, e outras necessidades do processo de desenvolvimento do produto.

Este projeto procura se beneficiar de outra vantagem, que é a codificação de elementos customizáveis. A paleta de cores do aplicativo contém cores com nomenclatura semântica, como plano de fundo, ação primária, texto primário e resposta positiva, ao invés de cores com nomenclatura literal, como preto, branco e azul. Isso permite uma rápida customização da interface para empresas e clientes diferentes sem grande esforço técnico, por centralizar a definição das cores e facilitar a substituição, quando necessário (CHOUDHARY, 2021).

2.3 Ferramentas utilizadas

O desenvolvimento de um sistema, geralmente, necessita de ferramentas fundamentais para a construção da solução ou ferramentas auxiliares, que fornecem melhorias para etapas do processo. Algumas possuem opções disponíveis, sendo necessário escolher a ferramenta que mais se adeque à necessidade e ao propósito do sistema. Nesta seção são apresentadas as ferramentas que foram empregadas ao longo do desenvolvimento do trabalho.

- Revit

  Revit é uma ferramenta desenvolvida pela empresa *Autodesk* que une arquitetura, engenharia e construção em uníco ambiente de modelagem, aumentando a eficiência e a economia de projetos nessas áreas (AUTODESK, 2021b).

  Por meio dessa ferramenta, é possível gerar modelos 3D das construções e exportá-los em formatos de arquivo distintos, que são compatíveis com diversos outras aplicações como, por exemplo, o formato FilmBox (FBX), de propriedade da *Autodesk*, que pode ser convertido, posteriormente, para um formato compatível com aplicativos *iOS*.

- Git e Github

  Git é uma ferramenta gratuita de controle de versionamento de arquivos que beneficia o desenvolvimento de código de algumas maneiras como: (1) facilitando a revisão e aprovação de modificações e (2) permitindo a recuperação de versões anteriores (GIT, 2021).

  Github é uma ferramenta gratuita que utiliza o *Git* e estende sua funcionalidade com mais alguns benefícios como: (1) criação de repositórios para armazenamento

dos arquivos na nuvem e (2) disponibilização de um portal *web* para gerenciamento do projeto (GITHUB, 2021).

- Xcode

  Xcode é um ambiente de desenvolvimento integrado, do inglês *Integrated Development Environment (IDE)*, formado por um conjunto de ferramentas utilizadas pelos desenvolvedores de aplicativos para as plataformas da *Apple* (APPLE, 2021d), como *iOS*, *MacOS* e *WatchOS*. Essa IDE permite a codificação em linguagem *Swift* e a instalação do aplicativo em dispositivos, simuladores ou reais, para fins de teste.

- UIKit

  UIKit é uma biblioteca que provê a infraestrutura necessária para os aplicativos iOS (APPLE, 2021c). Contém os elementos visuais utilizados na construção do *layout* e das jornadas de utilização do aplicativo, como texto, botões, imagens e listas.

- SceneKit

  SceneKit é uma biblioteca que permite a importação, manipulação e renderização de objetos 3D dentro de um aplicativo iOS (APPLE, 2021b). Dentre as funções disponíveis estão: controle de câmera, luz, animações e simulações de física.

- Swift Package Manager

  Para a gestão de módulos, é necessária uma ferramenta que controle a distribuição e integração ao projeto principal (SWIFT, 2021). Existem algumas ferramentas disponíveis no mercado, como *CocoaPods* e *Carthage* mas, para esse projeto, foi utilizado o *Swift Package Manager* por ser de fácil manipulação e por estar integrado, nativamente, ao *Xcode*.

2.4 Desenvolvimento ágil

De acordo com Sommerville (2011): "Os processos de desenvolvimento rápido de *software* são concebidos para produzir, rapidamente, *softwares* úteis. O *software* não é desenvolvido como uma única unidade, mas como uma série de incrementos - cada incremento inclui uma nova funcionalidade do sistema". Essa abordagem é um contraponto ao desenvolvimento dirigido a plano, que reúne métodos herdados dos projetos de engenharia que possuem foco na rigorosidade e no controle do processo, resultando em etapas de documentação e levantamento de requisitos mais longas e detalhadas.

Nos métodos ágeis, o desenvolvimento acontece de forma iterativa, adicionando pequenas novas funções ao sistema conforme novos requisitos surgem e as funções anteriores são finalizadas. Isso diminui o tempo gasto em processos e aumenta o tempo gasto na implementação do sistema. Como não é mais necessário aguardar um longo ciclo até uma

versão de validação ser disponibilizada, os problemas no sistema são encontrados mais rapidamente. Além disso, alterações nos requisitos do sistema são melhor incorporadas, pois não desencadeiam problemas em outras etapas que foram previamente definidas, sendo uma abordagem mais dinâmica para aceitar a velocidade das atualizações frequentemente requisitadas pelo cliente. Para atingir esses objetivos, os processos ágeis possuem algumas características comuns, derivadas do Manifesto Ágil. Ainda, de acordo com Sommerville (2011), algumas delas são:

- Envolvimento do cliente: os clientes são responsáveis por validar as iterações e prover os novos requisitos do sistema, priorizando as necessidades mais importantes;
- Entrega incremental: o desenvolvimento é realizado de forma incremental, entregando novos requisitos a cada nova versão do sistema;
- Aceitar as mudanças: o sistema deve ser desenvolvido de forma a acomodar as mudanças que ocorrerão no processo.

Por outro lado, algumas dificuldades são encontradas no momento de integrar um método ágil ao processo de desenvolvimento de um projeto, o que pode impedir o aproveitamento dos benefícios ou, até mesmo, transformar um benefício em um malefício. O envolvimento do cliente, por exemplo, é fundamental para a dinâmica do método. Consequentemente, caso o cliente não esteja presente ou não possua completo entendimento do sistema, os requisitos não fornecerão o conhecimento necessário para a implementação da funcionalidade com qualidade. Além disso, priorização de tarefas pode ser um problema para o fluxo do time, principalmente em casos de conflitos de prioridades dos clientes envolvidos. Por último, alterar a cultura e os processos de uma empresa pode gerar resistência e é comum que a mudança seja interrompida de forma a enfraquecer os processos antigos e não aproveitar os novos processos.

Com isso, é sempre muito importante ponderar o ambiente no qual o método ágil será implantado, validando se as características das pessoas envolvidas e o contexto do projeto se aproximam dos cenários indicados pela metodologia.

2.4.1 Kanban

Uma das formas de aplicar esses conceitos de desenvolvimento ágil é pela metodologia *Kanban*. O *Kanban* não é uma prática exclusiva do desenvolvimento de sistemas e pode ser utilizado em situações em que é necessário controlar a quantidade de uma operação dentro de um fluxo de processos. No contexto deste trabalho, a operação representa uma tarefa referente ao desenvolvimento do aplicativo, enquanto que os processos são as etapas pelas quais essas tarefas passam até que sejam concluídas.

Em resumo, uma quantidade de tarefas que podem ser simultaneamente trabalhadas é definida e uma nova tarefa não pode ser iniciada enquanto alguma das tarefas ativas não for finalizada. Assim, garante-se que o foco sempre esteja em finalizar as tarefas correntes, impulsionando a conclusão das tarefas. Essa mecânica define o método *Kanban* como um sistema de puxar atividades, em contraponto a sistemas de empurrar atividades, que tendem a adicionar novas atividades com base, apenas, em suprir novas demandas (ANDERSON, 2013).

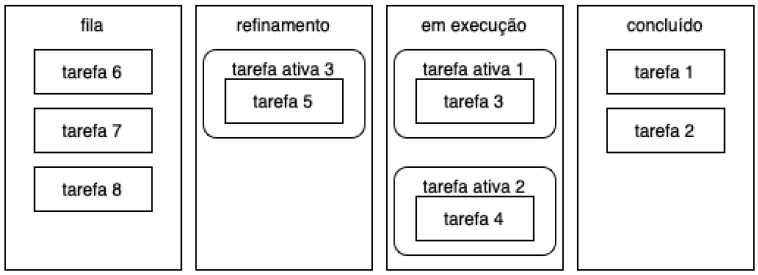

Figura 6 – Exemplo de um quadro Kanban

A Figura 6 exemplifica a utilização de um quadro *Kanban* para acompanhamento das tarefas em um processo que permite a atuação em 3 tarefas simultaneamente, indicando que entre as colunas refinamento e em execução não pode existir mais que 3 tarefas ativas, ou seja, cada tarefa deve estar vinculada à um dos blocos "tarefa ativa" para percorrer as etapas do processo. No exemplo, as tarefas 1 e 2 foram concluídas, as tarefas 3 e 4 estão em execução, a tarefa 5 está em refinamento e as tarefas 6, 7 e 8 estão na fila, aguardando a liberação de um bloco pela conclusão de uma tarefa.

2.4.2 Prototipação

Outro método aplicado ao desenvolvimento ágil de sistemas é a prototipação. Segundo Sommerville (2011), "um protótipo é uma versão inicial de um sistema de *software*, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções". Logo, um protótipo auxilia a evolução do projeto por antecipar mudanças de requisitos, verificar a viabilidade da proposta e definir interfaces de usuário, por exemplo.

Embora o protótipo seja semelhante à um produto final, geralmente, é inviável utilizá-lo com este objetivo pois a implementação mais "casual" gera alguns problemas estruturais e de documentação do sistema. Assim, é recomendado o descarte do protótipo e o início do desenvolvimento de um novo sistema utilizando os conhecimentos adquiridos na fase de prototipação.

3 Desenvolvimento

Este capítulo descreve a etapa de desenvolvimento do aplicativo. Primeiramente, a seção 3.1 apresenta o planejamento semanal das atividades e o objetivo de cada fase de desenvolvimento. Em seguida, a seção 3.2 aborda o processo de prototipação, pontuando os primeiros passos e as fucionalidades implementadas no protótipo. A seção 3.3 detalha a etapa de modelagem da aplicação e, por fim, a seção 3.4 descreve os ciclos de desenvolvimento, nos quais ocorrem a implementação das funcionalidades.

3.1 Planejamento

Com o intuito de organizar a sequência das atividades, foi definido um cronograma em três etapas, conforme a Figura 7. São elas:

1. Protótipo: desenvolvimento de um protótipo com o intuito de validar tecnicamente a viabilidade da solução, além de expôr um aplicativo navegável aos contribuidores da área de Engenharia Civil para detalhamento dos requisitos;
2. Desenho: modelagem da solução com diagramas e descrições para melhor entendimento do sistema;
3. Primeiro ciclo: desenvolvimento das funcionalidades principais do aplicativo e primeira validação dos resultados;
4. Segundo ciclo: refinamento das funcionalidades do primeiro ciclo e validação do aplicativo final.

Figura 7 – Cronograma de desenvolvimento

3.2 Protótipo

Seguindo o conceito de prototipação, detalhado na seção 2.4.2, a baixa visibilidade do escopo dos requisitos e das soluções técnicas envolvidas induziu o início do desen-

volvimento com um protótipo descartável. Nesta etapa, foi possível estudar o processo de geração de modelos 3D e implementar um pequeno aplicativo com as funcionalidades conhecidas até o momento.

3.2.1 Geração de modelos 3D

Diversos formatos de arquivo 3D podem ser importados e manipulados nativamente pelo *Xcode* em aplicativos *iOS*. O primeiro passo de estudo para esse desenvolvimento consistiu em avaliar a viabilidade da utilização dos arquivos 3D considerando o fluxo de trabalho dos usuários envolvidos em atividades de acompanhamento de obra. De acordo com Apple (2021a), aplicativos *iOS* suportam a seguinte lista de extensões de arquivos 3D:

- *Alembic*: .abc;
- *Universal Scene Description*: .usd, .usda, .usdc;
- *Universal Scene Description (zip*: .usdz;
- *Polygon*: .ply;
- *Wavefront Object*: .obj;
- *Standard Tessellation Language*: .stl.

Na plataforma *Revit*, o engenheiro tem a possibilidade de criar Vistas 3D, que são renderizações 3D do projeto modelado com suas diferentes estruturas, como paredes, pisos, portas, vigas, colunas, sistemas hidráulicos, sistemas elétricos, entre outras. Além disso, as Vistas possuem uma ferramenta de filtragem que permite a exibição de estruturas selecionadas e cada vista pode ser gerada manualmente como, por exemplo, criando uma vista para cada andar da construção. Desta forma, podemos criar modelos 3D relevantes para o acompanhamento da obra e importá-los simultaneamente no aplicativo, permitindo a separação dos marcadores para contextos mais apropriados e isolados.

A exportação de uma Vista 3D pode ser realizada em diversos formatos, segundo Autodesk (2021a). São eles:

- Formatos CAD *(Computed-aided Design)*: .dwg, .dxf, .dgn;
- *Design Web Format*: .dwf;
- *Industry Foundation Classes* : .ifc;
- *FilmBox*: .fbx;

- *Standard Tessellation Language*: .stl.

Dentre essa lista, apenas o formato *STL* foi reconhecido nativamente pelo *iOS*. Este formato, porém, apresentou um problema. O arquivo *STL* exportado pelo *Revit* sempre é importado pelo *Xcode* com uma rotação de 90 graus. Esse problema ocorre devido ao fato de que o eixo padrão utilizado pelo *Revit* ser diferente do utilizado pelo *Xcode*, o que inviabiliza o uso deste formato sem grandes contornos técnicos para o ajuste da rotação de todo arquivo importado.

O formato alternativo, utilizado para o desenvolvimento deste projeto, é o *FBX*, que deve ser convertido para o formato *Universal Scene Description (Zip-compressed) (USDZ)*, válido para aplicativos *iOS*, pela ferramenta *Reality Converter* (Figura 8). Por ser uma ferramenta *Beta*, o *Reality Converter* apresentou algumas inconsistências de compatibilidade com os arquivos gerados pelo *Revit*, sendo necessário um passo adicional de conversão para que a ferramenta gerasse o arquivo *USDZ* corretamente. Este passo consiste em importar o arquivo *FBX* do *Revit* na ferramenta *3DS Max* e exportá-lo novamente como *FBX* corrigindo, assim, os pontos incompatíveis com o *Reality Converter*.

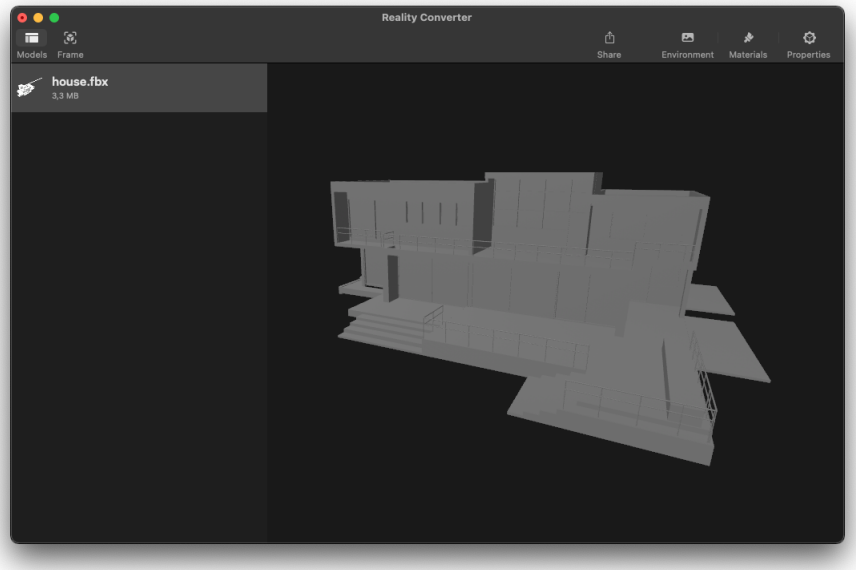

Figura 8 – Exportação de arquivo .USDZ no Reality Converter

3.2.2 Funcionalidades

O escopo do protótipo incluiu as seguintes funcionalidades:

- **Renderização**: a interface central do aplicativo acontece na renderização do modelo 3D de um projeto, como uma casa, um apartamento, um condomínio de apartamentos ou um prédio comercial. Para tanto, o protótipo teve o intuito de validar

sua importação e manipulação na plataforma *iOS*. O aprendizado adquirido nesta etapa contém a exibição do modelo 3D, importado como recurso do projeto para simplificação do problema, e os controles de câmera, como zoom, deslocamento e rotação. Este controle é fornecido pela biblioteca *SceneKit* que contém, nativamente, as ações para manipulação do modelo 3D, desde que ativadas na cena exibida por meio de um atributo do componente *SceneView*. É necessário, apenas, adicionar um controle de rotação para manter o modelo em posição vertical para simplificar o acompanhamento de seu posicionamento no momento da manipulação da câmera.

- **Marcador**: os marcadores são elementos de interface posicionados no modelo 3D para registrar os locais de relevância nos quais são vinculadas as fotos, como paredes, portas, colunas estruturantes, sistemas hidráulicos ou sistemas elétricos. O desenvolvimento incluiu a criação e a listagem desses marcadores, sendo necessário validar os mecanismos de gerenciamento de interação com o modelo 3D para leitura das coordenadas do toque do usuário na tela, indicando o posicionamento do marcador, bem como o armazenamento dessas coordenadas para uso posterior.

- **Foto**: a foto é o artefado armazenado nos marcadores como evidência dos registros relevantes à obra, como rachaduras e defeitos em acabamentos. Para tal, foi necessário validar a integração com a galeria de fotos do dispositivo e com o sistema de arquivos para armazenamento.

Com o escopo do protótipo definido, as atividades foram adicionadas ao novo quadro *Kanban* para acompanhamento e visualização do progresso das atividades, conforme Figura 9. As tarefas são adicionadas na coluna Fila, na qual aguardam a liberação da capacidade de desenvolvimento antes de serem puxadas para a coluna Em Desenvolvimento, onde ficam até que o desenvolvimento seja concluído. Na sequência, as tarefas são movidas para a coluna Desenvolvido, o que indica a finalização do desenvolvimento e o aguardo de uma validação inicial do resultado. Por último, a tarefa é movida para a coluna Finalizado, o que representa o fim do processo daquela tarefa e a liberação para uma nova tarefa ser iniciada.

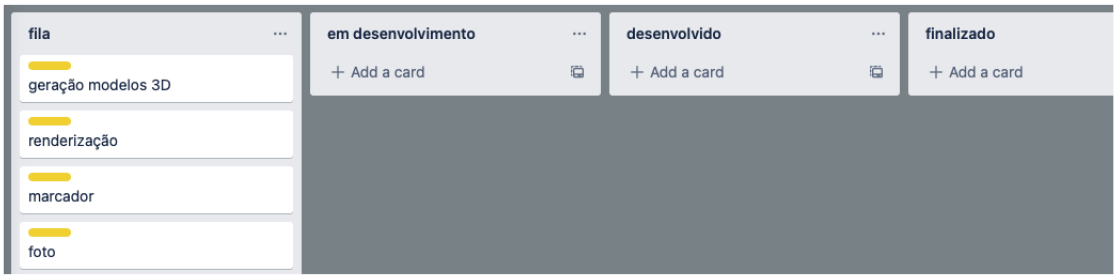

Figura 9 – Retrato do quadro Kanban para implementação do protótipo

Na Figura 10, têm-se o resultado do desenvolvimento do protótipo: na tela à esquerda, a renderização de um modelo 3D simples contendo um marcador (quadrado vermelho) na parede frontal. Na tela à direita, a lista de fotos vinculadas àquele marcador, que é exibida quando há interação com o marcador apresentado no modelo 3D.

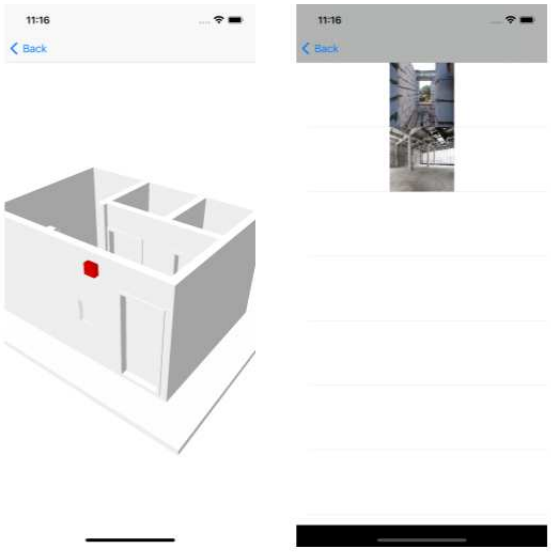

Figura 10 – Telas de Renderização e de Lista de Fotos - protótipo

3.2.3 Obtenção e validação dos requisitos

Após finalizado o desenvolvimento, foi realizada a validação do protótipo. Neste momento, as funcionalidades iniciais foram validadas e discutidas, e novas funcionalidades foram adicionadas ao escopo do aplicativo para incrementar sua relevância com outras ferramentas úteis ao dia-a-dia do acompanhamento de obras:

- **Lista de projetos**: um usuário do aplicativo tende à acompanhar mais de uma obra simultaneamente, principalmente se esse usuário trabalha em empresas de construção. Para permitir que o aplicativo seja utilizado para acompanhamento de mais de um projeto, então, faz-se necessária uma listagem desses projetos.
- **Diário de obra**: esse conceito sintetiza, de fato, o objetivo central do aplicativo. O diário de obra é um documento utilizado para registrar informações importantes sobre as atividades diárias em um canteiro de obras. Essa funcionalidade, agora, estende a utilização do aplicativo por trazer registros mais complexos e com informações de outra natureza, além das fotos integradas no protótipo, como data do registro, etapa e criticidade.
- **Categorização em etapas**: uma das informações do novo registro do diário de obra é a etapa. Os projetos de construção seguem uma lista de etapas e categorizar

os registros nessas etapas provê uma ferramenta de acompanhamento mais completa e detalhada, principalmente, por meio de filtragem, mencionado posteriormente.

- **Categorização em criticidade**: outra adição ao registro é o conceito de criticidade. Uma vez que o diário de obra contém diversos registros, incluir uma escala de criticidade facilita o gerenciamento das atividades por aumentar a visão sobre priorização desses registros incluindo, também, a possibilidade de adicionar notificações específicas para os níveis de criticidade definidos.

- **Filtragem**: a categorização dos registros em etapa e criticidade habilitam uma ferramenta ainda mais útil de filtragem, permitindo a busca mais precisa dos tipos de registros e, consequentemente, uma visualização mais clara dos eventos.

3.3 Desenho

Com base no levantamento de requisitos realizado na validaão do protótipo, foi possível iniciar o processo de modelagem da aplicação. Para tal, novas tarefas foram adicionadas ao quadro *Kanban*, representando as etapas de modelagem necessárias para obter uma visão detalhada do objetivo do aplicativo, conforme Figura 11.

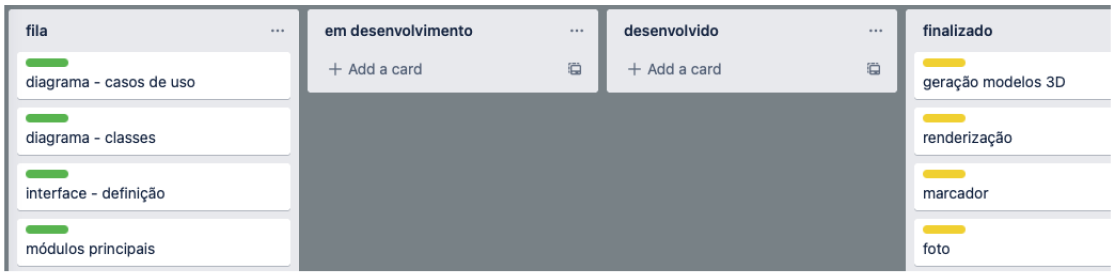

Figura 11 – Retrato do quadro Kanban para modelagem do aplicativo

3.3.1 Modelagem da solução e construção das tarefas

A modelagem da solução final é um passo importante para o planejamento do desenvolvimento, por permitir uma melhor visualização do sistema como um todo, ajudando a prever e antecipar possíveis problemas. Nessa etapa, então, foram produzidos alguns artefatos para auxiliar a continuação do projeto:

- **Diagrama de casos de uso**: neste diagrama, foram levantadas as operações principais do aplicativo, considerando o ponto de vista do usuário. O resultado (Figura 12) apontou os seguintes casos de uso:

  – **Listar projetos**: exibir os projetos disponíveis para o usuário, apresentando um indicador de alta criticidade além do título e do ícone do projeto;

  - **Visualizar renderizações**: exibir as renderizações relacionadas ao projeto de forma que seja possível alternar entre elas, caso exista mais de uma;
  - **Filtrar marcadores**: filtrar os marcadores considerando as informações de etapa e/ou criticidade dos registros vinculados e atualizar as renderizações para exibir, apenas, a lista de marcadores gerada pelo filtro;
  - **Listar marcadores**: exibir os marcadores adicionados às renderizações. Essa operação está inclusa na visualização das renderizações;
  - **Adicionar marcador**: criar um novo marcador e vinculá-lo à um ou mais renderizações. Obrigatoriamente, deve ser fornecido um título para identificação dp marcador;
  - **Remover marcador**: excluir um marcador de todas as renderizações relacionadas. Os registros criados por este marcador, porém, devem ser mantidos no diário de obras;
  - **Listar registros**: exibir os registros do diário de obras completo ou de um marcador específico, apresentando as informações básicas do registro e um indicador de alta criticidade;
  - **Adicionar registro**: criar um novo registro, preenchendo as informações: título (obrigatório), data do registro (inicializada com a data corrente), foto da galeria (opcional), etapa, criticidade e campo de texto opcional para outras informações relevantes;
  - **Remover registro**: excluir um registro do diário de obras completo e dos marcadores relacionados;
  - **Anexar foto**: anexar fotos aos registros criados e está inclusa na operação de adicionar um registro.

- **Diagrama de classes**: o diagrama de classes foi construído para facilitar a visualização das entidades da aplicação e suas relações dentro do sistema. O levantamento resultou nos seguintes modelos, conforme diagrama da Figura 13:
  - **Projeto**: é o objeto que simboliza a obra para qual o acompanhamento das atividades será realizado. É o objeto-raíz do sistema, contendo as demais entidades;
  - **Renderização**: referencia o modelo 3D de um projeto que será exibido no aplicativo para visualização e manipulação. Um projeto pode ter 1 ou mais renderizações associadas;
  - **Marcador**: é o objeto que armazena uma marcação em uma renderização e indica que há um ou mais registros associados àquela região do modelo 3D;

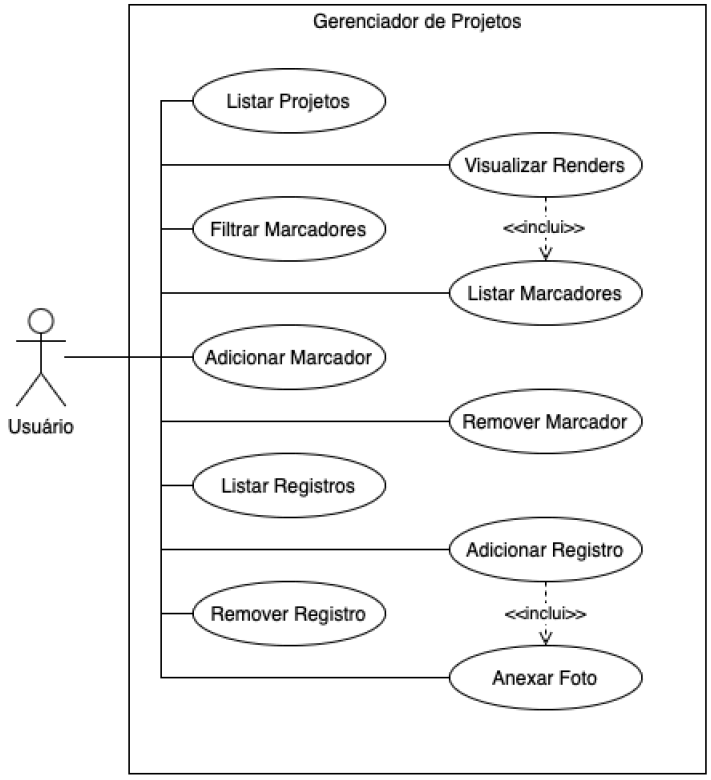

Figura 12 – Diagrama de casos de uso

- **Coordenada**: indica a posição do marcador no espaço do modelo 3D em um plano cartesiano de três dimensões;
- **Registro**: armazena as informações dos eventos relevantes para o acompanhamento da obra. A listas de registros de um projeto constitui o diário de obras;
- **Criticidade**: representa o nível de importância do registro para o acompanhamento/resultado da obra;
- **Etapa**: representa as etapas planejadas para a execução da obra e às quais os registros são relacionados.

- **Definição da experiência do usuário**:

  A jornada do aplicativo começa na tela de *Login*, na qual a autenticação do usuário garante acesso aos projetos e as informações posteriores. Em seguida, o usuário é redirecionado à tela com a lista dos projetos. Nesta lista, os projetos com registros

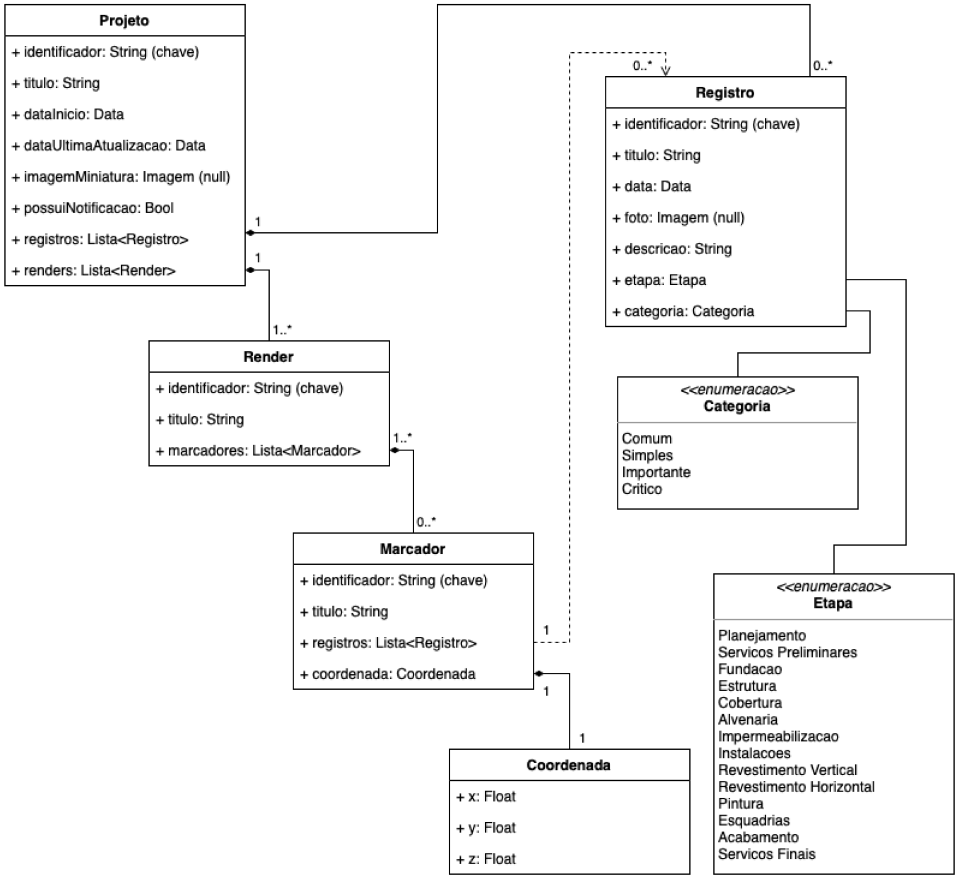

Figura 13 – Diagrama de classes

críticos possuem uma indicação (ícone amarelo) para alertar sobre pontos relevantes de acompanhamento. As duas telas são apresentadas na Figura 14. Ao selecionar um projeto, é apresentada a tela central do aplicativo: tela de renderização, conforme exibido na Figura 15. As ações disponíveis nessa tela são:

- **Navegar entre renderizações**: o usuário visualiza o identificador da renderização exibida e pode navegar entre as demais renderizações adicionadas ao projeto;
- **Visualizar registros**: o primeiro botão, no canto superior direito, abre a lista total de registros do projeto;
- **Filtrar marcadores**: o segundo botão, no canto superior direito, abre a lista de filtros de etapa e criticidade para filtragem dos marcadores (tela ao centro da Figura 17);

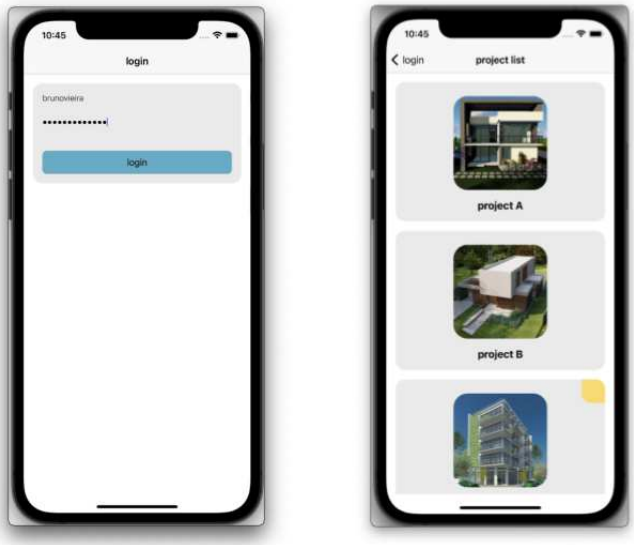

Figura 14 – Telas de Login e Lista de Projetos

– **Adicionar marcador**: o toque duplo em um ponto do modelo 3D inicia a jornada de adição de um marcador (tela à esquerda da Figura 17);
– **Visualizar registros de um marcador**: o toque duplo em um marcador abre a lista de registros vinculados ao marcador selecionado.

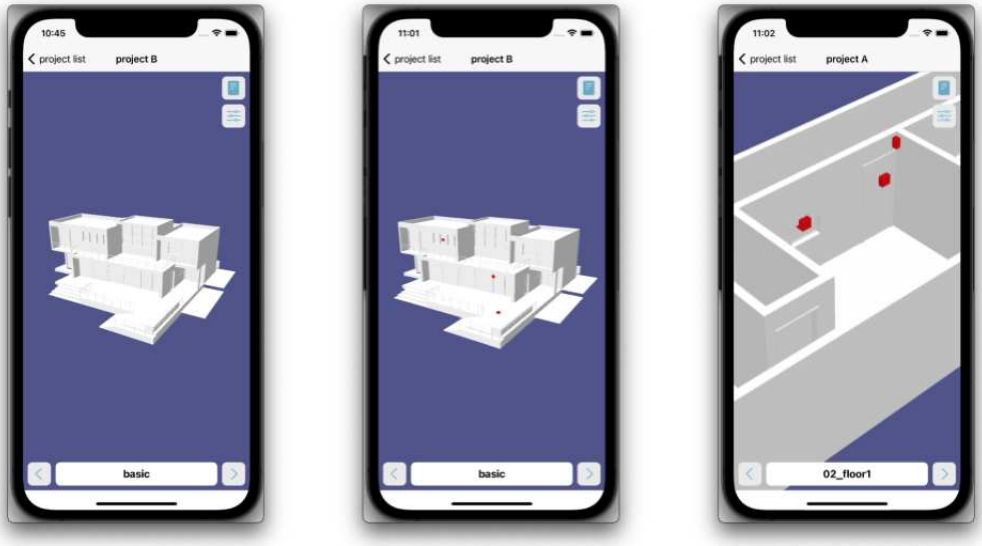

Figura 15 – Telas de Renderização

A tela lista de registros (ao centro da Figura 16) contém as operações referentes ao gerenciamento dos registros. São elas:

– **Adicionar registro**: o botão na barra de navegação redireciona à tela de adicionar registro (à esquerda da Figura 16). Caso a lista sendo exibida seja

de um marcador, o registro será vinculado ao marcador selecionado;

- **Remover registro**: deslizar um item para a esquerda exibe a opção de remover um registro, conforme exibido na tela ao centro da Figura 16;
- **Visualizar foto**: o toque na foto exibe uma tela para visualização da foto (à direita da Figura 16);
- **Remover marcador**: caso a lista sendo exibida seja de um marcador, o botão na parte inferior da tela (à direita da Figura 17) remove o marcador selecionado.

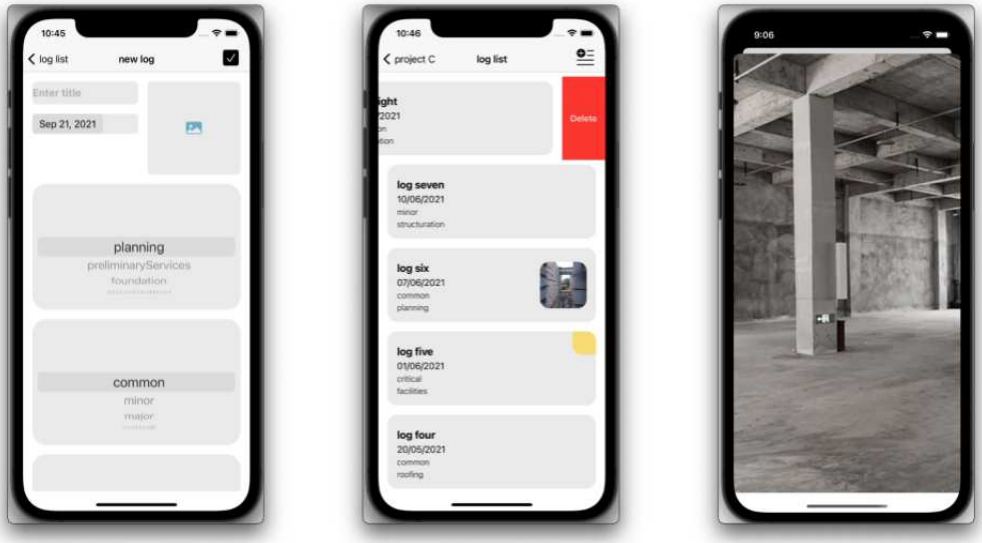

Figura 16 – Telas de Adicionar Registro, Remover Registro e Visualizar Foto

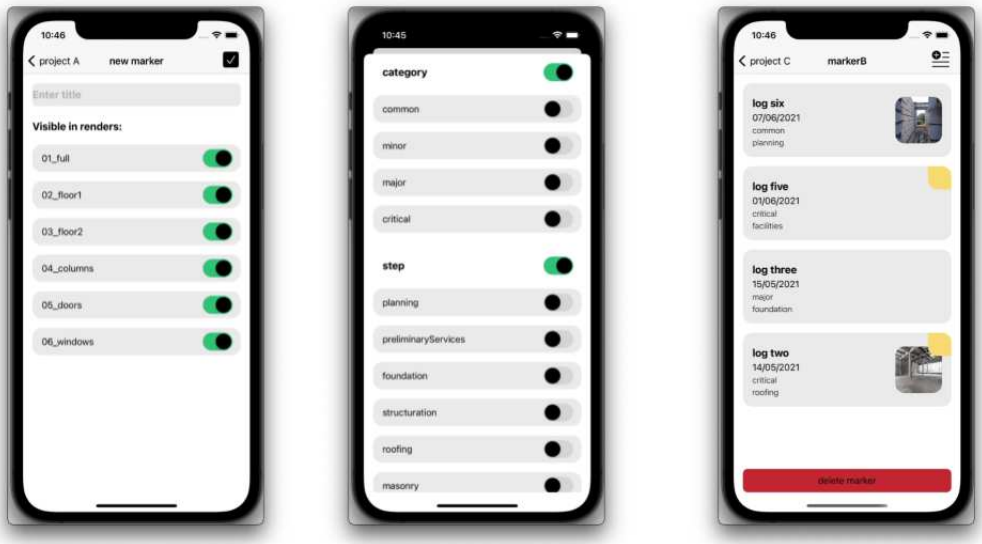

Figura 17 – Telas de Adicionar Marcador, Filtrar Marcadores e Remover Marcador

- Módulos principais:

- ***Core***: o módulo *Core* contém utilidades de código que podem ser reutilizadas por toda a aplicação, evitando duplicação de código e facilitando a manutenção de algumas tarefas mais objetivas, como conversão de arquivos *JavaScript Object Notation (JSON)*, formatadores de data e conversores de tipo de dado.
- ***Design System***: conforme mencionado na seção 2.2.1, o foco em DS desse projeto foi a customização/estilização do aplicativo para diferentes clientes, por meio da alteração da paleta de cores, tipografia, iconografia e ilustrações. Para tal, este módulo contém as definições dessas informações, bem como suas utilizações para a construção dos elementos de interface, como botões, campos de texto e imagens.
- ***Networking***: este módulo contém a implementação das funcionalidades necessárias para realizar requisições para o back-end. Em um primeiro momento, contém respostas fictícias dessas requisições para simular essas interações mas, futuramente, será substituído por um novo módulo de *networking* contendo as interações reais após o desenvolvimento do *backend*.

3.3.1.1 Definição das tarefas

A modelagem da solução simplifica a visibilidade da sequência das tarefas necessárias para o desenvolvimento. A Figura 18 apresenta a fila de tarefas, que foi novamente populada com base no levantamento das funcionalidades do aplicativo em preparação para o início do desenvolvimento.

3.4 Implementação

O foco do primeiro ciclo de desenvolvimento foi a estruturação do projeto e a implementação das funcionalidades principais do aplicativo, uma vez que o protótipo desenvolvido anteriormente foi descartado. As atividades dessa etapa ocorreram conforme a seguinte sequência:

1. **Criação do repositório principal**: criação de um novo repositório na ferramenta *Github* para armazenamento e gerenciamento do código fonte do projeto principal;
2. **Inicialização do projeto principal**: criação do projeto na ferramenta *Xcode* nos parâmetros definidos no início do projeto, considerando o *iOS* 13 como requerimento mínimo para a execução do aplicativo;
3. **Configuração dos módulos**: uma vez que os módulos são conjuntos de código reutilizáveis, uma possível abordagem consiste em mantê-los separados do projeto principal. Para tanto, foram criados novos repositórios para gerenciamento dos códigos modulares e integração com o *Swift Package Manager*;

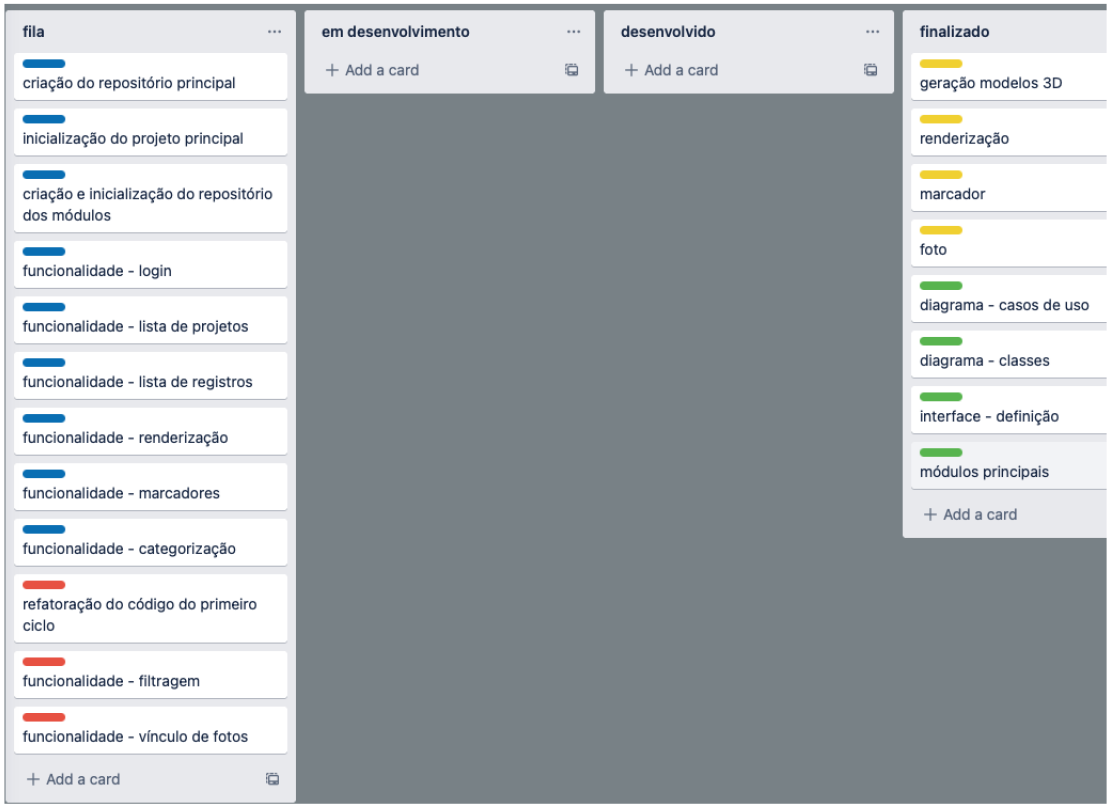

Figura 18 – Retrato do quadro Kanban para implementação das funcionalidades

4. **Codificação**: desenvolvimento das funcionalidades *Login*, Lista de Projetos, Lista de Registros, Renderização, Marcadores e Categorização. A arquitetura MVVM foi escolhida para o desenvolvimento, uma vez que o projeto, inicialmente, apresenta caracterísicas que favorecem sua utilização, como baixa complexidade de regras de manipulação de dados e foco na camada de visualização das informações.

3.4.1 Arquitetura MVVM

Embora seja uma arquitetura recomendada pelo próprio site de desenvolvedores *Android*, uma derivação da arquitetura MVVM também é bastante utilzada no desenvolvimento *iOS*, pois aplica o conceito de separação de responsabilidades de forma bastante clara, produzindo código mais limpo e com maiores manutenibilidade e escalabilidade, conforme Figura 19.

Aplica-se a arquitetura a cada tela do aplicativo, com a presença opcional de algumas dessas entidades. Essas entidades são:

- ***View***: *views* são entidades que gerenciam a interface do usuário, exibindo o *layout* da aplicação e redirecionando as interações do usuário como ações para serem tratadas pelo *ViewController*.

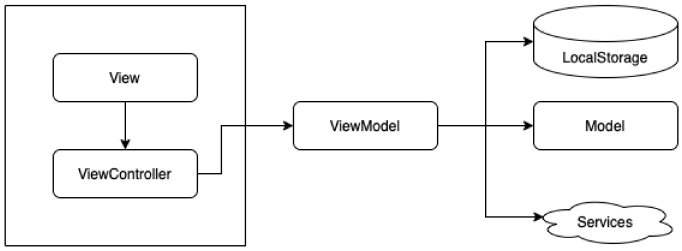

Figura 19 – Arquitetura MVVM para iOS de uma cena

- ***ViewController***: contém as informações que são apresentadas pela *View* que controla. Além disso, é responsável pelo roteamento entre as telas e por requisitar a execução das operações ao *ViewModel*;
- ***ViewModel***: é a entidade mais complexa da arquitetura por centralizar informações das demais entidades. Contém as regras de negócio e de apresentação dos dados, pois possui acesso à camada *Model*. Também reage às requisições do *ViewController*, redirecionando operações para o *LocalStorage* ou para os *Services*;
- ***Model***: os modelos são os objetos modelados com as informações que são trafegadas pela aplicação;
- ***LocalStorage***: é a entidade responsável pelo gerenciamento do armazenamento de informações no dispositivo. Realiza operações de leitura e escrita dessas informações. É uma entidade opcional;
- ***Services***: realizam operações nos serviços para buscar dados ou salvar informações em um servidor. É uma entidade opcional.

A divisão de responsabilidades descrita por essa arquitetura facilita a criação de testes unitários para validação de qualidade do aplicativo, o que favorece a integridade da solução a longo prazo.

3.4.2 Segundo ciclo de desenvolvimento

A última etapa planejada para o desenvolvimento do aplicativo contém o implementação das últimas duas funcionalidades mapeadas pelo diagrama de casos de uso: Filtragem e Vínculo de Fotos. Além disso, nessa etapa, também foram realizadas pequenas refatorações de código para melhorar a qualidade do código previamente desenvolvido ou realizar pequenos ajustes apontados na validação realizada.

Após o fim do segundo ciclo de desenvolvimento, uma nova reunião de validação do aplicativo com o cliente foi realizada. Nessa apresentação, também foram levantadas

novos requisitos, que não serão implementados no escopo deste trabalho, mas serão parte da lista de funcionalidades para o prosseguimento do projeto.

4 Conclusão

Neste capítulo, é apresentada a conclusão deste trabalho. A seção 4.2 aborda os trabalhos futuros para expansão do aplicativo desenvolvido, a seção 4.1 compara a solução desenvolvida com algumas soluções de mercado. Por fim, a seção 4.3 apresenta possibilidades de aplicações deste trabalho em outras áreas além da Engenharia Civil.

Este trabalho solidifica a viabilidade da construção de um aplicativo simples, prático e útil para auxílio no processo de acompanhamento de uma obra. A interface objetiva favorece o acesso rápido das principais funções do aplicativo, o que aumenta seu potencial de utilização uma vez que, quanto mais objetiva é a experiência do usuário, menor é a barreira de inclusão da ferramenta em seu dia-a-dia.

Embora a aplicação tenha implementado apenas a utilização local das informações, o número de melhorias e funcionalidades levantadas durante o processo de validação, conforme apresentado adiante na seção 4.2.1, simbolizam a necessidade e o interesse de profissionais da área em uma solução com tal objetivo.

O aplicativo desenvolvido permite a importação de modelos 3D associados ao gerenciamento de uma obra, fornecendo os controles de navegação necessários para que os pontos de interesse sejam marcados e armazenados no modelo. Além disso, o diário de obra permite a adição de registros, contendo dados relevantes, incluindo fotos, ao projeto. Por fim, o sistema de filtragem facilita a busca desses registros posteriormente.

Essas funcionalidades permitem uma melhor organização e armazenamento das informações relevantes ao acompanhamento da obra, com o potencial de colaborar com a qualidade e eficiência do projeto de engenharia civil.

4.1 Comparação com soluções disponíveis

O resultado da implementação do aplicativo proposto neste trabalho permite uma comparação com demais aplicativos disponíveis no mercado em relação à funcionalidades e usabilidade. Alguns destes são mencionados a seguir:

- *Diário de Obra*[1]: este aplicativo possui foco na utilização de um diário de obras digital e compartilhado, com funcionalidades como criação de tarefas e vínculo de fotos. A aplicação, porém, não possui recursos de auxílio visual do projeto;

[1] https://diariodeobras.net

- *SiteWorks*[2]: com foco em visualização 2D, esta solução faz uso de arquivos em formato *Portable Document Format (PDF)*, nos quais o usuário pode adicionar marcações de pontos de interesse e vincular tais pontos à tarefas. A visualização 2D, entretanto, possui controles de navegação mais simplificados e fornecem menos informações em comparação à visualização 3D;

- *ConstruCode*[3]: é uma ferramenta completa e possui funcionalidades como um diário de obra digital e adição de anotações em pontos específicos dos documentos 2D importados na ferramenta. Porém, possui foco em disponibilizar detalhes de execução de tarefas específicas, categorizadas e apresentadas em forma de uma etiqueta com códigos escaneáveis adicionados ao canteiro de obras.

Existem diversos aplicativos móveis disponíveis com o objetivo de auxiliar o dia-a-dia do acompanhamento de obras. Alguns destes possuem funcionalidades exclusivas para resolver problemas específicos, ao passo que também fornecem os recursos básicos que abrangem grande parte das necessidades. Este trabalho apresenta o primeiro passo de uma nova solução e a evolução do aplicativo em trabalhos futuros tende a aumentar sua usabilidade, sua qualidade e, por consequência, sua competitividade em relação à outros aplicativos de mercado.

4.2 Trabalhos futuros

O escopo deste projeto teve, como foco, a implementação do aplicativo *iOS*, mas pode ser expandido para acrescentar outros sistemas na aplicação, conforme Figura 20:

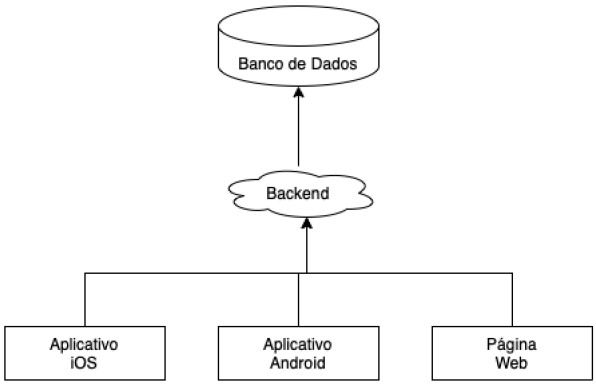

Figura 20 – Peças do sistema completo

[2] https://siteworks.io
[3] https://site.construcode.com.br

- ***Backend***: os aplicativos de frontend possuem funcionalidade limitada quando não estão conectados à um *backend* para armazenamento seguro das informações em bancos de dados e processamento de operações mais complexas. Desta forma, é essencial que um sistema *backend* seja desenvolvido para que o aplicativo expanda sua utilidade;

- **Desenvolvimento *Android* e *Web***: com o *backend* desenvolvido, também existe a possibilidade de expandir o sitema para outras plataformas, aumentando o alcance de usuários e tornando o sistema ainda mais útil;

- **Publicação**: por fim, com as demais peças do sistema desenvolvidas, é necessária a publicação dos aplicativos nas lojas para *Android/iOS* e a hospedagem da página *Web*.

4.2.1 Funcionalidades

Conforme mencionado na seção 3.4, ao fim do desenvolvimento do aplicativo, diversas funcionalidades foram levantadas como melhorias para tornar a aplicação mais útil e completa para o público alvo. São elas:

- **Extração de relatórios**: uma funcionalidade bastante útil para usuários que trabalham em empresas de construção civil é a extração de relatórios. Uma vez desenvolvido o backend com um banco de dados para armazenar os dados cadastrados pelo aplicativo, as informações estarão em um ambiente favorável para a consolidação e exportação de relatórios que podem acelerar o processo de comunicação entre os envolvidos na gestão do projeto. “Frequência de eventos críticos” e “etapas com maior incidência de ocorrências” são relatórios que podem ser úteis em tomadas de decisão relevantes para a continuação do projeto e/ou projetos posteriores;

- **Captura de fotos pelo aplicativo**: também visando uma melhoria prática na utilização do sistema, pode-se utilizar diretamente a câmera do dispositivo para tirar fotos dentro do próprio aplicativo ao invés de exigir o passo extra de tirar fotos pelo aplicativo de câmera e vinculá-las aos registros pela galeria;

- **Painel de pendências**: para facilitar ainda mais a visualização dos registros de pendência, pode-se adicionar uma tela inicial contendo os pontos mais importantes, exibindo as notificações críticas, as datas limites mais próximas, progresso da obra e outras informações úteis ao acesso rápido da aplicação.

- **Download dos arquivos 3D**: o desenvolvimento do *backend* possibilitaria importar os arquivos de renderização 3D de um servidor, ao invés do processo manual realizado atualmente para importação direta no aplicativo iOS;

- **Painel financeiro**: parte fundamental da gestão de um projeto é a gestão das finanças. Pode-se extender a utilização dos registros desenvolvidos até o momento com informações sobre os custos como, por exemplo, "qual foi o valor gasto para corrigir uma falha crítica do projeto na etapa de alvenaria" ou "qual foi a economia em adquirir materiais para a etapa de pintura". Essas informações podem ser adicionadas à um sistema de painel, contendo gráficos e referências do orçamento do projeto para que o gestor tenha maior visibilidade dos eventos relacionados a esse tema;

- **Vínculo de contatos**: vincular contatos da agenda do dispositivo aos registros criados no diário de obras pode trazer praticidade ao ter o contato rapidamente disponível para chamadas, criando uma forma mais ágil de colher informações ou resolver problemas em aberto;

- **Resolução de registro pendente**: dentro do diário de obra podem existir registros que necessitam de resolução, como um registro crítico de "compra de materiais sem estoque". Essa ideia traz a necessidade de adicionar a possibilidade de encerrar um registro dado como resolvido e, consequentemente, atualizar as notificações dos projetos;

- **Data limite para resolução**: extendendo a funcionalidade de resolução, uma nova informação que indique uma data limite ou objetivo para a resolução de uma pendência pode ajudar a organização e priorização das atividades por parte do usuário que está fazendo o acopanhamneto de uma obra;

- **Integração com *Building Information Modeling (BIM)***: BIM é um modelo virtual que contém todas as informações referentes à uma construção, contendo detalhes precisos de cada atributo do projeto. A integração com esse modelo traz maior riqueza de dados para o aplicativo, que poderiam ser utilizados desde o momento da modelagem 3D ao levantamento financeiro do projeto.

4.3 Aplicação em outras áreas

Este aplicativo foi desenvolvido como solução para o requisito de adicionar marcações em pontos de interesse em modelos 3D no contexto da Engenharia Civil. Essa abordagem pode ser aplicada, também, à outros contextos, desde que o objeto de trabalho seja um modelo 3D.

Uma aplicação da área de Biologia, por exemplo, pode utilizar o mesmo conceito para adicionar marcadores em modelos 3D de órgãos do corpo humano. Uma aplicação da área de Geografia pode adicionar marcadores em modelos 3D de regiões montanhosas ou de análises ambientais.

Espera-se que este trabalho contribua com os detalhes dos desafios técnicos da aplicação dos conceitos de modelagem 3D e marcação de pontos de interesse para o fomento de novas ideias em outras áreas de atuação.

Referências

ANDERSON, D. J. *Kanban: Successful Evolutionary Change for Your Technology Business*. [S.l.]: Blue Hole Press Inc, 2013. v. 1. Citado na página 19.

APPLE. *ModelIO*. 2021. <https://developer.apple.com/documentation/modelio/mdlasset/1391813-canimportfileextension>. Acessado em 23/06/2021. Citado na página 21.

APPLE. *SceneKit*. 2021. <https://developer.apple.com/documentation/scenekit>. Acessado em 20/06/2021. Citado na página 17.

APPLE. *UIKit*. 2021. <https://developer.apple.com/documentation/uikit>. Acessado em 20/06/2021. Citado na página 17.

APPLE. *Xcode*. 2021. <https://developer.apple.com/documentation/xcode>. Acessado em 20/06/2021. Citado na página 17.

AUTODESK. *Export a View to an Image File*. 2021. <https://knowledge.autodesk.com/support/revit-lt/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/RevitLT-DocumentPresent/files/GUID-CDB40C22-EDA3-4FFE-B0A1-AF9A89CF5E50-htm.html>. Acessado em 07/10/2021. Citado na página 21.

AUTODESK. *Software Revit*. 2021. <https://www.autodesk.com.br/products/revit/overview>. Acessado em 07/10/2021. Citado na página 16.

CHOUDHARY, A. *Defining colors on your Design System*. 2021. <https://uxdesign.cc/defining-colors-in-your-design-system-828148e6210a>. Acessado em 15/08/2021. Citado na página 16.

COSTA, J. D. da. *Aplicação na construção civil de técnicas e ferramentas de planejamento e controle, baseados no conceito da construção enxuta*. 57 p. Monografia (Bacharelado) — Escola Politécnica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2016. Citado na página 7.

GIT. *Git*. 2021. <https://git-scm.com>. Acessado em 25/06/2021. Citado na página 16.

GITHUB. *Github*. 2021. <https://github.com/features>. Acessado em 25/06/2021. Citado na página 17.

HASSU, T. *Design System na visão da Meiuca*. 2021. <https://medium.com/meiuca/1-design-system-na-vis~ao-da-meiuca-cc0d67aa8d1b>. Acessado em 10/07/2021. Citado na página 15.

IBGE, D. d. I. Pesquisa anual da indústria da construção. v. 29, 2020. Acessado em 18/10/2021. Citado na página 7.

KHOLMATOVA, A. *Design Systems*. [S.l.]: Smashing Media AG, 2017. v. 1. Citado na página 15.

ROBERT, C. M. *Clean Code: A Handbook of Agile Software Craftsmanship*. [S.l.]: Pearson, 2008. v. 1. Citado na página 11.

ROBERT, C. M. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. [S.l.]: Pearson, 2017. v. 1. Citado na página 12.

SACCOMANI, P. *Native Apps, Web Apps or Hybrid Apps? What's the Difference?* 2019. <https://www.mobiloud.com/blog/native-web-or-hybrid-apps#6>. Acessado em 06/10/2021. Citado na página 10.

SHARMA, S. *Native vs Hybrid vs Cross-Platform — What To Choose?* 2020. <https://medium.com/flutterdevs/native-vs-hybrid-vs-cross-platform-what-to-choose-3221130f7cc5>. Acessado em 05/10/2021. Citado na página 10.

SILVA, M. S. T. C. *Planejamento e Controle de Obras*. 98 p. Monografia (Bacharelado) — Escola Politécnica, Universidade Federal da Bahia, Salvador, 2011. Citado na página 7.

SOMMERVILLE, I. *Engenharia de Software*. [S.l.]: Pearson Prentice Hall, 2011. v. 9. Citado 3 vezes nas páginas 17, 18 e 19.

SWIFT. *Swift Package Manager*. 2021. <https://swift.org/package-manager>. Acessado em 20/06/2021. Citado na página 17.