Carlos Rafael de Oliveira do Amaral Leitão

Aplicativo móvel de terapia digital para a formação de hábitos alimentares saudáveis e monitoramento de perda de peso em mulheres de 40 a 60 anos.

Trabalho de Graduação

Universidade Federal de Pernambuco
graduacao@cin.ufpe.br
www.cin.ufpe.br/~graduacao

RECIFE

2017

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

Carlos Rafael de Oliveira do Amaral Leitão

Aplicativo móvel de terapia digital para a formação de hábitos alimentares saudáveis e monitoramento de perda de peso em mulheres de 40 a 60 anos.

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau em Ciência da Computação

Orientador: Cristiano Coelho de Araújo

RECIFE

2017

Agradecimentos

Este trabalho é o resultado de toda uma jornada acadêmica com todas as conquistas e dificuldades que estão intrínsecas a ela. Durante essa trajetória recebi o apoio incondicional dos meus pais e da minha irmã e por isso os agradeço, primeiramente. Vocês estavam sempre comigo comemorando as vitórias e me ajudando nos momentos de dificuldades, principalmente no começo do curso.

Agradeço também a todos os amigos que fiz ao longo do caminho em que compartilhamos momentos juntos, desde projetos, estudos em grupos a brincadeiras, porque sem eles também não conseguiria chegar até aqui. Em especial um agradecimento a Bruno César por todos os projetos que fizemos e a Juliane Sabrina por todo o tempo passado estudando juntos para diversas cadeiras e nessa reta final para terminar os nossos trabalhos.

Durante o caminho acadêmico, surgiu uma nova trajetória na minha vida, a empreendedora, e nesta um agradecimento grande vai à Felipe Farias. Desde projetos anteriores até o momento atual na Eleve, o convívio e aprendizado em conjunto foi grande e agora é só o começo dessa nova trilha. Obrigado também a Fernanda, nossa sócia, assim como todos que fazem parte da Eleve, e em especial a Geovane por desenvolvermos juntos este aplicativo.

Agradeço à Cristiano Coelho por ter me apoiado e orientado durante este processo e ter revisado este trabalho de graduação e à todos os amigos e familiares que me acompanharam nesta trajetória.

Resumo

Este trabalho de graduação apresenta o desenvolvimento, utilizando metodologias ágeis, de um aplicativo móvel que permite a um *coach* que pode ser o médico, nutricionista ou educador físico realizar o acompanhamento de turmas de pacientes, tipicamente mulheres entre os 40-60 anos, com sobrepeso/obesidade, que estão em um programa para perda de peso média estimada em 4-6% da massa corporal do indivíduo, ao longo de 16 semanas de terapia. Este programa de terapia digital foi concebido pela startup Eleve e vem sendo validado com turmas de mulheres com o perfil de usuário que desejam atingir a redução de peso de forma sustentável. Esta solução digital é necessária pois a obesidade é uma doença crônica que afeta uma parcela significativa da população brasileira e cujas complicações comprometem não apenas a saúde e bem-estar do paciente, mas também traz grandes custos para o sistema de saúde. Em uma parcela significativa dos casos, a obesidade pode ser prevenida, desde que os indivíduos com sobrepeso ou obesos aprendam a reeducar seus hábitos alimentares e de exercício físico. Isso significa que perder peso de forma saudável exige mudanças permanentes, e difíceis de serem conseguidas, na rotina da pessoa. Essas dificuldades levam na prática a um alto índice de evasões em tratamentos, assim como regressões dos resultados Um fator que influencia na mudança de comportamento é o acompanhamento contínuo por um profissional qualificado, porém o médico ou nutricionista não tem tempo para acompanhar e instruir cada paciente individualmente. Para a validação do programa de terapia digital estavam sendo utilizadas diversas ferramentas como Telegram que foram adaptadas para que o *coach* fosse capaz de realizar o acompanhamento. A partir destas validações iniciais foi iniciado o desenvolvimento de um aplicativo dedicado ao monitoramento das pacientes. Para tanto foram avaliadas as metodologias para desenvolvimento ágil e selecionada a metodologia Scrum para o desenvolvimento da aplicação. O aplicativo, também chamado Eleve, foi disponibilizado na loja de aplicativos Google Play para a plataforma Android e testado com 3 participantes de uma turma real que está sendo acompanhada por um *coach.* Os resultados da avaliação nesta turma inicial serão utilizados para o futuro desenvolvimento da terapia digital visando encontrar o product/market fit e uma solução que possa ser repetível e escalável.

Palavras-chave: obesidade, prevenção, tratamento médico, terapia digital, ferramenta mobile.

Lista de Figuras

Sumário

1.Introdução

Doenças crônicas não transmissíveis (DCNTs) foram responsáveis por 74% das mortes no Brasil em 2014 (WHO, 2014), sendo as doenças cardiovasculares, câncer, diabetes e DPOC(Doença Pulmonar Obstrutiva Crônica) as que tiveram a maioria desses óbitos (WHO, 2014) .

A preponderância dessas morbidades está associada a um pequeno conjunto de fatores de risco, destacando-se a má alimentação, inatividade física, fumo e uso prejudicial de álcool, que são percebidos através da elevação de fatores de risco intermediários, como o sobrepeso/obesidade, entre outros (ARMENIO, Letizia et al., 2013).

Para se ter noção de quão grande o problema em questão é, no Brasil, a porcentagem de adultos com sobrepeso ultrapassou os 53,8% segundo o último levantamento do Governo Federal, enquanto que o número de pacientes obesos no país cresceu em 60% nos últimos 10 anos, chegando a 18,9% da população (VIGITEL, 2016). Dados os quais, evidenciam a necessidade do desenvolvimento de políticas e programas escaláveis para inspirar a adoção de hábitos de vida mais saudáveis.

Uma questão importante que decorre disso é que, na maioria dos casos, essas doenças, e seus fatores de riscos intermediários como o sobrepeso/obesidade, podem ser prevenidos através de mudanças comportamentais, que estão associadas aos fatores de risco modificáveis comuns.

Porém, para adultos com hábitos e preferências consolidados, modificar comportamentos alimentares ou aderir a uma atividade física não é um desafio trivial, por mais que a necessidade da mudança seja reconhecida pela pessoa.

Por outro lado, o uso dos smartphones e da tecnologia digital em geral está cada vez mais presente no dia-a-dia das pessoas. Além da maior presença dos smartphones, com o avanço do poder de processamento dos dispositivos móveis, o desenvolvimento de aplicativos está melhorando a interação da população com as mais diversas áreas da sociedade.

Uma das áreas em que houve uma crescente no surgimento de aplicativos é a saúde. O número de indivíduos que usam dispositivos móveis para monitorar e gerenciar sua saúde graças aos milhares de aplicativos disponíveis está agora em centenas de milhões. Dando origem, assim, a um novo modelo de entrega de saúde, o *mHealth*. Segundo a Organização Mundial da Saúde (WHO, 2011), o *mHealth* é definido como a prática médica e de saúde pública suportada por dispositivos móveis, como telefones celulares, dispositivos de monitoramento de pacientes, assistentes digitais pessoais (PDAs) e outros dispositivos sem fio. Os aplicativos de *mHealth* viram um crescimento considerável ao longo dos últimos anos. De acordo com o último relatório de aplicativos de mHealth da Research2Guidance publicado em outubro de 2016, houve um crescimento de 57% no número de aplicativos desse setor publicados em grandes lojas de aplicativos no mesmo ano (RESEARCH2GUIDANCE, 2016).

Com a consolidação do setor de mHealth, houve o surgimento de uma tendência em como alguns aplicativos entregam serviço de saúde, a terapia digital. Terapia digital é uma nova abordagem ao tratamento médico onde o uso da tecnologia pode substituir ou auxiliar a terapia convencional de alguma condição médica ou psicológica.

Essa abordagem consiste em programas imersivos que atuam de forma confiável e remota para mudar os comportamentos individuais, e a fim de alcançar resultados clínicos positivos consegue efetuar mudanças comportamentais e de estilo de vida, geralmente estimuladas por um conjunto de intervenções digitais. Também, devido à natureza digital da metodologia, os dados podem ser coletados e analisados, utilizando *machine learning* e outras técnicas de IA, gerando relatórios de progresso e sugerindo medidas preventivas, tornando-se, assim, um tratamento mais efetivo.

1.1 Motivação

Assim, nesse contexto é que foi criada a Eleve. Uma startup cuja primeira solução a ser desenvolvida é uma terapia digital comportamental que traga maior adesão e eficácia ao processo de emagrecimento. Esta deve ser uma solução escalável para a perda de peso sustentável entre pessoas com sobrepeso e/ou que possam apresentar elevado risco de desenvolver doenças crônicas.

Para tanto, o programa da Eleve é fundamentado em quatro áreas da medicina comportamental: melhora da alimentação, prática de atividade física, melhora do sono e redução de estresse que se dividem em quatro módulos com 16 semanas ao todo.

Ao longo do programa, as participantes recebem conteúdos abordando cada uma das quatro áreas trabalhadas pelo programa, provendo conhecimento e abordagens práticas na reeducação. As participantes são inseridas em grupos onde todos possam se apoiar e trocar experiências e é apresentada a um orientador que faz o acompanhamento durante todo o programa.

Além disso, uma parte muito importante para o programa e para a reeducação alimentar é a utilização de um diário. Vários estudos mostraram que pessoas que mantêm diários alimentares são mais propensas a ter sucesso na perda de peso e na manutenção.

Estudos como (HOLLIS, Jack F. et al, 2008) viram que pessoas que mantiveram um diário alimentar seis dias por semana perderam cerca de duas vezes mais peso do que aqueles que mantiveram registros um dia por semana ou menos. Já outro estudo sobre perda de peso em mulheres com sobrepeso ou obesidade publicado no *Journal of the Academy of Nutrition and Dietetics* descobriu que as mulheres que mantiveram diários alimentares perderam consistentemente mais peso do que aqueles que não o fizeram (KONG, Angela et al, 2012).

Dessa maneira, o desenvolvimento do aplicativo de monitoramento deste trabalho é essencial para que as participantes do programa consigam obter resultados satisfatórios. Só a partir do uso deste diário é que feedbacks semanais poderão ser enviados ajudando a dar direcionamentos na nova rotina das

participantes, uma das intervenções mais importantes que fazemos. E com a evolução da ferramenta poderemos também entregar outros aspectos fundamentais para o programa como o conteúdo e as interações entre os participantes no grupo e entre o orientador e participante.

1.2 Objetivos

O objetivo deste trabalho é o desenvolvimento do aplicativo móvel de monitoramento que dê suporte à intervenção digital do programa de mudança de hábitos e perda de peso da Eleve.

A ferramenta desenvolvida para o programa Eleve será dividida em módulos, sendo o módulo que será abordado por esse trabalho o de monitoramento pelo participante. Com essa primeira versão do aplicativo implementada, o participante poderá registrar e acompanhar três dados fundamentais ao longo do programa: o peso, as refeições consumidas e as atividades físicas feitas.

1.3 Estrutura do documento

Para relatar a implementação do aplicativo, este documento será dividido em 5 capítulos, incluindo este de introdução.

No segundo capítulo faremos uma revisão da literatura abordando algumas metodologias ágeis, que auxiliam no desenvolvimento de software de maneira eficiente, e as principais arquiteturas para desenvolvimento Android. No terceiro será mostrada algumas ferramentas já utilizadas para se fazer o monitoramento no processo de reeducação alimentar. Já no quarto capítulo a implementação do aplicativo será relatada assim como o processo de testes para a melhoria da interação com o mesmo.

Após haver o embasamento teórico para o desenvolvimento e ter relatado a implementação, no capítulo 5 apresentaremos as conclusões obtidas e falaremos sobre trabalhos futuros em cima da ferramenta.

2. Revisão da literatura para desenvolvimento

Existem várias maneiras para um software ser desenvolvido, podendo ou não metodologias e arquiteturas serem utilizadas. Porém um dos requisitos para uma aplicação, e consequentemente a empresa que está por trás da mesma, ser competitiva no mercado é que haja eficiência no seu desenvolvimento.

Para tanto, estruturas e metodologias foram estudadas, aprimoradas e comprovadas como eficientes ao longo dos anos para que no início de novos projetos desenvolvedores possam escolher as que mais se encaixam nos seus requisitos.

Dessa maneira, houve a necessidade de um estudo sobre as metodologias e arquiteturas que estruturam melhor o desenvolvimento de um aplicativo e isso é que será abordado neste capítulo.

Primeiro, será mostrada as diferenças entre o desenvolvimento mobile e de softwares tradicionais, após isso algumas metodologias ágeis serão abordadas para mostrar os seus benefícios no gerenciamento de softwares. Ao fim, as principais arquiteturas utilizadas no desenvolvimento Android serão abordadas.

2.1 Diferenças entre desenvolvimento móvel e tradicional

Segundo (WASSERMAN, 2010), em muitos aspectos, o desenvolvimento de aplicativos móveis é semelhante ao desenvolvimento de software no geral. Questões comuns incluem integração com hardware do dispositivo, bem como preocupações tradicionais de segurança, desempenho, confiabilidade e limitações de armazenamento. No entanto, as aplicações móveis apresentam alguns requisitos, restrições e desafios que são menos comuns em relação às aplicações de software tradicionais, incluindo (WASSERMAN, 2010) :

1. **Potencial interação com outras aplicações** - a maioria dos dispositivos embutidos têm software de fábrica instalados mas os dispositivos móveis

podem ter inúmeras aplicações de fontes variadas, com a possibilidade de interações entre elas.

2. **Manuseio de sensores** - a maioria dos smartphones, incluem um acelerômetro que responde ao movimento do dispositivo, uma tela sensível ao toque que responde a variados gestos, junto com teclados reais ou virtuais, um sistema de posicionamento global(GPS), um microfone utilizável por aplicações que não sejam chamadas de voz, uma ou mais câmeras e múltiplos protocolos de rede.
3. **Famílias de plataformas de hardware e software** - a maioria dos dispositivos embutidos executam código que é personalizado, construído para as propriedades desse dispositivo, mas os dispositivos móveis têm que suportar aplicativos que foram escritos para todos os variados dispositivos que suportam o sistema operacional e também para diferentes versões do sistema operacional. Assim, cabe ao desenvolvedor decidir quais versões do sistema operacional e implementar como o aplicativo dará suporte a cada uma delas.
4. **Interface do usuário** - com softwares em geral, o desenvolvedor pode controlar todos os aspectos da experiência do usuário, porém um aplicativo móvel deve compartilhar elementos comuns da interface do usuário com outras aplicações e deve aderir a diretrizes da interface do usuário desenvolvidos externamente, muitas das quais são implementados nos kits de desenvolvimento de software (SDKs) que fazem parte da plataforma.
5. **Complexidade de testes** - Os testes para aplicativos móveis são mais trabalhosos do que para os softwares tradicionais. Testadores são confrontados com alguns outros fatores que devem ser levados em consideração, por exemplo, as diferenças entre as versões do sistema operacional, os modelos de smartphones e tamanhos de tela e os múltiplos métodos de entrada. Além de que aplicações web móveis(aplicativos que quase sempre requerem uma conexão de dados e que geralmente são projetados para executar em todas as plataformas) são particularmente desafiadoras para testar. Não só elas têm muitos dos mesmos problemas

encontrados no teste de aplicativos da web, mas também têm problemas adicionais associados à transmissão e redes de dados do telefone.

6. **Consumo de energia** - Muitos aspectos de um aplicativo afetam seu uso do poder do dispositivo e, portanto, o tempo de vida da bateria do dispositivo. Os dispositivos dedicados podem ser otimizados para uma vida útil máxima da bateria, mas as aplicações móveis podem, inadvertidamente, fazer uso de recursos que drenam a bateria.

E alguns desafios relacionados ao desenvolvimento de aplicativos móveis mostrados por El-Kassas et al. (2015), são listados:

1. **Recursos limitados dos dispositivos móveis** - Embora sejam mais poderosos do que antes, os smartphones ainda têm limitações como capacidade de processamento, espaço de armazenamento e a conectividade para dispositivos móveis será afetada pelo movimento.
2. **Heterogeneidade de sistemas operacionais móveis** - Há a diferença entre os ambientes operacionais para aplicativos (ou seja, o aplicativo desenvolvido utilizando-se o SDK Android só podem ser executado em dispositivos Android). Assim, para desenvolver a mesma aplicação em diferentes sistemas operacionais (OS), o desenvolvedor precisa aprender a desenvolver sobre esses SDKs de diferentes sistemas usando suas linguagens de programação e APIs ou utilizar soluções *cross-platform*.
3. **A heterogeneidade dos dispositivos pode exigir versões diferentes do mesmo aplicativo** - Diferentes recursos de computação e configurações de hardware de dispositivos devem ser considerados ao desenvolver aplicativos, por exemplo, o tamanho de tela que varia entre os dispositivos, os diferentes métodos de entrada, entre outros.
4. **Manutenção de aplicativos**: atualizações freqüentes da plataforma móvel podem afetar alguns aplicativos tornando-os inutilizáveis na nova versão, sendo, a manutenção e as atualizações desses aplicativos necessárias. Além disso, o gerenciamento de versões é difícil porque os usuários não podem atualizar para a nova versão do aplicativo quando lançado. A manutenção ou as atualizações do aplicativo desenvolvidas para diferentes plataformas

significam que o desenvolvedor repete as mesmas atualizações em todas as versões de diferentes plataformas.

5. **Desenvolvimento entre plataformas**: o desenvolvimento da mesma aplicação para diferentes plataformas significa repetir o mesmo trabalho várias vezes porque cada fornecedor de plataforma fornece aos desenvolvedores diferentes linguagens de programação e ferramentas de desenvolvimento.

2.2 Metodologias para desenvolvimento de projetos

O desenvolvimento de aplicativos mobile teve um crescimento meteórico nos últimos anos (PEREZ, 2017). O número de aplicativos nas lojas virtuais dos dois maiores sistemas operacionais para smartphones (Google Play para Android e App Store para o iOS) está cada vez maior. Um exemplo desse crescimento é a marca atingida pela Google Play de mais de 2.800.000 apps disponíveis na loja em março deste ano (STATISTA, 2017).

Com essa enorme variedade de aplicativos nas lojas virtuais, para um *app* poder realmente se diferenciar de tantos outros disponíveis é preciso que na sua construção haja também um foco na entrega rápida de valor para o usuário. Desse modo, os riscos do desenvolvimento são reduzidos já que o quanto mais rápido um aplicativo é colocado na loja mais cedo será possível avaliá-lo e adaptá-lo às necessidades comerciais e às mudanças de requisitos ao longo do processo.

Assim, surge a necessidade de entender quais metodologias são utilizadas e quais delas propiciam essa adaptabilidade necessária para o desenvolvimento de aplicações em um ambiente tão competitivo.

2.2.1 Metodologia tradicional e a transição para o *Agile*

Para o desenvolvimento de qualquer software com potencial de comercialização é necessário a utilização de alguma metodologia com processos que servirão de base para as etapas do desenvolvimento guiando a evolução do

software das fases iniciais de criação às fases finais de aceitação pelo usuário e manutenção. Esses processos são representados através do ciclo de vida de desenvolvimento de softwares (ou em inglês Software Development Lifecycle, SDLC). O SDLC é um modelo conceitual utilizado no gerenciamento de projetos que descreve os estágios envolvidos em um projeto de desenvolvimento, desde o estudo de viabilidade inicial até a etapa de manutenção da aplicação concluída, e que poderia ser dividido basicamente em 6 fases: Concepção, Design, Desenvolvimento, Estabilização/Testes, Lançamento e Manutenção.

Várias metodologias SDLC foram desenvolvidas para orientar os processos envolvidos, incluindo o modelo de Cascata (que era o modelo SDLC original); desenvolvimento rápido de aplicações (RAD, na siga em inglês); desenvolvimento de aplicações conjuntas (JAD, na siga em inglês); o modelo da fonte; o modelo em espiral, entre outros (ROUSE, 2009).

Como alguns métodos se encaixam melhor em tipos específicos de projetos há uma flexibilidade na estruturação do ciclo de vida e com isso diferentes metodologias foram criadas para guiar os processos envolvidos. Frequentemente, diversos modelos são combinados em algum tipo de metodologia híbrida, porém independentemente do modelo escolhido a documentação é crucial (geralmente feita em paralelo com o processo de desenvolvimento) e na análise final, o fator mais importante para o sucesso de um projeto possa ser o quão próximo ficou o plano seguido daquele que foi traçado inicialmente.

Um bom exemplo de uma versão popular dos modelos de ciclo de vida tradicionais é o modelo em Cascata(do inglês, Waterfall model). Muitas vezes considerada a abordagem clássica do ciclo de vida do desenvolvimento de software, o modelo em Cascata descreve um método de desenvolvimento rígido e linear. O desenvolvimento em Cascata têm metas distintas para cada fase e as mesmas têm que serem completadas para a próxima ser iniciada, não havendo mais volta no processo.

A abordagem do modelo em Cascata funciona muito bem se o processo for previsível e tenha requisitos que não mudem ao longo do desenvolvimento, dessa maneira, um cronograma normalmente é definido com prazos para cada estágio e um produto pode prosseguir através do processo de desenvolvimento.

Na prática, porém, o desenvolvimento em Cascata muitas vezes não atende às expectativas, pois não adota as inevitáveis mudanças e revisões que se tornam necessárias com a maioria dos projetos. Devido ao seu design, essa abordagem não permite aos usuários a flexibilidade para reverter e retrabalhar o resultado da etapa anterior. Assim, uma vez que um aplicativo está na fase de teste, é muito difícil voltar e mudar algo que não foi pensado no estágio do conceito.

Com esses problemas no modelo mais tradicional da organização do desenvolvimento, houve a necessidade da criação de novas metodologias e em 2001 o termo *Agile* foi aplicado a esse novo conjunto (AGILEALIANCE, 2017).

Ágil é forma de gestão e desenvolvimento de software que usa uma abordagem de planejamento e execução iterativa e incremental voltado para processos empíricos (complexos, caóticos ou com muita incerteza, tem mudança ao longo do processo, não são repetitivos e são imprevisíveis) que dividem o problema em produtos menores e que visa entregar software funcionando regularmente. Além disso, visa a aproximação e maior colaboração do time de desenvolvimento com os experts de negócios, comunicação face-to-face ao invés de muita documentação escrita, redução dos riscos associados às incertezas dos projetos, abraçar e responder às mudanças de forma mais rápida e natural e é claro a satisfação final dos clientes (STEFFEN, 2012).

A maioria dos métodos ágeis tentam minimizar o risco através do desenvolvimento de software em períodos de tempo curtos, chamados iterações, que tipicamente duram de uma a quatro semanas. Cada iteração é como um projeto de software em miniatura em si e inclui todas as tarefas necessárias para liberar o mini-incremento com novas funcionalidades: planejamento, análise de requisitos, design, codificação, teste e documentação. No final de cada iteração, o time reavalia as prioridades do projeto.

2.3 Principais metodologias ágeis para desenvolvimento

Pelo termo Ágil se referir a diferentes metodologias e frameworks que desenvolvem software de forma iterativa e incremental, essas metodologias compartilham muito da mesma filosofia, bem como muitas das mesmas

características e práticas. Mas, do ponto de vista da implementação, cada uma tem seu próprio conjunto de práticas, terminologia e táticas. Nas próximas seções são abordadas algumas das principais metodologias utilizadas.

2.3.1 Extreme Programming

A Programação Extrema (XP) é uma abordagem pragmática para o desenvolvimento de software que enfatiza primeiro a satisfação do cliente e adquire uma abordagem incremental para desenvolver o produto, usando testes e revisões contínuas.

A Programação Extrema melhora um projeto de software de cinco maneiras essenciais, que representam os seus valores:

- Comunicação
- Simplicidade
- Feedback
- Respeito
- Coragem

Os programadores extremos comunicam-se constantemente com seus clientes e outros programadores. Eles mantêm seu design simples e limpo. Eles recebem comentários testando seu software a partir do primeiro dia. Eles entregam o sistema aos clientes o mais cedo possível e implementam as mudanças conforme sugerido. Cada pequeno sucesso aprofunda seu respeito pelas contribuições únicas de cada membro da equipe. Com esta base, os programadores extremos são capazes de responder corajosamente a mudanças de requisitos e tecnologia (WELLS, 2009).

XP é apresentado por uma série de práticas e princípios que desenvolvedores devem seguir:

1. **Simplicidade**: A simplicidade é central para o XP. O produto entregue deve ser simples, fornecendo os recursos necessários da maneira mais simples para construir algo real que funcione de forma limitada e que depois possa ser incrementado. O uso da metodologia e da

técnica deve ser simples também; Os desenvolvedores de XP evitam a documentação mais do que *user stories* e casos de teste, a menos que haja uma demonstração convincente de seu valor para o cliente. Os praticantes de XP se esforçam pela elegância e simplicidade em suas práticas de desenvolvimento, levando à refatoração.

2. **Refatoração**: Refatoração é a otimização do código interno e da arquitetura do software e é um elemento-chave do XP. É também uma resposta para evitar os perigos do design iterativo, em que diferentes iterações desenvolvidas sobre o software podem ser mal integradas e internamente incompatíveis, já com a constante refatoração isso não ocorre tanto.
3. **Programação em pares:** Essa prática apresenta a ideia de que o código deve ser escrito por pares de programadores, forçando o programador principal a descrever o código para o outro programador, assim estimulando novas ideias. Em vez de revisões periódicas, a ideia chave da programação em pares é que dois especialistas qualificados podem rever e otimizar o código e as práticas de desenvolvimento de cada um deles.
4. **Testes**: A diferença vital entre as práticas de teste do XP e as práticas tradicionais é que o XP insiste que os casos de teste para todos os recursos sejam desenvolvidos anteriormente, com as *user stories*.
5. **Builds contínuos**: Os profissionais de XP preferem a construção contínua, garantindo compatibilidade e funcionalidade continuamente à medida que o produto é criado.
6. **Cliente sempre disponível**: o XP exige que o cliente seja completamente integrado à equipe de desenvolvimento, disponível para revisar recursos, compilações e testes e avaliar o produto à medida que ele evolui.

2.3.2 Scrum

Scrum é um *framework* de gerenciamento de projetos ágil usada para desenvolver e sustentar produtos complexos, principalmente para projetos de desenvolvimento de software. O Scrum reduz a complexidade para se concentrar na construção de produtos que atendam às necessidades do negócio e entregar os mesmos, de forma incremental e empiricamente. Isso é possível, através do compromisso com as curtas iterações do trabalho, oferecendo novas capacidades do software a cada 2-4 semanas.

Scrum é baseado na teoria empírica de controle de processos, ou empirismo. O empirismo afirma que o conhecimento vem da experiência e toma decisões com base no que é conhecido. Scrum emprega uma abordagem iterativa e incremental para otimizar a previsibilidade e controlar o risco (SUTHERLAND; SCHWABER, 2016).

Três pilares sustentam todas as implementações do controle empírico de processos e assim também o Scrum: transparência, inspeção e adaptação (SUTHERLAND; SCHWABER, 2016).

1. **Transparência** - Aspectos importantes do processo devem ser visíveis para aqueles responsáveis pelo resultado. A transparência exige que esses aspectos sejam definidos com um padrão comum para que os observadores compartilhem uma compreensão comum do que está sendo visto.
2. **Inspeção** - Os usuários do Scrum devem freqüentemente inspecionar os artefatos do Scrum e progredir em direção ao objetivo da Sprint para detectar variações indesejáveis. Porém, as inspeções não devem ser tão freqüentes que interfiram no trabalho e devem ser realizadas por inspetores especializados.
3. **Adaptação** - Se um inspetor determinar que um ou mais aspectos de um processo se desviam de limites aceitáveis, o processo ou o material sendo processado deve ser ajustado. Um ajuste deve ser feito o mais rápido possível para minimizar o maior desvio.

O funcionamento do Scrum se baseia na definição e nas regras que ligam os 3 principais aspectos: time, eventos e artefatos.

A. Times do Scrum

Um time no Scrum é composto por um ***Product Owner***, a equipe de desenvolvimento e um ***Scrum Master***. As equipes do Scrum são auto-organizadas e multifuncionais. Equipes auto-organizadas escolhem a melhor forma de realizar seu trabalho, ao invés de ser dirigido por outros fora da equipe, enquanto que as equipes multifuncionais têm todas as competências necessárias para realizar o trabalho sem depender de outros de fora da equipe. O modelo da equipe em Scrum foi projetado para otimizar flexibilidade, criatividade e produtividade (SUTHERLAND; SCHWABER, 2016).

Os times no Scrum entregam produtos de forma iterativa e incremental, maximizando oportunidades para *feedbacks* e assegurando que uma versão potencialmente útil de produto esteja sempre disponível.

O ***Product Owner*** é responsável por maximizar o valor do produto e o trabalho da equipe de desenvolvimento. Por ser a única pessoa responsável pela gestão do *product backlog,* que é uma lista contendo todas as funcionalidades desejadas para o produto, o *Product Owner* está focado na compreensão dos requisitos comerciais e de mercado, priorizando o trabalho a ser feito pela equipe de desenvolvimento de acordo com os requisitos. Entre suas funções estão:

- Criar e gerenciar o *product backlog*;
- Ordenar os itens no *product backlog* para melhor alcançar metas e missões;
- Estar sempre em contato com o negócio e a equipe para garantir que todos entendam os itens de trabalho no *product backlog*;
- Dar à equipe uma orientação clara sobre quais recursos devem ser os próximos a serem entregues;

O ***Scrum Master*** é responsável por garantir que Scrum seja entendido e seja feito de forma correta. *Scrum Masters* fazem isso assegurando que a equipe adira à teoria, práticas e regras do Scrum.

Eles treinam a equipe, o *product owner* e a equipe de negócio no processo do Scrum e buscam maneiras de melhorar sua prática. Um bom *Scrum Master* compreende profundamente o trabalho que está sendo feito pela equipe e pode ajudar a equipe a otimizar seu fluxo de entrega.

Os *Scrum Masters* também procuram resolver impedimentos e distrações para a equipe de desenvolvimento, isolando-os de interrupções externas sempre que possível. Parte importante do trabalho do *Scrum Master* também é evitar um anti-padrão comum entre equipes novas no Scrum: mudar o escopo do *sprint* depois que ele já começou. Às vezes *product owners* querem sugerir adições de *features* em um *sprint* mas manter cada *sprint* com o planejamento inicial é essencial para uma boa estimativa e planejamento do produto, além de evitar uma fonte de interrupção para a equipe de desenvolvimento.

Uma distinção importante a se fazer é que o *Scrum Master* não é um gerente de projeto, até mesmo porque gerentes de projetos não fazem parte do Scrum. Uma equipe de Scrum controla seu próprio destino e se auto-organiza em torno de seu trabalho, não havendo um líder. Depois de itens serem priorizados no *backlog*, as equipes ágeis tiram uma certa quantidade de trabalho fora do *backlog* e comprometem-se a completá-la nesse sprint. Nem os *scrum masters*, nem os *product owners* mandam o trabalho para a equipe.

O **time de desenvolvimento** é composto por profissionais que realizam o trabalho de entregar um incremento do produto com potencial de ser publicado no final de cada *sprint*. As equipes de Scrum mais eficazes têm geralmente de 5 a 7 membros e são eles que decidem sobre cada sprint. Eles prevêem quanto trabalho eles acreditam que podem completar durante a iteração usando o histórico das outras sprints como um guia.

Os times de desenvolvimento têm as seguintes características (SUTHERLAND; SCHWABER, 2016):

- Eles são auto-organizados. Ninguém (nem mesmo o *Scrum Master*) diz à equipe de desenvolvimento como transformar o *backlog* em incrementos de

funcionalidades potencialmente liberáveis. Além disso, não há líder geral da equipe que decide qual pessoa irá fazer qual tarefa ou como um problema será resolvido. São questões que são decididas pela equipe como um todo;

- As equipes de desenvolvimento são multifuncionais, com todas as habilidades necessárias para criar um incremento do produto;
- Scrum não reconhece títulos para membros do time que não sejam “desenvolvedor”, independentemente do trabalho realizado pela pessoa;
- Scrum não reconhece sub-equipes na equipe de desenvolvimento, independentemente de domínios específicos que precisam ser abordados como testes ou análise de negócios;
- Os membros da equipe de desenvolvimento individual podem ter habilidades especializadas e áreas de foco, mas a responsabilidade pertence à equipe de desenvolvimento como um todo.

B. Eventos do Scrum

Os eventos definidos no *Scrum* são usados para criar regularidade e minimizar a necessidade de reuniões não definidas. Todos os eventos têm uma duração máxima. Uma vez que um *Sprint* começa, sua duração é fixa e não pode ser encurtada ou alongada, porém os outros eventos podem terminar sempre que o objetivo do evento é alcançado.

Além do próprio Sprint, que é um container para todos os outros eventos, cada evento no Scrum é uma oportunidade formal de inspecionar e adaptar algo. Esses eventos são especificamente projetados para permitir a transparência crítica e a inspeção, dois dos pilares da teoria em que o Scrum se baseia. Os eventos são definidos abaixo:

O **Sprint** é a base do Scrum. É um período de tempo de geralmente 2 a 4 semanas de desenvolvimento, durante o qual um incremento feito, utilizável e potencialmente publicável é criado. Um novo *sprint* começa sempre imediatamente após a conclusão do anterior.

Cada sprint contêm e consiste em **Sprint Planning**, **Daily Scrums**, o trabalho de desenvolvimento, o **Sprint Review** e **Sprint Retrospective,** que dão estrutura ao mesmo.

No **Sprint Planning**, o trabalho a ser realizado no próximo Sprint é planejado de maneira colaborativa por todo o time do Scrum. Nessa reunião de planejamento de sprint estão presentes o *product owner*, *Scrum Master* e todo o time de desenvolvimento.

Durante a reunião de planejamento de sprint, o proprietário do produto descreve as características de maior prioridade para a equipe. A equipe faz perguntas suficientes para que eles possam transformar uma história de usuário de alto nível *product backlog* em tarefas mais detalhadas do *backlog* da sprint.

O **Daily Scrum** é um evento de 15 minutos para a equipe de desenvolvimento sincronizar as atividades e criar um plano para as próximas 24 horas. Isso é feito inspecionando o trabalho desde o último Daily Scrum e prevendo o trabalho que poderia ser feito antes do próximo.

O *Daily Scrum* é mantido na mesma hora todos os dias para reduzir a complexidade.

Na reunião, os membros da equipe de desenvolvimento explicam:

- O que eu fiz ontem que ajudou a equipe de desenvolvimento a atingir o objetivo da Sprint?
- O que eu farei hoje para ajudar a equipe de desenvolvimento a atingir o objetivo da Sprint?
- Vejo qualquer impedimento que atrapalhou a mim ou a equipe de desenvolvimento atender o objetivo da Sprint?

Um **Sprint Review** é realizado no final do Sprint para inspecionar o novo incremento e adaptar o produto caso seja necessário. Durante o Sprint Review, a equipe Scrum e as partes interessadas colaboram sobre o que foi feito no Sprint.
Com base nisso e em quaisquer alterações ao *product backlog* durante o Sprint, os participantes colaboram nas próximas coisas que poderiam ser feitas para otimizar o valor. É nessa reunião também que há a apresentação do incremento, destinada a obter feedback e fomentar a colaboração.

A **Sprint Retrospective** é uma oportunidade para a equipe se inspecionar e criar um plano para que as melhorias sejam incorporadas durante o próximo Sprint. Essa reunião ocorre após a revisão do Sprint e antes do próximo Sprint Planning.

C. Artefatos do Scrum

Os artefatos do Scrum representam trabalho ou valor para proporcionar transparência e oportunidades para inspeção e adaptação. Os artefatos definidos pelo Scrum são projetados especificamente para maximizar transparência das informações-chave para que todos tenham o mesmo entendimento do artefato (SUTHERLAND; SCHWABER, 2016).

O **Product Backlog** é uma lista ordenada de tudo o que pode ser necessário no produto e é a única fonte de requisitos para qualquer alteração a ser feita ao produto. O *product owner* é responsável pelo *product backlog*, incluindo seu conteúdo, disponibilidade e ordem.

A primeira versão do *product backlog* apenas estabelece requisitos iniciais conhecidos e de melhor compreensão de todo o time. Um *product backlog* nunca está completo e evolui à medida que há um maior aprendizado sobre o produto e seu público.

O *product backlog* lista todos os recursos, funções, requisitos, aprimoramentos e correções que constituem as alterações a serem feitas no produto em versões futuras. Os itens do *product backlog* têm uma descrição, ordem, estimativa e valor.

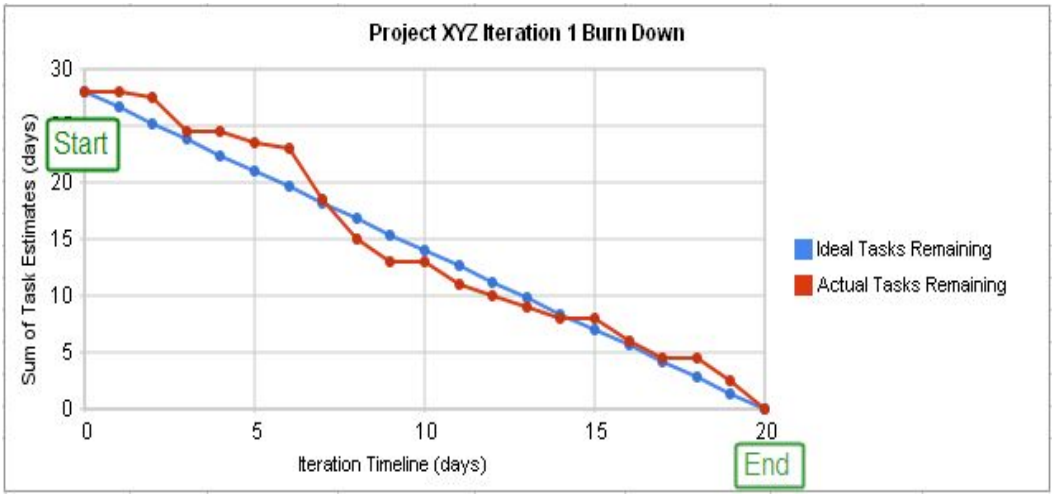
O **Sprint Backlog** é uma lista de tarefas identificadas pela equipe do Scrum para serem concluídas durante o próximo *Sprint*. Durante a reunião de planejamento de *Sprint*, a equipe seleciona alguns itens do *product backlog*, geralmente na forma de histórias de usuários, e identifica as tarefas necessárias para completar cada história de usuário. A maioria dos times também estimam quantas horas cada tarefa levará alguém na equipe a completar.

Um *Sprint backlog* geralmente é mantido como uma planilha, como no exemplo da figura abaixo:

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

Exemplo de um *Sprint Backlog* [17]

Durante o sprint, espera-se que os membros da equipe atualizem o backlog do sprint à medida que novas tarefas são requeridas ou completadas, o que pode ser muitas vezes feito durante o *Daily Scrum*. Uma vez por dia, o trabalho estimado restante no Sprint é calculado e colocado em um gráfico pelo Scrum Master, resultando em um *burndown chart.* Esse gráfico mostra a variação de tempo de trabalho estimado do que está sendo realmente feito, como pode ser visto na figura abaixo:



Burndown chart

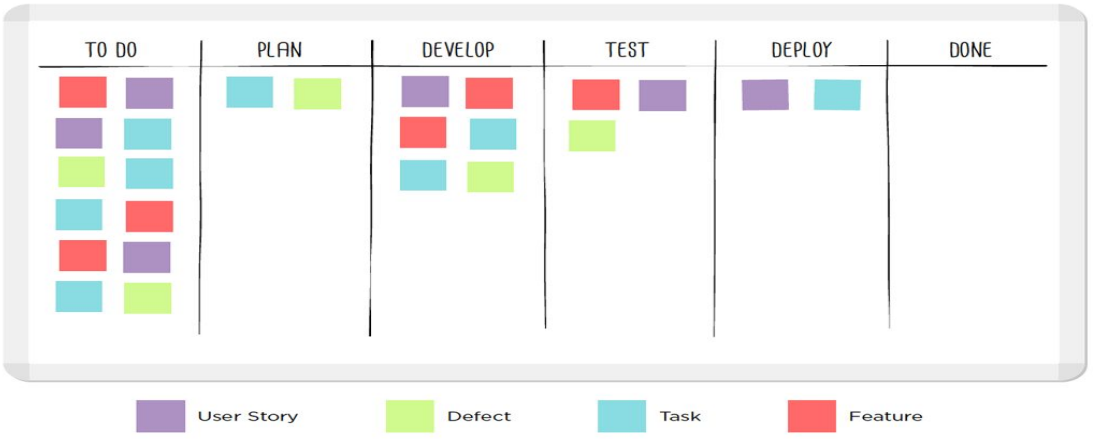
O **Incremento** é a soma de todos os itens do *product backlog* concluídos durante um Sprint com o que já foi implementado nos incrementos de todos os sprints anteriores. No final de um Sprint, o novo incremento deve ter sido “feito”, o que significa que deve estar em condições de ser utilizado e cumprir a definição que foi acordada pelo time para um incremento poder ser marcado como “feito”. Ele deve ser utilizável, independentemente de o *product owner* decidir realmente publicá-lo ou não.

2.3.3 Kanban

Kanban, na verdade, não é uma metodologia ágil e sim um método para gerenciar a criação de produtos com ênfase na entrega contínua sem sobrecarregar a equipe de desenvolvimento. Kanban é uma maneira para que equipes e organizações visualizem seu trabalho, identifiquem e eliminem gargalos e obtenham melhorias operacionais drásticas em termos de vazão e qualidade da produção.
Kanban é baseado em 3 princípios simples:

1. **Visualize o fluxo de trabalho**

Uma representação visual do processo permite ver exatamente como tarefas mudam de "não feito" para "bem feito". Quanto mais complexo um processo é, mais útil e importante torna-se a criação de um fluxo de trabalho visual, da mesma maneira que o kanban pode ser usado se existirem poucos passos. Independente da complexidade do projeto criar um quadro kanban permite a visualização do status do trabalho de maneira fácil.



Representação de um quadro kanban

A figura acima ilustra um quadro kanban. Basicamente, o quadro tem estados como colunas, onde cada item de trabalho passa da esquerda para a direita indicando o progresso do mesmo no fluxo de produção.

2. **Limitar o trabalho em progresso** (do inglês Work in Progress, WIP)

Atribuir limites explícitos a quantos itens podem estar em andamento em cada estado do fluxo de trabalho, isso ajuda a equilibrar a abordagem baseada em fluxo para que os times não se comprometam com trabalho demasiado ao mesmo tempo. Há um limite para o número de tarefas em que a equipe pode estar trabalhando bem. Se um projeto é simples ou complexo ou se a equipe é pequena ou grande, há uma quantidade ideal de trabalho que pode estar no processo ao mesmo tempo sem sacrificar a eficiência. Faça mais coisas fazendo menos, essa é a idéia por trás deste princípio.

3. Medir e melhorar o fluxo

O objetivo de qualquer projeto é garantir que o time seja otimizado com a maior eficácia possível. O aspecto visual e os limites do WIP ajudam a identificar os gargalos e o time a otimizar o fluxo de atividades do início ao fim.

Kanban fornece duas métricas chave para medir objetivamente a produtividade da equipe: **tempo de entrega**(*lead time*) e **tempo de ciclo**(*cycle time*). O tempo de espera é o tempo desde o momento em que a tarefa entra na fase de início até quando sai da fase de conclusão, isto é, da inserção no *backlog* à tarefa ter sido marcada como feita.

O tempo de entrega também deve ser gerenciado para garantir que a equipe esteja fornecendo recursos em tempo hábil. Uma equipe pode otimizar seu tempo de entrega por meio de qualquer uma das seguintes maneiras (MORRIS, 2012):

• Quebrar um passo em várias etapas menores;

• Juntar várias etapas menores em uma;

• Automação através de scripts;

• Adicionar recursos, por exemplo, computação de infra-estrutura;

• Ciclos adicionais de teste / revisão para limitar perda de atividades;

A outra métrica é o tempo de ciclo que mede quando o trabalho na tarefa realmente começou e quando foi concluído. Isso não está medindo o esforço, mas a duração que o ciclo permaneceu no sistema.

As métricas do Kanban [18]

A partir da figura, podemos entender que o tempo de ciclo(*cycle time*) é o tempo necessário para que um recurso se mova do início ao fim do processo após

ter sido iniciado, enquanto o tempo de entrega(*lead time*) é o tempo entre o momento em que o recurso entra e sai de todo a cadeia do processo. O cálculo do tempo médio de entrega e do tempo de ciclo fornece informações sobre o fluxo de trabalho. O tempo médio de entrega pode fornecer uma prova empírica de que o processo está melhorando, pois dá uma sensação de velocidade de entrega. Embora o tempo de ciclo possa fornecer uma medida objetiva de quanto tempo demora a entregar uma *feature* que pode ser usada na estimativa de *features* futuras e planejamento.

Como abordado durante toda essa seção, as metodologias ágeis oferecem uma estrutura para ajudar as equipes a desenvolverem produtos com incrementos de funcionalidades entregues constantemente, focando em uma entrega rápida de valor comercial, por isso ela foi escolhida ao invés da gestão de desenvolvimento mais tradicional.

Das abordagens ágeis, a mais simples a ser adotado e que pareceu a mais adequada para o time de desenvolvimento da Eleve foi o Scrum. Uma prática chave no Extreme Programming, a programação em pares, não seria viável pelo demanda de tempo que seria requerido para dois programadores desenvolverem o código juntos já que a equipe só é composta por dois desenvolvedores com divisões de tarefas específicas para se otimizar a produção.

Além disso, a estrutura do Scrum dividida em Sprints e seus eventos possibilita que a equipe entenda mais sobre a sua capacidade produtiva e que aprenda à medida que os ciclos vão acontecendo. Com isso, o Scrum está sendo utilizado e também há o auxílio de alguns elementos do Kanban como o uso do seu quadro para poder haver transparência no status das tarefas de cada Sprint.

2.4 Principais arquiteturas para desenvolvimento Android

Partindo do princípio que a aplicação que possibilitaria a intervenção digital do programa da Eleve seria móvel, a próxima decisão a ser tomada deveria ser em qual plataforma a primeira versão do aplicativo seria desenvolvida.

Por já ter trabalhado com o desenvolvimento Android e pela plataforma ter uma enorme utilização no mercado brasileiro, mais de 95% dos smartphones vendidos aqui são Android (HIGA, 2016), optamos por construir nossa primeira versão em Android. Com essa escolha, e já com a metodologia de desenvolvimento em mente, precisavamos então entender como estruturar a implementação do aplicativo em si da melhor maneira.

Com o avanço dos componentes de hardware dos smartphones fez-se com que o poder de processamento dos mesmos aumentasse drasticamente possibilitando, assim, a criação de aplicativos cada vez mais complexos. Com esse aumento de complexidade houve também um crescimento da necessidade de se desenvolver aplicativos mais sustentáveis.

Dessa maneira, no desenvolvimento de uma aplicação complexa, desafios que geralmente são encontrados provavelmente também foram abordados antes e já possuem algumas ótimas soluções para manter a sustentabilidade do aplicativo a longo prazo. Essas soluções são muitas vezes referidas como padrões, e os padrões mais comumente abordadas são os de design e de arquitetura. Eles simplificam o desenvolvimento e devem ser usados sempre que forem apropriados.

Os principais padrões de arquitetura que serão abordados nesta seção são: o Model-View-Presenter (MVP), Model-View View-Model (MVVM) e a arquitetura padrão do Android, o Model-View-Controller (MVC).

2.4.1 Model-View-Controller (MVC)

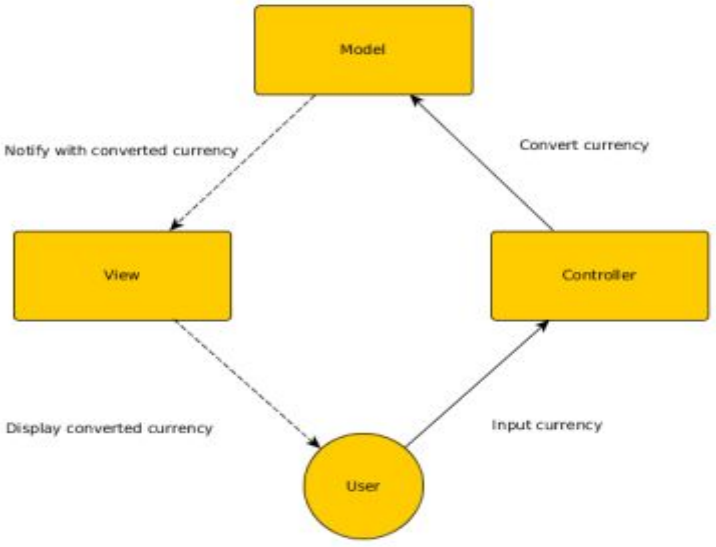
O MVC é um padrão sólido e estabelecido que visa isolar as diferentes funções de uma aplicação, processo conhecido como separação de preocupações. O MVC é dividido em três áreas com diferentes responsabilidades:

1. **Model** - O *Model* lida com a lógica comercial e muitas vezes consiste em objetos que descrevem o sistema. É o componente que armazena dados/estado em classes de objetos complexos ou simples dados primitivos. Pode não armazenar os dados em si, mas, ao invés disso, ele recupera os dados remotamente.

2. **View** - É a interface do usuário e não deve conter nenhuma lógica de negócio. Em vez disso, a função é ouvir eventos como o gesto ou cliques e conversar com o controlador para delegar essas tarefas.
3. **Controller** - Media o acesso da View ao Model. Seu trabalho é receber solicitações da View, analisar o pedido e chamar o Model correto para lidar com esse pedido. Assim, ele muda o estado no Model e às vezes, passa as mudanças de volta para a View.

A seguir, um exemplo de aplicativo para converter moedas é usado para explicar como esses componentes interagem uns com os outros:

- Um usuário pode inserir uma moeda e ler a conversão da mesma em uma UI.
- O *Controller* então passa a entrada do usuário para o *Model*, que cuida da lógica de negócios (neste caso, convertendo a moeda).
- O *Model*, em seguida, alerta a View de alguma forma para que ela possa se atualizar e exibir a moeda convertida.



Comunicação entre os componentes do MVC

Implementação do MVC em Android

A arquitetura MVC é a arquitetura padrão para aplicativos nativos do Android. Um exemplo de diagrama UML que pode representá-la é descrito na figura abaixo. Mesmo o layout.xml, que é o arquivo onde o layout da aplicação é definida, não sendo uma classe Java, portanto, o mesmo não deveria ser mostrado no diagrama UML, pela sua importância na arquitetura ele foi incluído.

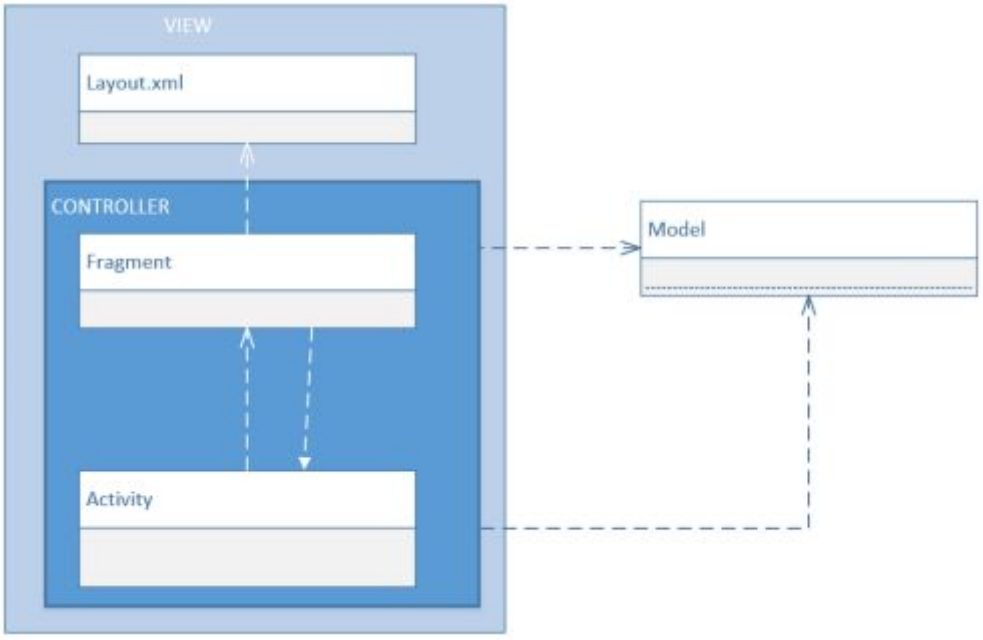


Diagrama UML que representa o MVC [19]

O *Model* normalmente é uma classe de algum o objeto Java e é o responsável pela lógica de negócios.

A *View* é a combinação do arquivo de recursos de layout e o Activity/Fragment. Não é só *XML* do layout porque o recurso de layout não possui controle total sobre a *UI*. Por exemplo, todos os widgets em arquivos de layout devem ser inflado para Activities/Fragments antes de serem exibidos na tela. Além

disso, o arquivo de layout não pode controlar a visibilidade do widget dinamicamente (LOU, 2016).

O *Controller* é a *Activity* ou o *Fragment*. Ele captura uma série de eventos da *View* e envie resposta de volta ou envia pedidos para o *Model* para obter e atualizar dados.

Assim, no Android, o componente *View* é o arquivo xml e Activity/Fragment. Os *Controllers* são as Activities/Fragments. O que mostra que a fronteira entre esses componentes não é clara e eles se sobrepõem.

2.4.2 Model-View-Presenter (MVP)

O MVP foi inicialmente apresentado por Mike Potel que o primeiro artigo sobre essa arquitetura em 1996 (POTEL, 1996). O MVP é derivado do MVC, mas deixa mais claro a separação entre o *Model* e *View-Controller*, assim podendo ser representado como dois conceitos fundamentais: gerenciamento de dados e UI.

Embora no artigo original do MVP, seis componentes são definidos, atualmente a maioria do tempo a arquitetura MVP é descrita como 3 componentes: *Model*, *View* e *Presenter*.

As funções deles são basicamente iguais com as do MVC:

O **Model** contém a lógica da aplicação. Ele controla como os dados podem ser criados, armazenados e modificados.

A **View** é uma interface passiva que exibe dados e roteia as ações do usuário para o Presenter.

O **Presenter** atua como intermediador. Ele recupera os dados do *Model* e o mostra na *View*. Ele também processa as ações do usuário encaminhadas para ele pela *View*.

O *Presenter* e o *Controller* têm um papel semelhante. Eles são responsáveis pela comunicação entre *Model* e *View*. Porém, o *Controller* não gerencia *Model* e *View* tão estritamente quanto o *Presenter*.

No padrão MVC, a camada *View* é de alguma maneira inteligente e pode recuperar dados diretamente do *Model*. No padrão MVP, a *View* é completamente passiva e os dados são sempre entregues à *View* pelo *Presenter*. Com MVP, o acoplamento entre a *View* e o *Model* é eliminado. O *Model* sabe nada sobre o *Presenter* e vice-versa.

Assim a grande diferença em relação ao MVC é como a *View* é tratada. Em vez de ter um *Controller* que manipula a entrada do usuário, o fluxo MVP começa a partir da *View*. Ela captura eventos de entrada do usuário e os encaminha para o *Presenter*. O *Presenter* pode então interagir com o *Model*. Para um evento simples, o *Presenter* faz um decisão e atualiza a *View*. Se for uma ação complexa, o Presenter enviará uma mensagem para o *Model* para recuperar dados relevantes e depois atualizar a *View*.

A atualização da View pode ocorrer de duas maneiras. Quando o Model é atualizado, ele alerta a View afetada que é atualizada para refletir o estado atual do sistema através do padrão de design *Observer,* caracterizando o MVP com *Supervising Controller* (FOWLER, 2006). Ou o MVP pode implementar *Passive View* (FOWLER, 2006). Isso torna a *View* completamente passiva, o que significa que não precisa atualizar-se a partir do *Model*, sendo assim, trabalho dos *Presenters* lidar com a entrada do usuário (vinda da *View*) e atualizar a *View* para refletir o estado atual do sistema.

A figura abaixo ilustra como seria o exemplo do aplicativo de conversão de moedas usando o MVP Passivo:

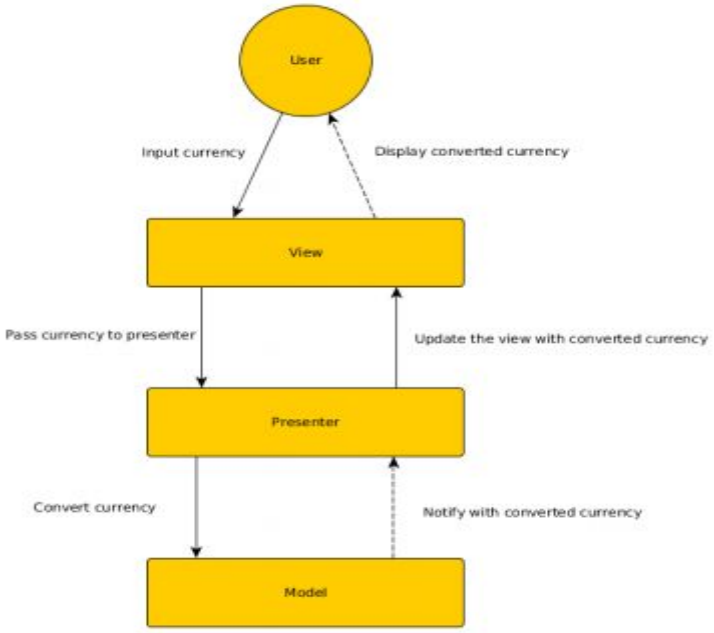
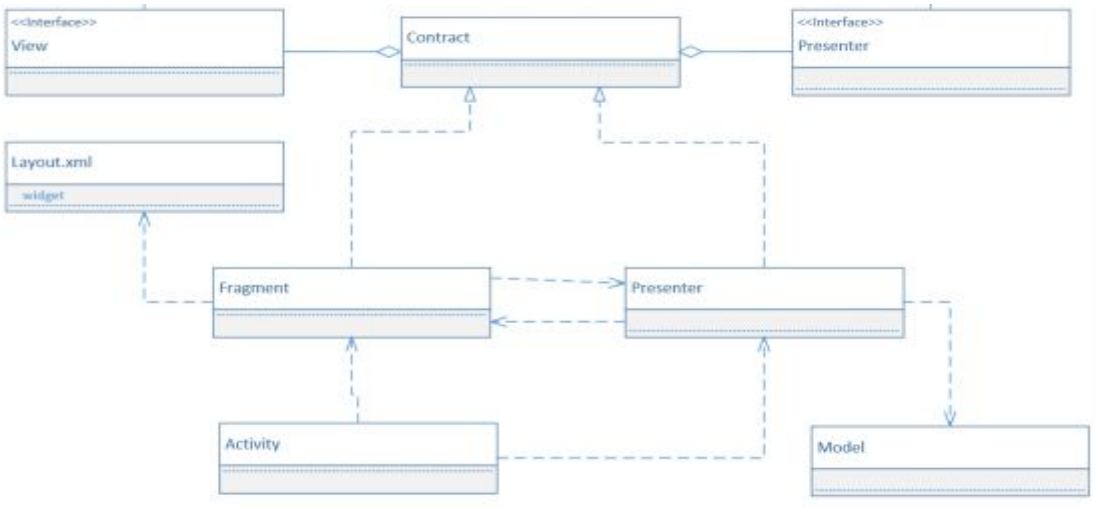


Figura - Comunicação entre os componentes do MVP

Implementação do MVP em Android

Não há regras específicas e formais para a implementação do MVP Android, no entanto, felizmente, o Google publicou uma amostra open source do MVP no GitHub [23], servindo de base para implementações. Dessa maneira, um exemplo de diagrama UML que pode representar a estruturação do código é mostrada na figura abaixo:



Representação do MVP em UML [19]

A Classe de Contrato é responsável por gerenciar a interface entre uma *View* específica e seu *Presenter,* que é a composição da interface da *View* e da interface do *Presenter*. A *View* só pode chamar os métodos do *Presenter* mostrados na interface do *Presenter* e vice-versa.

Os *Fragments/Activities* e layout xml são a *View*. Na maior parte do tempo, o *Fragment* é que lida com as responsabilidades da *View* já que a *Activity* destina-se a criar a instância do *Fragment*(*View*) e *Presenter*, e depois conectar os dois componentes.

Como explicado em (LOU, 2016), o fluxo de interação de todas as classes ocorre da seguinte maneira:

1. A classe da Activity é chamada com conectando a View e o Presenter.
    a. Uma instância do Fragment é criada no método OnCreate () da Activity.
    b. A instância do Presenter é criada chamando o método do construtor do mesmo. Neste momento, View e Presenter estão vinculados um ao outro neste construtor. 1) A instância da View é passada como o parâmetro, então a View é vinculada para este Presenter. 2) O método setPresenter() da View é chamado passando a instância do Presenter como parâmetro, vinculando, assim, o Presenter com a View.
2. A View é visível para os usuários, o que indica que o *Fragment* está no estado *resume*.

2.4.3 Model-View View-Model (MVVM)

A arquitetura Model-View-ViewModel (MVVM), também baseada no MVC, foi apresentada pela primeira vez pelo arquiteto de software John Grossman da equipe do Microsoft Silverlight em uma postagem no seu blog em 2005 (GOSSMAN, 2005).

A arquitetura MVVM possui três componentes: Model, View e ViewModel. Assim como nas outras duas arquiteturas, no MVVM a **View** mostra UI do aplicativo

e o **Model** representa os dados e a lógica de negócios. O **View-Model** destina-se a gerenciar o estado da *View*, passando os dados e operações para a View e também gerenciando a lógica e o comportamento dela (GOSSMAN, 2005).

A conexão entre o *View-Model* e a *View* é mais complexa do que no MVP, pois existem dois tipos de conexões: conexões tradicionais e as de *databinding*. As tradicionais são semelhantes no MVC e MVP, onde o View-Model muda a View em uma chamada de método no código.

Já *databinding* é um novo mecanismo introduzido pelo MVVM. Ele significa que os dados de fontes de dados estão vinculados aos seus consumidores. Ou seja, os dados do armazenamento local estão vinculados a layouts, assim, os dados do *Model* estão ligados diretamente à *View,* em casos simples. Além disso, uma característica importante do *databinding* é que as mudanças de dados são sincronizadas automaticamente entre fontes e consumidores (MVVM... 2016).

Assim, com o *databinding* do MVVM, o View-Model não precisa notificar a View sobre as mudanças através do código e a View sabe que os dados são carregados e ela mesmo os mostra.

Com isso, enquanto que no MVP e MVC após os dados serem carregados, o Presenter e o Controller é que irão atualizar a View no código, com o *databinding*, a View está ligada aos dados, então quando os dados são carregados a View muda automaticamente.

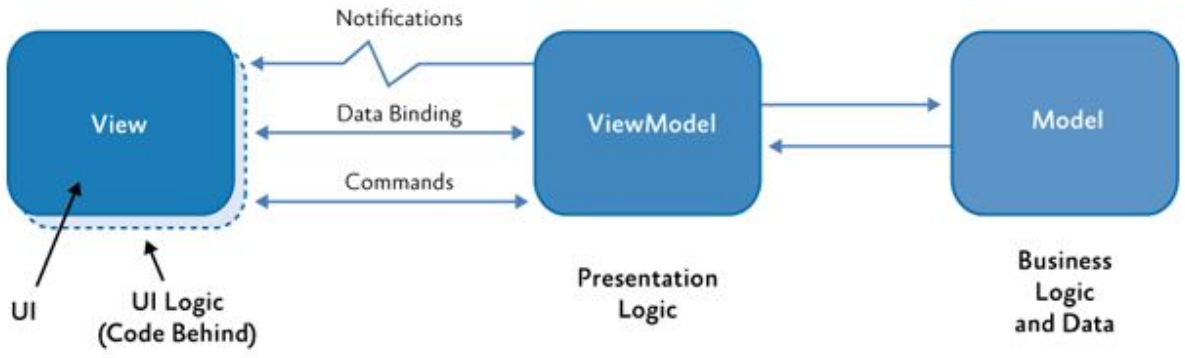


Figura 9 - Comunicação entre os componentes do MVVM [26]

Na programação em Android, assim como em outras plataformas, há sempre uma evolução. Novas versões, novas práticas e ferramentas emergem

constantemente. Os desenvolvedores sempre visam encontrar a prática mais correta, porém, várias delas podem se adequar ao projeto em que eles estão envolvidos e mudanças de arquitetura podem ocorrer ao longo do processo.

Diante disso, depois de já ter utilizado a arquitetura MVC em projetos anteriores e ter constatado as possíveis complicações de não se ter a fronteira entre a View e Controller devidamente claras durante a implementação, optamos por usar o MVP. Essa escolha foi baseada no fácil entendimento da arquitetura e também por não ter tido tanto contato com o MVVM. Até agora o desenvolvimento baseado no MVP está nos trazendo vários benefícios, principalmente pela equipe de desenvolvimento ser pequena e esta arquitetura nos prover separações claras que podem ser atribuídas entre nós.

3.Competidores: Ferramentas de Monitoramento

Com a tecnologia mais presente no dia a dia das pessoas e com um número crescente de aplicativos nas lojas virtuais disponíveis para os diversos perfis de usuários, existe uma quantidade enorme de ferramentas que foram desenvolvidas para atender necessidades nas mais diversas áreas. Como citado anteriormente neste trabalho, o setor de mHealth está com cada vez mais aplicativos sendo publicados, e entre eles há várias alternativas para ferramentas relacionadas ao registro de peso, refeição e atividades físicas que auxiliam o usuário a manter uma vida mais saudável e perder peso.

Com isso, nesta seção iremos avaliar algumas das ferramentas que estão disponíveis no *Google Play* para que possamos descobrir as funcionalidades mais frequentes, além de elementos da experiência do usuário como o *design* do aplicativo.

3.1 Lifesum

Lifesum é um aplicativo que ajuda o usuário a monitorar toda a sua alimentação e as atividades físicas praticadas. Além disso, ele possibilita o registro da quantidade de água consumida durante o dia e o usuário pode criar um perfil com informações sobre medidas corporais e peso, para assim acompanhar a sua evolução.

Para acompanhar a sua alimentação, o aplicativo fornece ao usuário um banco de dados de alimentos com várias informações nutricionais como calorias, quantidade de carboidratos, proteínas, gorduras, etc. Dessa maneira, ao registrar uma nova refeição o usuário sabe bem o quanto está ingerindo e na tela principal do aplicativo há a comparação do que já foi consumido com o que ainda pode-se ingerir.

Para o monitoramento de atividade física, o usuário pode inserir exercícios com o auxílio de um banco de atividades que já calcula as calorias gastas a partir da quantidade de minutos. Esse valor calculado depois já é atualizado na tela principal onde é mostrado a quantidade de calorias restantes que podem ser consumidas no dia.

Nas figuras seguintes mostraremos como essas funcionalidades estão dispostas no aplicativo:

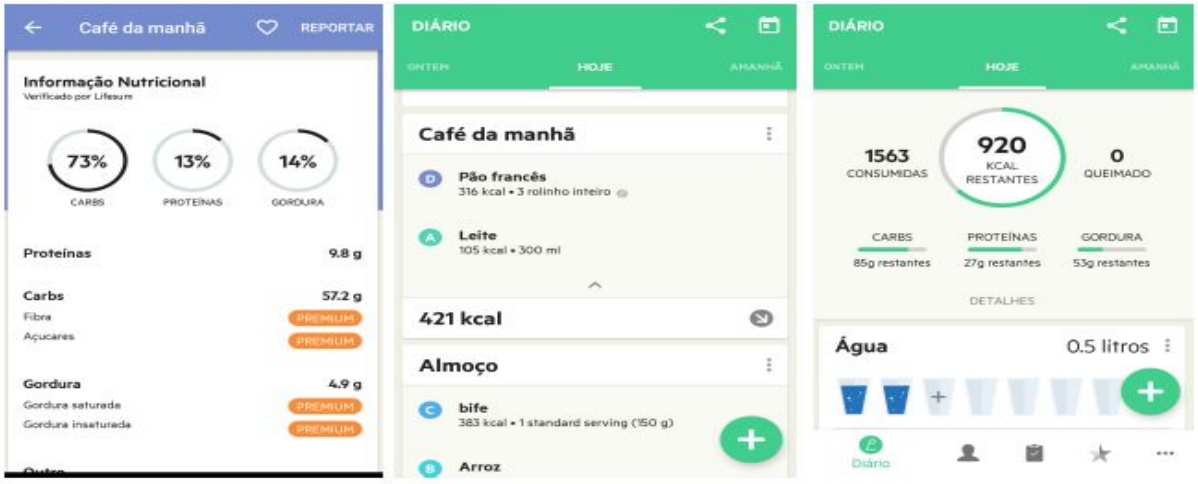


Figura - Telas de inserção de refeição e tela principal

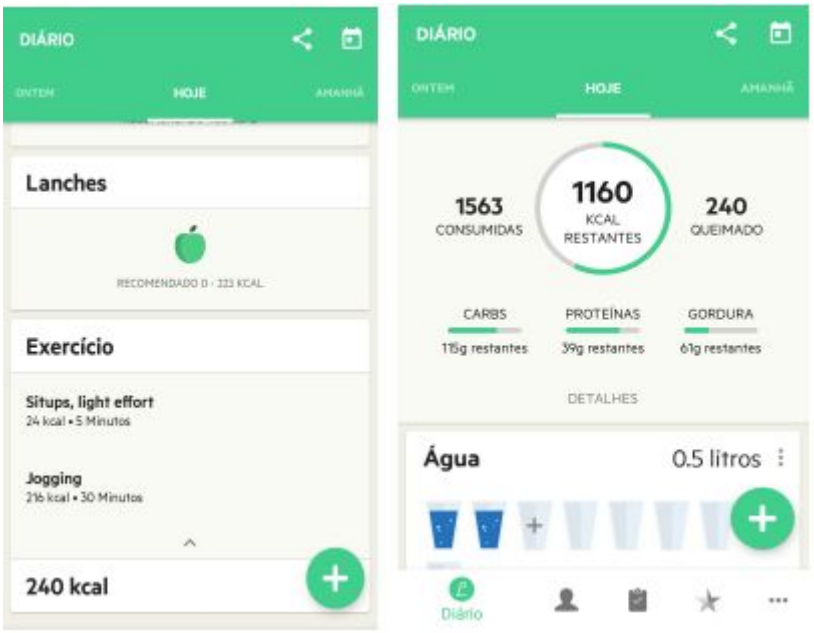


Figura - Visualização de atividades inserida e resultado na tela principal

3.2 Noom Coach

Noom Coach é um aplicativo para o programa de saúde e perda de peso do Noom[1]. Por ser o aplicativo que é utilizado pelos participantes do programa de 16 semanas do Noom, ele oferece mais funcionalidades de acompanhamento do que outras ferramentas.

Ao se inscrever no programa e tendo acesso à versão paga do aplicativo, o usuário é inserido em um grupo com outros participantes e recebe o apoio de um coach durante todo o processo. No aplicativo, o usuário pode conversar com o coach e outros participantes através do chat, assim, sendo acompanhado pelo coach e interagindo com outros com o mesmo objetivo.

Além do chat, na versão paga, o participante do programa recebe conteúdos educacionais toda semana e conta com uma base de receitas que podem o ajudar a se alimentar melhor.

Na versão gratuita, assim como outras ferramentas, ao registrar refeições o usuário pode checar valores nutricionais e calóricos sobre os alimentos, porém, o Noom Coach oferece algumas funcionalidades informativas a mais. Os alimentos são separados em três cores, verde(nutritivos), amarelo(comer com moderação) e vermelho(limitar as porções), para facilitar o entendimento do usuário, há *cards* com mais informações gerais sobre os grupos alimentares ingeridos e a quantidade de calorias consumida é passada de maneira bem simples e amigável, como pode ser visto abaixo.

[1] https://www.noom.com/

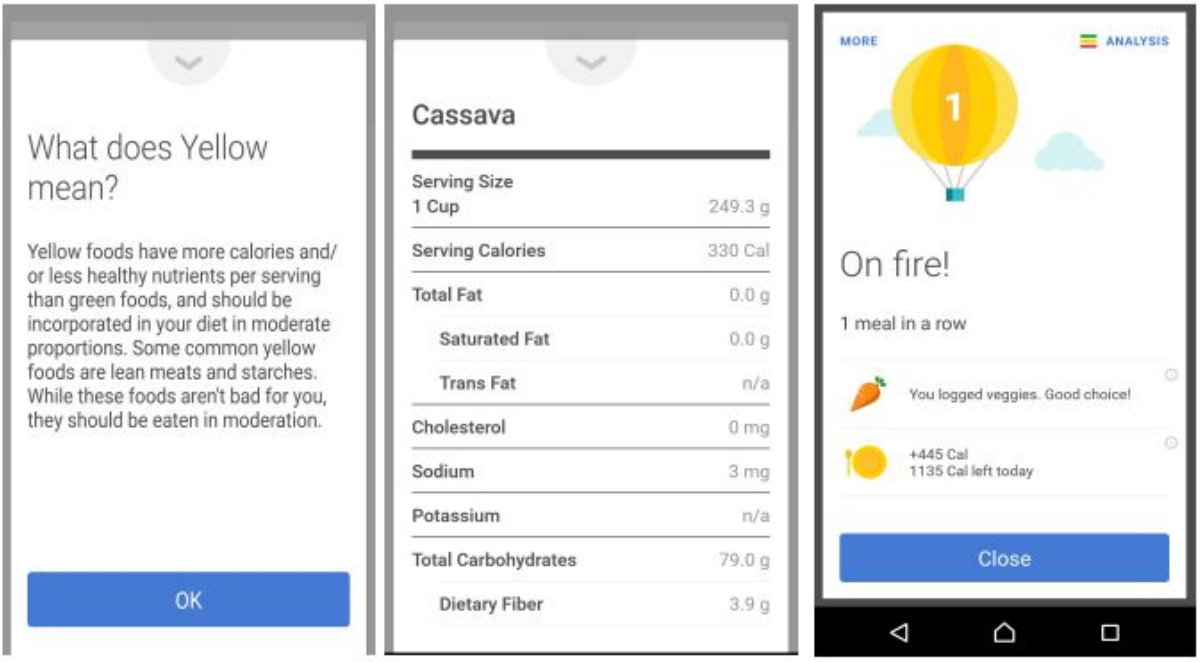


Figura - Telas de informações ao inserir refeição do Noom

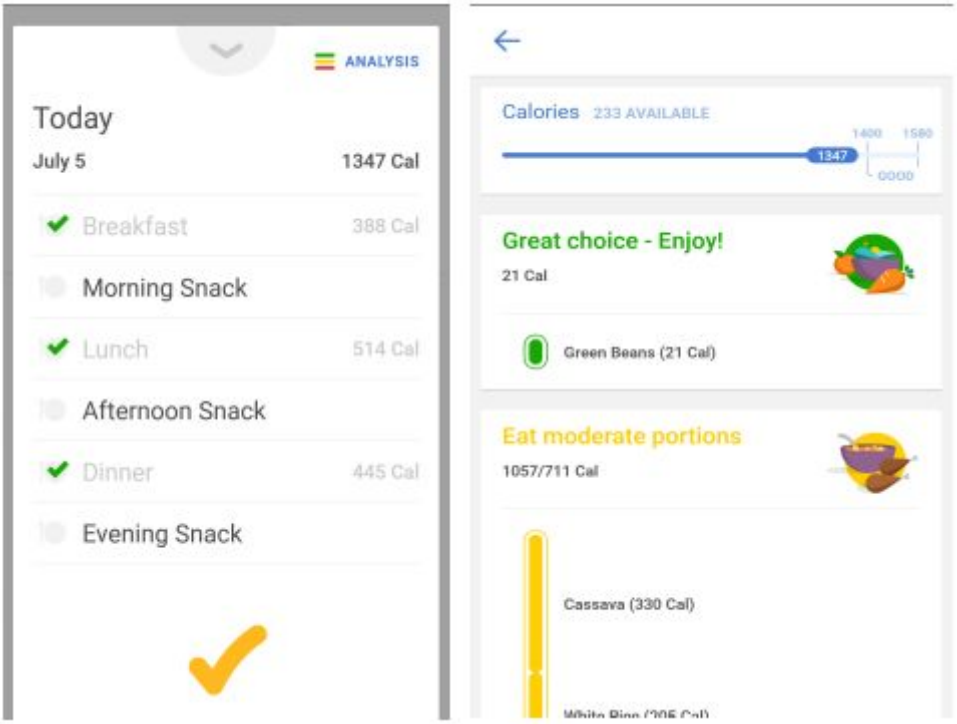


Figura - Telas com calorias por refeição e por alimento

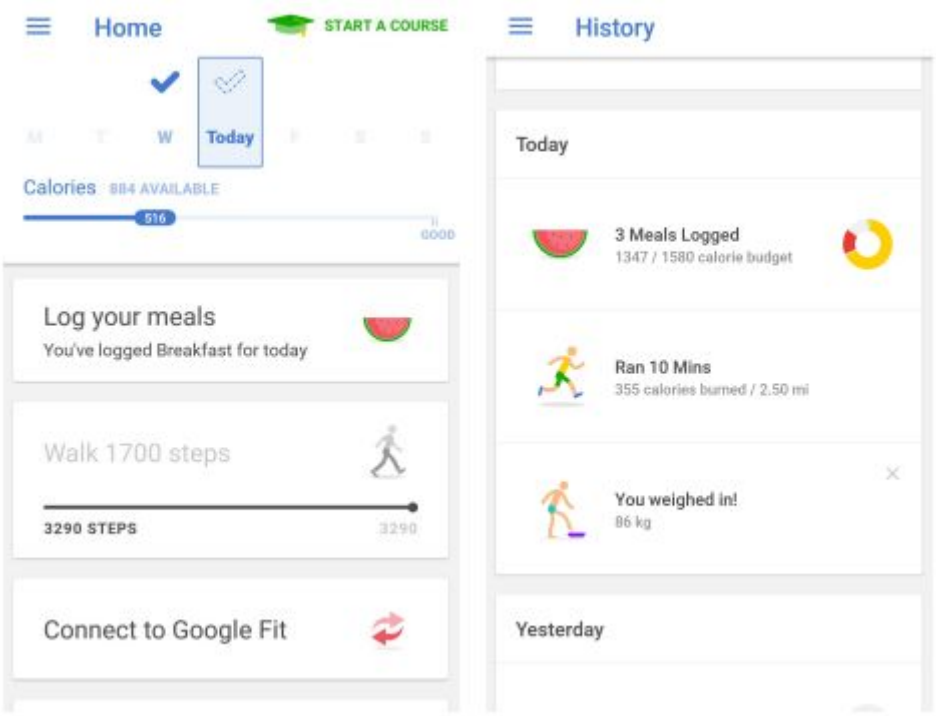


Figura - Tela principal e tela de histórico dos registros

3.3 Dieta e Saúde

Dieta e Saúde é um aplicativo e programa de emagrecimento digital desenvolvido pelo MinhaVida[2], um dos maiores sites de saúde e bem-estar do país. Uma das principais diferenças dele em relação à outras ferramentas de contagem de calorias, é que ele se baseia em um dos pilares do Dieta e Saúde: a Dieta dos Pontos.

Essa dieta contabiliza os pontos de cada alimento considerando quantidade, calorias e qualidade nutricional. Ao registrar-se no aplicativo, o usuário coloca o peso atual e o peso meta e, então, recebe uma avaliação que mostra quantos pontos ele deve consumir por dia durante o processo de emagrecimento.

Com isso, além das informações nutricionais fornecidas na hora de inserir o alimento, o usuário também sabe a quantidade de pontos equivalentes e pode ver o impacto destes valores no seu consumo de pontos.

[2] http://www.minhavida.com.br/

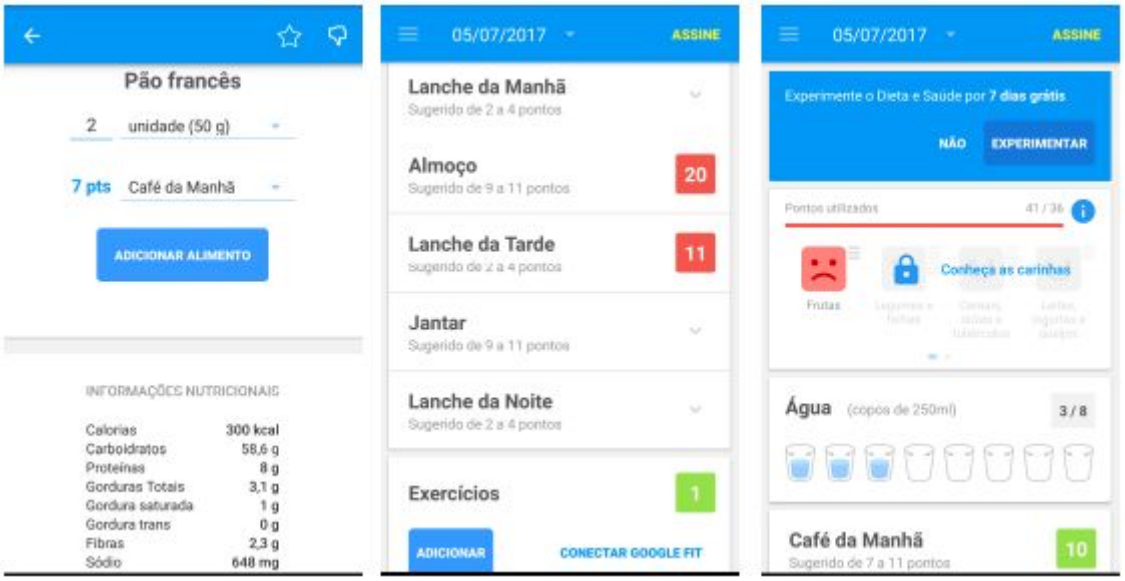


Figura - Telas de inserção de alimento e tela principal do diário

Como pode ser visto na figura acima, na tela principal do diário alimentar o usuário é informado com a quantidade de pontos das refeições registradas e da quantidade sugerida para cada uma delas. Além disso, na parte superior da tela principal do diário o usuário conta com o valor total de pontos relativos ao sugerido para a sua dieta e, na versão paga, com o valor de pontos consumidos em relação aos grupos alimentares, através de carrinhas.

O aplicativo Dieta e Saúde, assim como os outros analisados, também conta com uma versão paga onde o usuário tem acesso a mais funcionalidades e recebe mais apoio durante o processo de reeducação alimentar. Ainda na versão gratuita, há uma área onde os usuários podem compartilhar suas experiências e interagirem entre si através de um *feed* de postagens, o Comunidade. Já na versão paga há o desbloqueio de outras áreas da ferramenta: Cardápio, Reuniões e Fitness.

Na área de Cardápio, são mostradas sugestões de pratos para planejar as refeições além de ter *feedback* sobre os alimentos inseridos no aplicativo. Em Reuniões, o usuário pode entrar em contato com a equipe de nutricionistas e psicólogos do Dieta e Saúde em caso de necessidade de auxílio e em Fitness há uma série de circuito de atividades físicas disponível para serem realizados com vídeos de demonstração.

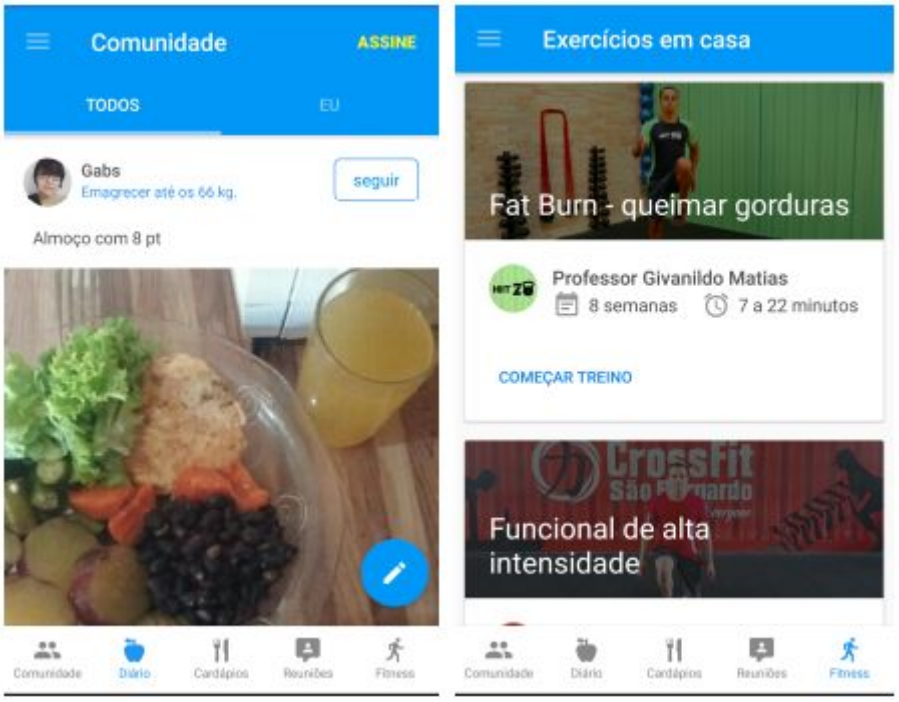


Figura - *Feed* da Comunidade e área de *Fitness* do *app*

Mesmo com essa variedade de opções para se fazer o monitoramento demonstrada através destes três apps e de tantos outros que estão disponível para download no Google Play, poucos conseguem entregar os valores de monitoramento, apoio no acompanhamento e conscientização sobre a nova rotina saudável que as pessoas que buscam emagrecimento precisam.

Além disso, pelo programa da Eleve se tratar de um programa pago não poderíamos utilizar aplicações móveis fornecidas por outros serviços. Com isso, surgiu a necessidade do desenvolvimento do aplicativo deste trabalho. Pela limitação do tempo, o escopo do aplicativo referido neste trabalho ainda está somente no monitoramento, no entanto, já agora fornecemos uma ferramenta intuitiva em comparação com outras no mercado.

À medida que incrementamos mais a aplicação deste trabalho poderemos fornecer os valores acima referidos para que os participantes do programa possam ter a melhor experiência para o emagrecimento em uma só ferramenta.

4. Desenvolvimento da aplicação

Para estruturar o desenvolvimento do aplicativo Android de monitoramento do programa Eleve fez-se o uso de uma das abordagens da metodologia Ágil, o Scrum. Como visto anteriormente neste documento, o Scrum por se tratar de um *framework* ágil para desenvolvimento incremental de software que se baseia na transparência, inspeção e adaptação ele traz bastante benefícios para a equipe que o utiliza. Através dos artefatos e eventos que o compõem, exemplos como *Sprints*, *Product Backlog*, *Sprint Planning* etc…, e pelo desenvolvimento ser incremental, a equipe é capaz de melhor planejar e estimar a implementação da ferramenta fazendo com que o desenvolvimento seja feito de uma maneira mais eficiente.

Definido que o Scrum seria utilizado e com os requisitos iniciais levantados a partir de *user stories* pôde-se levantar o *Product Backlog* inicial e já fazer o Sprint Planning para o primeiro Sprint de desenvolvimento. A partir de então, houve a repetição do ciclo de iteração do Scrum(planejamento, implementação e revisão de *Sprints*) para o desenvolvimento do aplicativo de maneira incremental.

Nesta seção, mostraremos como o monitoramento era feito antes do aplicativo e a lista de requisitos inicias do aplicativo. Depois abordaremos a arquitetura de todo o sistema(do aplicativo Android à comunicação com o servidor), ferramentas auxiliares utilizadas para a viabilidade do sistema e detalharemos as *Sprints* que foram necessárias para a construção do aplicativo.

4.1 Versão pré-aplicativo

Antes do início do desenvolvimento do aplicativo para monitoramento, fez-se necessário a implementação de alguma solução digital para que as participantes da primeira turma pudessem registrar a sua alimentação, atividades físicas e peso e, assim, a equipe da Eleve pudesse obter esses dados para fazer análises semanais. Para isso, como o contato da coach, que é a mentora especializada em formação de

hábitos que acompanha cada participante, com as participantes já seria feito através da utilização do Telegram[3], decidimos implementar um bot para fazer esse registro.

Para que as participantes registrassem dados no seu diário, era preciso adicionar o bot na lista de conversas e ter o primeiro contato. Das próximas vezes, as participantes só precisam interagir com o bot seguindo alguns fluxos de conversa em que ele foi programado.

Para iniciar a interação com o bot, a participante insere “Oi” e ele responde disponibilizando as três opções de registro. A partir de então, o fluxo de cada uma das opções variam um pouco mas sempre ocorre através de trocas de mensagens entre o bot e a participante. No fim do registro a participante é avisada de que a operação foi realizada com sucesso.

Figura - Interações com o bot no Telegram

Na figura acima, vemos telas com demonstrações da interação com o bot para fazer o registro de refeição, atividade física e peso. Assim, pelo Telegram, as participantes podem conversar com a coach, com outras participantes e interagem com o bot. Funcionalidades que serão todas passadas para o aplicativo mas que, exceto o monitoramento, foram deixadas fora do escopo deste trabalho.

[3] https://telegram.org/

4.2 Lista inicial de requisitos

Para começarmos a desenvolver a aplicação, precisamos levantar quais são os requisitos iniciais que deverão ser implementados para o produto, chamado de *Product Backlog* no Scrum. Sendo a primeira versão do *product backlog*, ela não é a mais completa e nem a mais correta já que o backlog muda à medida que há um aprendizado sobre o produto e o público.

No *product backlog* inicial levantado, listamos funcionalidades com suas respectivas descrições para depois priorizarmos e organizarmos a primeira *Sprint*. Na figura abaixo podemos ver os requisitos iniciais para o aplicativo:

Sign Up	Usuário acessa a ferramenta pela primeira vez; Usuário fornece e-mail, nome e cria uma senha.
Login	Usuário entra na ferramenta automaticamente. Caso ele tenha feito logout, deve fornecer e-mail e senha.
Logout	Usuário clica para fazer Logout. Ferramenta volta para tela de Sign Up / Login
Acessar dias anteriores	Usuário desliza a tela para acessar dias anteriores OU Usuário escolhe um dia específico no calendário.
Inserir Peso	Usuário clica para inserir novo log de peso. Usuário fornece peso com uma casa decimal de precisão. Ferramenta salva peso no servidor. Ferramenta volta para tela anterior.
Visualizar Status Atual de Peso	É exibido quanto de peso ele perdeu e o seu peso atual.
Visualizar Progresso de Perda de Peso	Usuário clica para visualizar seu progresso e tem acesso a um gráfico com sua perda de peso (desde o início do programa) até o dia atual. São também exibidos o seu peso atual e perda de peso total, junto ao gráfico.
Inserir Refeição	Usuário clica para inserir refeição. Ferramenta redireciona para uma nova tela. Usuário pode tirar foto do que comeu. Usuário deve descrever o que comeu, apontar qual a refeição do dia, sua satisfação em relação ao alimento e, o horário que comeu. Ao concluir inserção, ferramenta volta para a tela anterior.
Visualizar Refeições do Dia	É exibido o que o usuário comeu em cada refeição do dia específico, assim como a hora e a sua satisfação relacionada.
Visualizar Refeição específica	Usuário clica para visualizar uma refeição específica e tem acesso à foto da refeição, junto aos demais dados.
Visualizar Progresso da Alimentação	Usuário clica para visualizar o progresso de sua alimentação e tem acesso a um gráfico que aponta a satisfação que o usuário tem tido em relação à sua alimentação.
Inserir Atividade Física	Usuário clica para inserir a atividade física. Ferramenta direciona usuário para uma nova tela. Usuário insere o tempo de exercício, intensidade, tipo (não obrigatoriamente) e satisfação em relação ao exercício.
Visualizar atividade do dia.	Usuário tem acesso a quantos minutos e em qual intensidade de exercício ele praticou naquele dia.
Visualizar atividade física específica	Usuário tem acesso a todas as informações referentes àquela atividade.
Visualizar progresso a atividade física.	Usuário tem acesso a um gráfico com quantos minutos de atividade física ele vem praticando semanalmente, assim como intensidade média de cada semana. É também exibido quantos minutos já foram praticados na semana em que se encontra.

Figura - Product Backlog inicial para o aplicativo

Como dito anteriormente, o product backlog inicial sofre mudanças ao longo do tempo com adição e remoção de funcionalidades de acordo com o maior entendimento do produto e da necessidade de seus usuários. Desse modo, uma das funcionalidades listada inicialmente, “Visualizar Progresso da Alimentação”, não foi incorporada nas Sprints desse primeiro módulo de monitoramento do aplicativo da Eleve.

4.3 Arquitetura do sistema

A aplicação Android desenvolvida teve sua arquitetura modificada para adaptações que ocorreram ao longo do processo de desenvolvimento e para a versão final documentada neste trabalho temos a estrutura mostrada na figura abaixo:

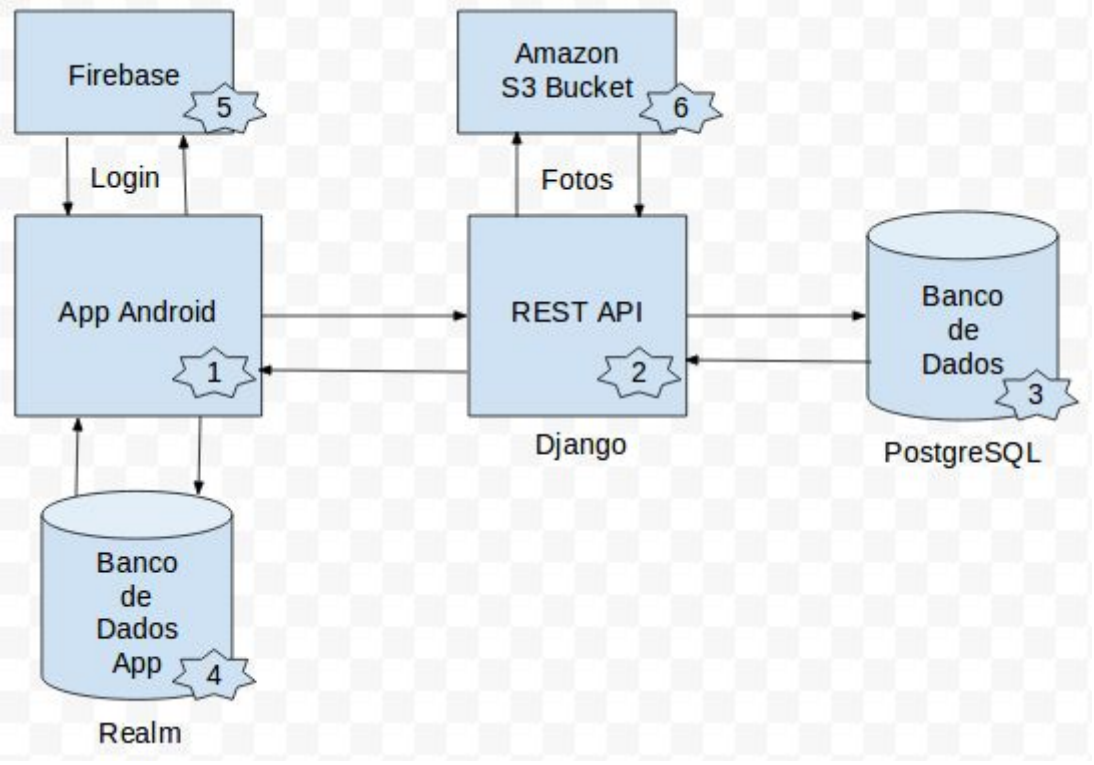


Figura - Arquitetura do sistema

A arquitetura do sistema é formada por :

1. **Aplicativo Android** - Componente principal do sistema.

2. **REST API** - Responsável por cuidar da lógica de negócios. A API desenvolvida atende às requisições feitas pelo aplicativo e se comunica com o banco para armazenar os dados.
3. **Banco de dados do servidor** - É nele que os dados de monitoramento inseridos pelas participantes no aplicativo são armazenados.
4. **Banco de dados do aplicativo** - Onde os dados inseridos pelas participantes são armazenados dentro do aplicativo.
5. **Projeto no Firebase** - Informações básicas de login/registro do usuário também é mantido no Firebase[4] para possibilitar a análise de algumas métricas obtidas pela utilização do aplicativo.
6. **Bucket no S3 da Amazon** - Onde as fotos das refeições das participantes são salvas.

Para entendermos o funcionamento da arquitetura na prática podemos demonstrar três fluxos de interação entre os componentes:

1) Login/Registro do usuário no aplicativo

Para fazer o login/registro do usuário e entrar no aplicativo, os dados inseridos nas respectivas telas são enviados para o servidor e um projeto no Firebase que receberá os dados de autenticação. Após o login/registro ser confirmado nos dois sistemas o usuário entra no aplicativo. Com essa mínima configuração e comunicação com o Firebase podemos usar algumas funções da ferramenta de Analytics do serviço do Google.

2) Inserção de peso no aplicativo

O fluxo mais comum que ocorre no sistema é quando há a inserção/atualização de peso, atividade física ou refeição sem foto. Nesse caso, os componentes 1,2,3 e 4 interagem.

Ao inserir um novo dado de monitoramento no aplicativo, faz-se uma requisição à REST API que recebe essa informação como um JSON e armazena

[4] https://firebase.google.com/?hl=pt-br

esse novo dado no banco do servidor. Após a inserção/atualização no banco, a API retorna a requisição feita pelo aplicativo com as informações fornecidas e mais algumas outras obtidas no servidor, como a id do registro. Ao receber isso, o aplicativo armazena essas novas informações no banco local, assim, havendo consistência entre os dados do servidor e aplicativo.

3) **Inserção de refeição com foto no aplicativo**

Na inserção/atualização de refeição com foto o fluxo segue como o anterior mas agora há a interação do servidor com o Bucket no S3 da Amazon para ocorrer o armazenamento da foto.

Quando há uma requisição com foto à API, o JSON chegará ao servidor com um valor a mais que é a *string* em Base64 que representa a foto enviada. Com isso, o servidor criará um novo objeto representando a foto no banco mas a foto em si estará armazenada no Bucket do S3. Dessa maneira o espaço disponível para a máquina virtual do servidor não será esgotada por causa das fotos enviadas pelas participantes.

4.4 Ferramentas auxiliares

Como visto na seção anterior, na arquitetura detalhada há dois componentes que inicialmente não estavam presentes e agora estão sendo usados para dar mais funcionalidades ao sistema: o Firebase e o S3 da Amazon.

Nesta seção, descreveremos um pouco as duas ferramentas e a importâncias delas na implementação do sistema.

4.4.1 Firebase

Firebase é uma plataforma Backend-as-a-Service que possibilita o desenvolvimento de aplicativos mobile e web sem a necessidade de implementação do servidor, já que o mesmo é fornecido pelo serviço. A plataforma é composta por *features* complementares onde o desenvolvedor pode combiná-las para construir a solução que mais se adequa às suas necessidades.

Dentre as *features* oferecidas pelo Firebase em seu *dashboard*, a principal é o *Firebase Analytics.* O Analytics provê insights sobre o uso do aplicativo e engajamento do usuário ajudando desenvolvedores a entender como os usuários se comportam para, assim, decisões mais precisas serem tomadas.

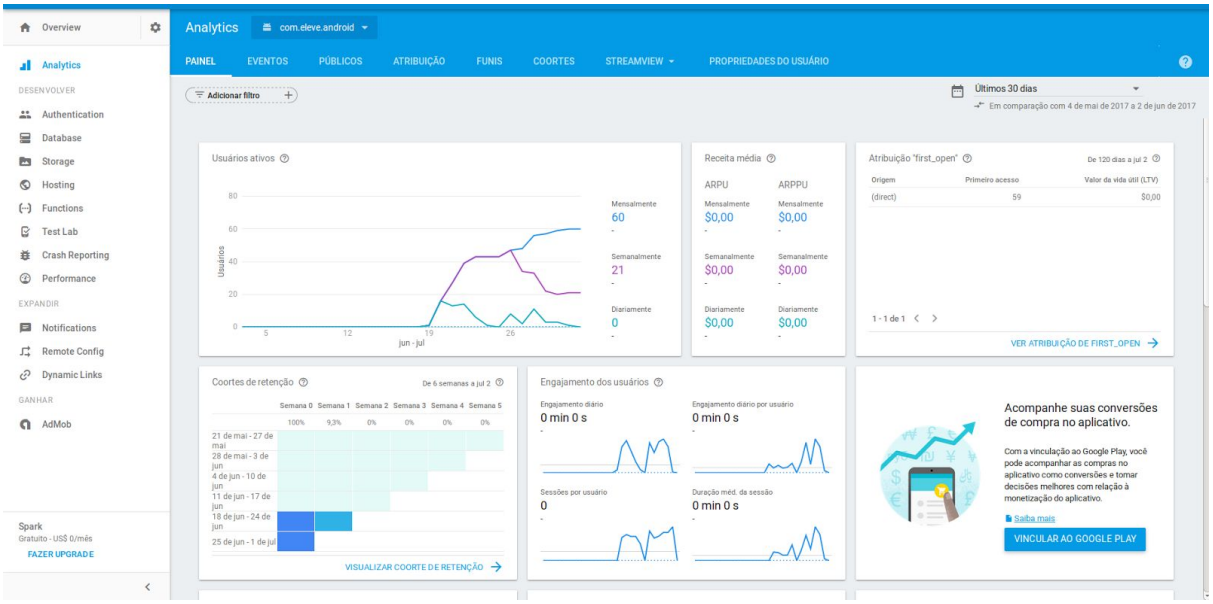


Figura - Dashboard do Firebase Analytics

Instalando o Firebase SDK no aplicativo, o Firebase já começa a obter métricas demográficas, de retenção, engajamento e de renda, porém, muito mais eventos podem ser definidos para serem captados durante a utilização do aplicativo fornecendo uma quantidade enorme de informações sobre o usuário.

Para o desenvolvimento do aplicativo de monitoramento referido neste documento, o Firebase será usado para podermos registrar/logar os participantes na ferramenta de autenticação do mesmo, para assim podermos utilizar o *Analytics*.

Inicialmente, o Analytics será usado para obtermos algumas das métricas principais fornecidas pelo Firebase SDK, como: quantidade de usuários ativos, engajamento diário no aplicativo, duração média da seção, análise da quantidade média de tempo passado em cada tela e modelos dos dispositivos utilizados.

Posteriormente, com o maior entendimento da plataforma outras *features* do Firebase serão utilizadas como o *Crash Reporting*, para a melhora análise de *crashes*, *Performance,* para melhora do desempenho e *Notifications,* para aumentar o engajamento do usuário.

4.4.2 Amazon S3

O *Amazon Simple Storage Service* (Amazon S3) é um serviço de armazenamento de objetos com uma interface de web service simples para armazenar e recuperar qualquer volume de dados de qualquer parte da web.

Esse serviço é acessado pela API do Amazon S3 e com ela podemos criar e manipular *buckets* e objetos que serão armazenados no S3. Os ***buckets*** são os containers fundamentais para o armazenamento de dados, é nele que os objetos são guardados. Os **objetos** são as entidades fundamentais armazenadas no Amazon S3, eles consistem em dados de objetos e metadados.

Além disso, há dois outros conceitos relacionados ao Amazon S3: as chaves e regiões. Uma **chave** é o identificador exclusivo para um objeto dentro de um *bucket*. Como a combinação de um *bucket*, chave e ID de versão identifica de forma exclusiva cada objeto, o Amazon S3 pode ser considerado como um mapeamento entre "balde + chave + versão" e o próprio objeto. Já a **região** é onde o *bucket* é geograficamente criado, escolhida da melhor maneira para otimizar a latência, minimizar custos ou cumprir os requisitos regulamentares.

Assim, para o armazenamento das fotos das refeições enviadas pelos usuários do aplicativo, um *bucket* foi criado para que os objetos das fotos fossem guardadas no mesmo. A partir do bucket criamos uma pasta para cada um dos usuários que enviaram fotos das refeições e nestas pastas as fotos são armazenadas.

Dessa maneira, o servidor se comunica com o Amazon S3 através da API fornecida pela Amazon para que ao uma foto ser inserida/atualizada/deletada o mesmo possa armazenar esses dados no S3 não ocupando espaço no servidor.

4.5 Sprints de desenvolvimento

Pelo Scrum ter sido a abordagem Ágil para a construção do aplicativo, o desenvolvimento foi dividido em ciclos fixos de tempo onde ao final de cada um deles deveríamos entregar um incremento publicável do aplicativo. Como explicado no capítulo anterior, esse ciclo é chamado de *Sprint* e, no nosso caso, utilizamos um período de duas semanas para o mesmo.

Por usar a arquitetura MVP e por ter dois desenvolvedores para implementar o aplicativo, pudemos fazer a separação de funções para cada. Um desenvolvedor era responsável pelo *front-end,* implementando a *View* do MVP e o outro, o autor deste trabalho, era responsável pelo *back-end*, implementando o *Model* e *Presenter* da arquitetura. Com essa separação provida pela arquitetura MVP foi possível atribuir as tarefas iniciais de cada Sprint para cada um dos desenvolvedores de maneira mais fácil.

Nesta seção, veremos como foi a implementação de cada uma das *Sprints* com algumas de suas principais tarefas listadas e poderemos acompanhar a evolução do aplicativo ao longo desse período inicial de desenvolvimento.

4.5.1 Sprint 1 - Monitoramento do Peso

Da lista inicial dos requisitos e user stories mostrados em 3.1, decidimos dividir a implementação do monitoramento em três partes que correspondem a Sprints diferentes. Nessa primeira Sprint escolhemos trabalhar somente com o monitoramento do peso já que por ser a primeira Sprint ela também serviria de experiência para o planejamento das próximas.

Ao final desta Sprint, com o primeiro incremento do aplicativo desenvolvido era possível a execução de alguns dos requisitos iniciais:

- Usuário inserir peso do dia e dos dias anteriores
- Usuário atualizar peso do dia e dos dias anteriores
- Usuário visualizar peso do dia e dos dias anteriores

Nessa primeira Sprint foi implementada a base do design do aplicativo: utilizando-se de um *TabLayout* o usuário poderia ver informações do monitoramento do dia atual, assim como de dias anteriores ao deslizar a tela.

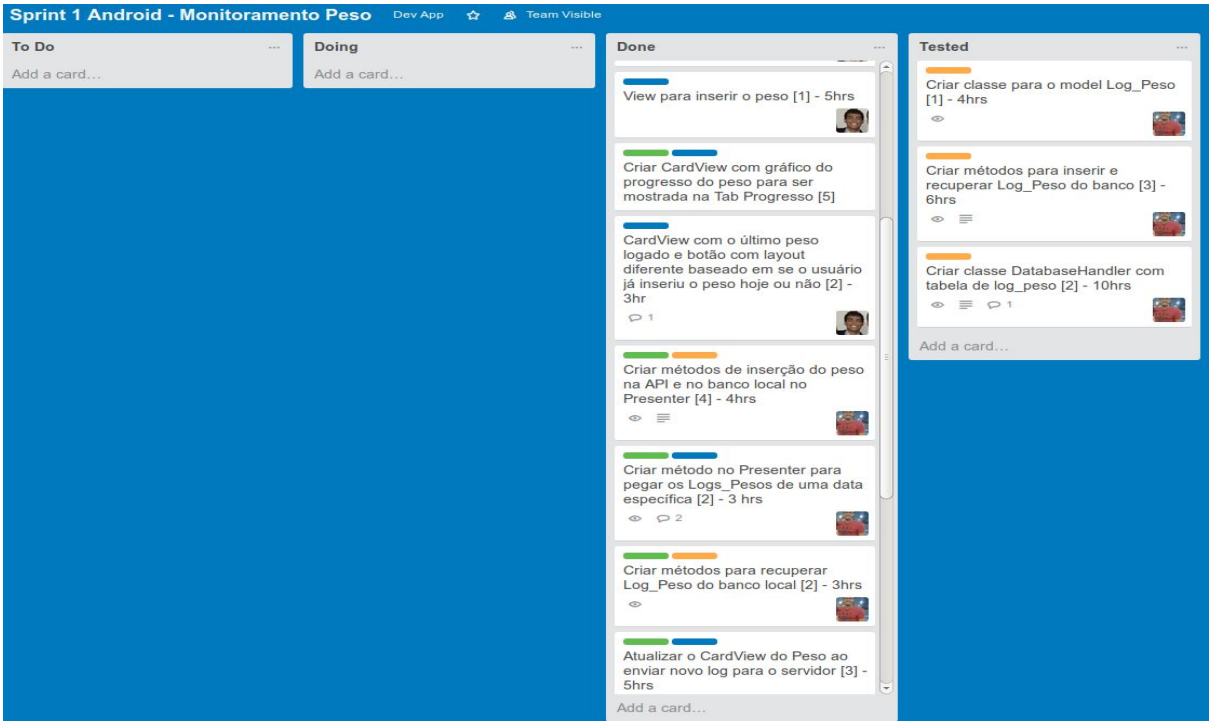


Figura - Tarefas da Sprint 1

Na figura acima podemos ver algumas das tarefas implementadas na primeira Sprint de desenvolvimento. Para podermos visualizar as tarefas de cada Sprint de maneira mais fácil e para fazermos o acompanhamento da evolução de cada uma das tarefas ao longo da Sprint, usamos o *KanBan Board* na ferramenta online Trello.

4.5.2 Sprint 2 - Monitoramento Refeição e Login

Após a experiência com a carga de trabalho da primeira, planejamos a segunda Sprint inserindo mais atividades na mesma, porém, com isso não conseguimos entregar um incremento publicável ao final. Com esse erro de planejamento, na *Sprint Retrospective,* tivemos a oportunidade de fazer uma análise maior dos problemas que ocorreram e houve um bom aprendizado.

Nessa Sprint, mesmo não havendo um incremento com todas as funcionalidades completas, ao final deste ciclo era possível a execução de algumas tarefas:

- Usuário insere refeição do dia e dos dias anteriores
- Usuário visualiza refeição na lista de refeições do dia
- Usuário consegue efetuar login/registro/logout no aplicativo
- Usuário visualiza o progresso do peso durante o programa

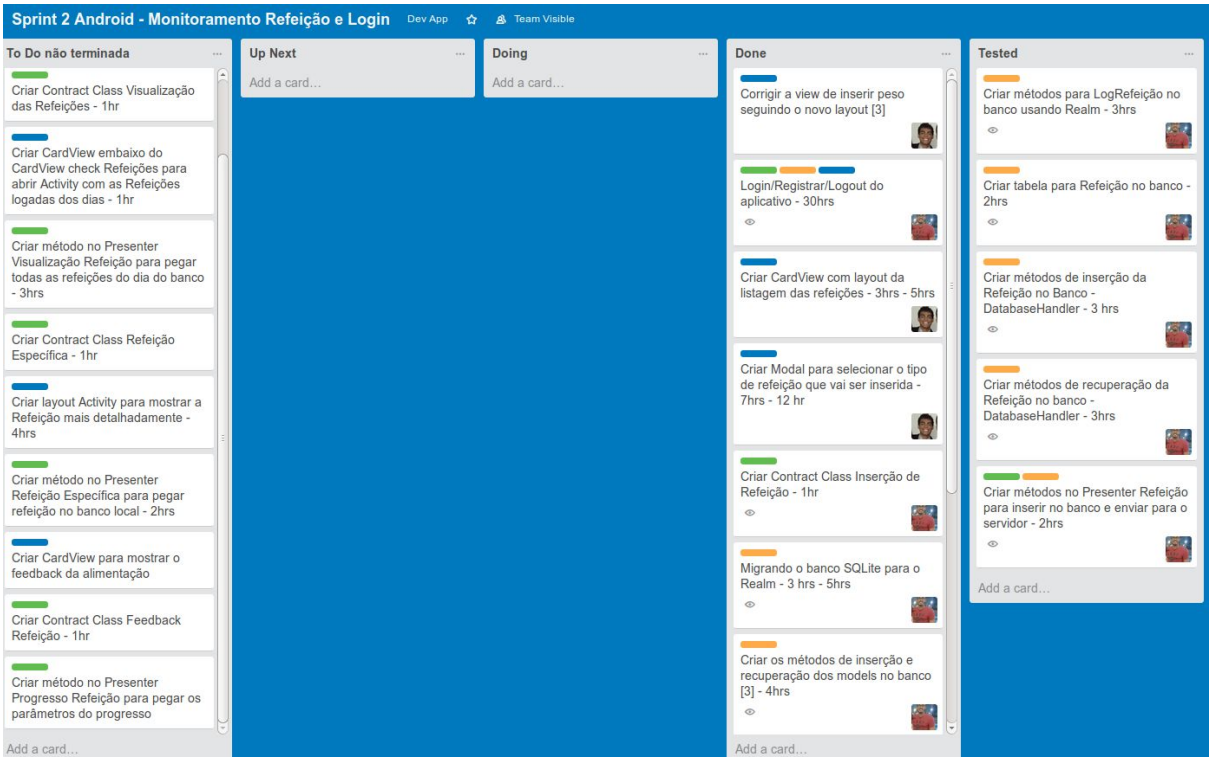


Figura - Tarefas da Sprint 2

Algumas das telas que foram terminadas ao final desta Sprint e da anterior podem ser vistas abaixo:

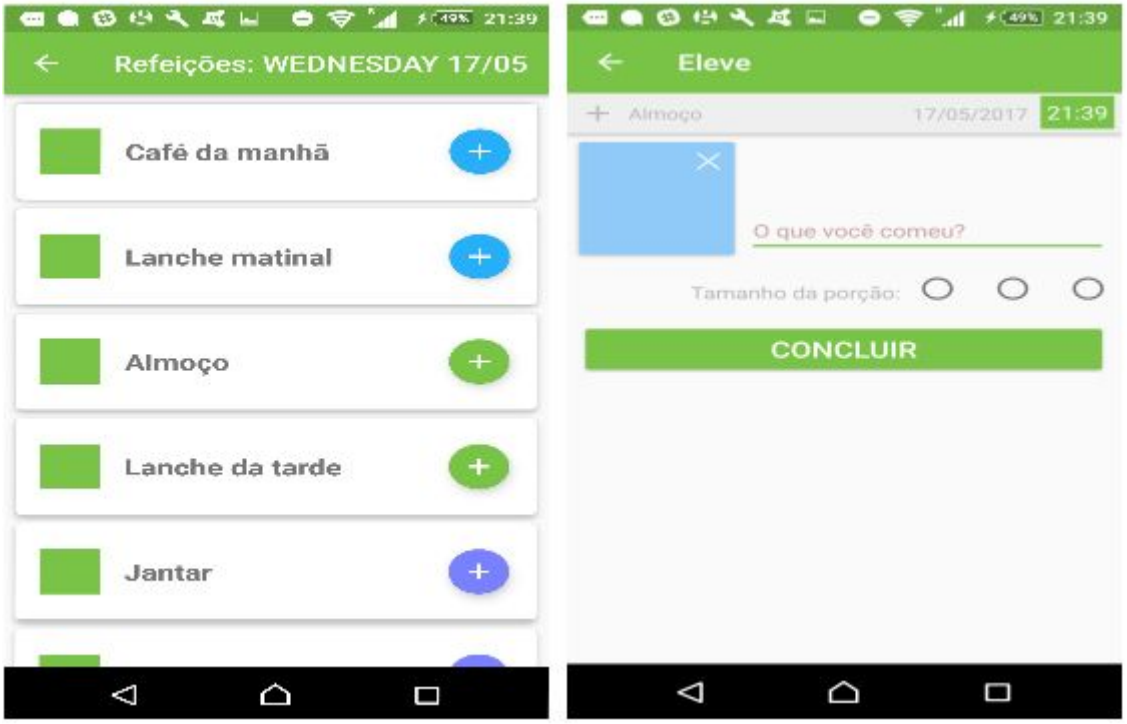


Figura - Telas de refeições do dia e inserção de refeição

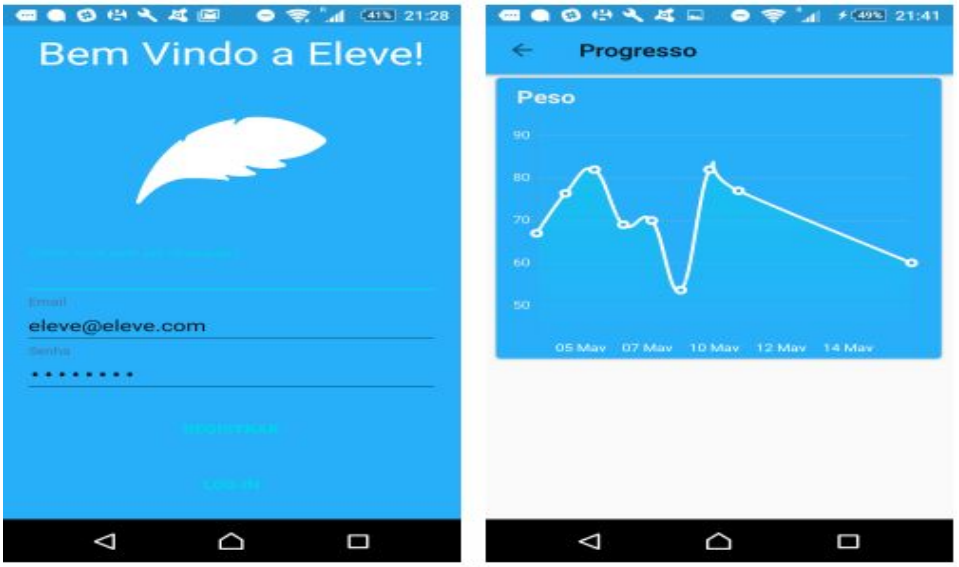


Figura - Telas de login/registro e de progresso do peso

4.5.3 Sprint 3 - Monitoramento Atividade Física

Nesta terceira Sprint, tarefas que ainda estavam incompletas da Sprint anterior foram levantadas como prioridade e houve a criação de novas tarefas para a parte de monitoramento de atividade física.

Nesta Sprint mais funcionalidades do *product backlog* foram adicionadas ao aplicativo:

- Usuário visualiza de maneira mais detalhada as refeições inseridas
- Usuário edita as refeições do dia e de dias anteriores
- Usuário insere foto associada à refeição
- Usuário insere atividade física do dia e de dias anteriores
- Usuário visualiza atividade física do dia e de dias anteriores
- Usuário edita atividade física do dia e de dias anteriores
- Usuário visualiza o progresso em relação à atividade física ao longo do programa

Com isso, ao final desta Sprint tivemos todos os requisitos levantados inicialmente implementados. Porém, esta versão ainda não foi a publicada na loja de aplicativos da Google por faltar fazer a correção de alguns *bugs* levantados à

medida que testamos a utilização do aplicativo internamente, o que ocorreu na próxima *Sprint*. Abaixo, vemos algumas das telas ao término da Sprint:

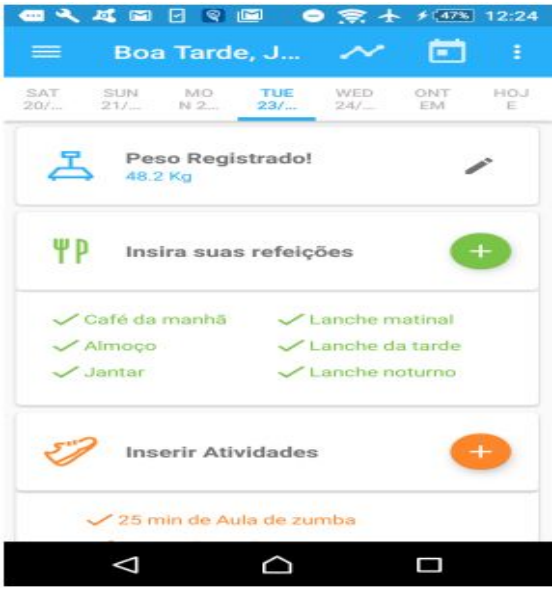


Figura - Tela principal do aplicativo

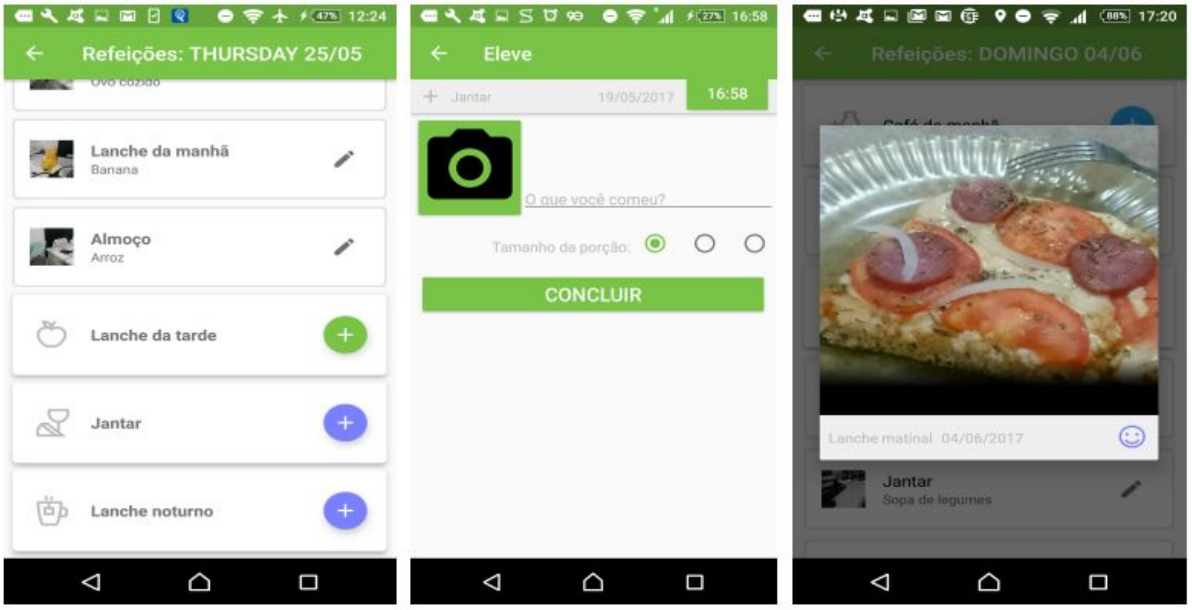


Figura - Telas do monitoramento refeição

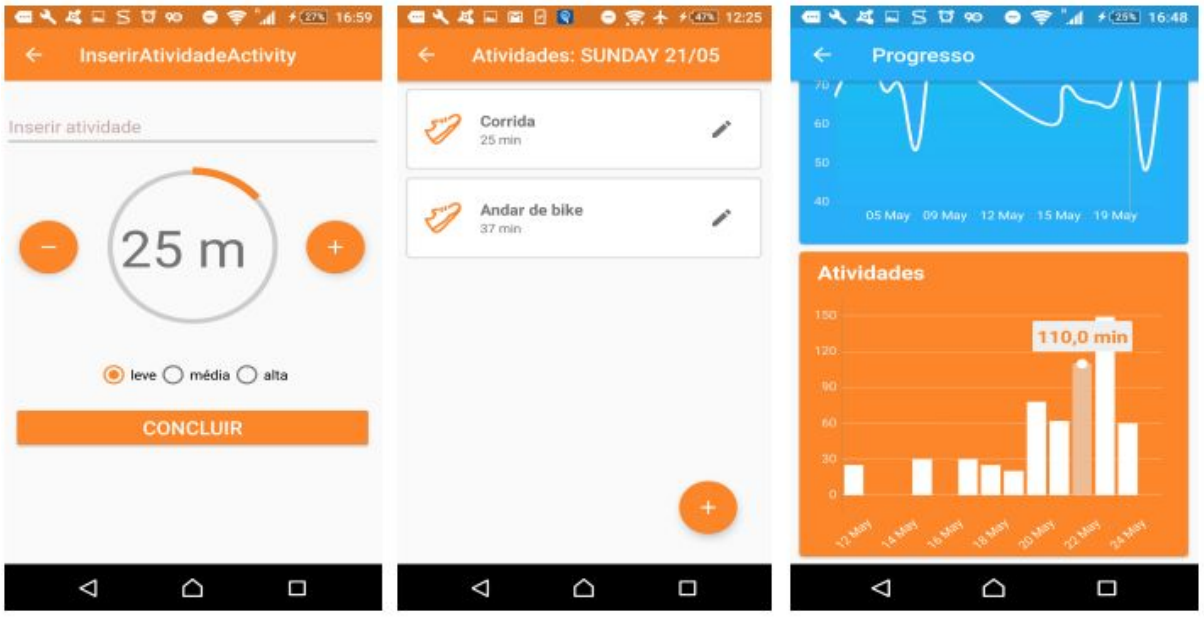


Figura - Telas do monitoramento atividade e Tela de Progresso

4.5.4 Sprint 4 - Ajustes pré-lançamento na loja

Para fazer o lançamento do aplicativo no Google Play, nesta última Sprint, continuamos com o desenvolvimento do aplicativo Eleve com a implementação de um novo incremento que possibilitará o consumo do conteúdo semanal fornecido durante o programa, e corrigimos alguns bugs que apareceram após a utilização do *app* internamente.

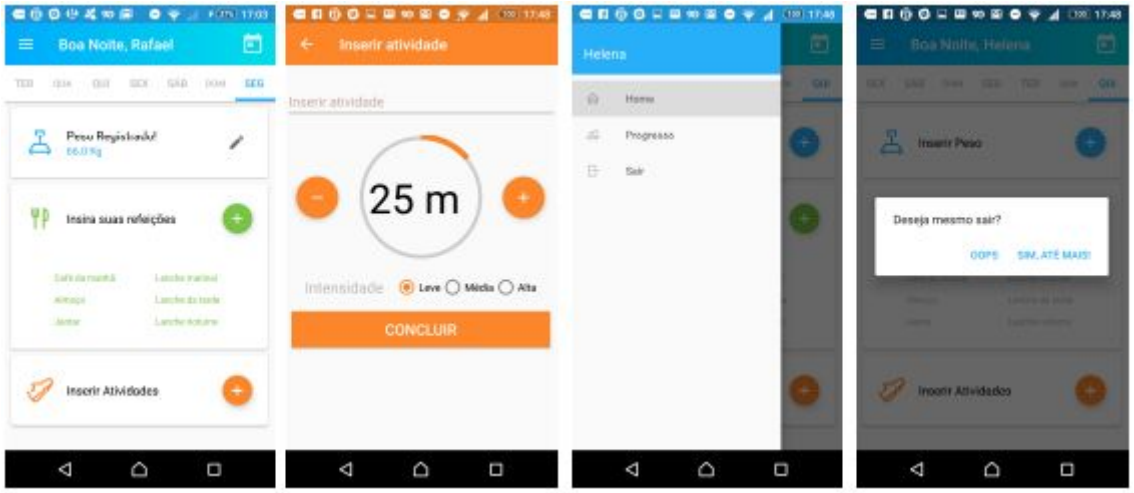


Figura - Simples mudanças visuais antes da publicação

Acima vemos algumas das mudanças que ocorreram nas telas. As tabs da tela principal do aplicativo agora têm o texto modificado(com data e dia da semana ou somente o dia) de acordo com o tamanho da tela do smartphone e para sair do aplicativo o usuário tem que confirmar a ação.

4.6 Testes e publicação do aplicativo

Com as últimas modificações feitas no aplicativo, o mesmo foi publicado no Google Play com dois objetivos iniciais. O primeiro objetivo da publicação era que pudéssemos aprender a publicar e atualizar versões do aplicativo para que isso não fosse um problema quando participantes já estivessem utilizando o aplicativo. Além disso, com o aplicativo publicado pudemos, de maneira mais fácil, fazer testes com mulheres no perfil das participantes do programa Eleve para que analisássemos algumas das possíveis dificuldades na utilização.

Para os testes, organizamos um roteiro de tarefas que seriam comumente realizadas pelas participantes do programa Eleve e testamos o uso do aplicativo com três mulheres. Da lista de atividades, podemos listar algumas abaixo:

1. Entrar no aplicativo fazendo registro
2. Registrar peso
3. Editar peso
4. Registrar uma refeição somente com descrição
5. Registrar uma refeição com descrição e foto
6. Visualizar a lista de refeições registradas no dia
7. Registrar uma atividade física
8. Editar uma atividade física
9. Registrar peso de um dia anterior
10. Visualizar o progresso do peso na tela de Progresso

Para realizarmos os testes deixamos as participantes livres para a utilização do aplicativo e observamos e gravamos a interação delas com o mesmo. Ao término do teste fizemos algumas perguntas sobre como foi a experiência com o aplicativo, sobre a execução das tarefas, das dificuldades que tiveram e da opinião geral sobre o *app*.

Com os testes, foi possível vermos alguns problemas que dificultariam a utilização da ferramenta, alguns erros de design que induziram as participantes a clicarem em partes que inicialmente estavam sem interação e elementos que passaram despercebidos durante o uso do aplicativo. Feitas as anotações e listados os pontos de melhora, fizemos os últimos ajustes antes de publicar a ferramenta.

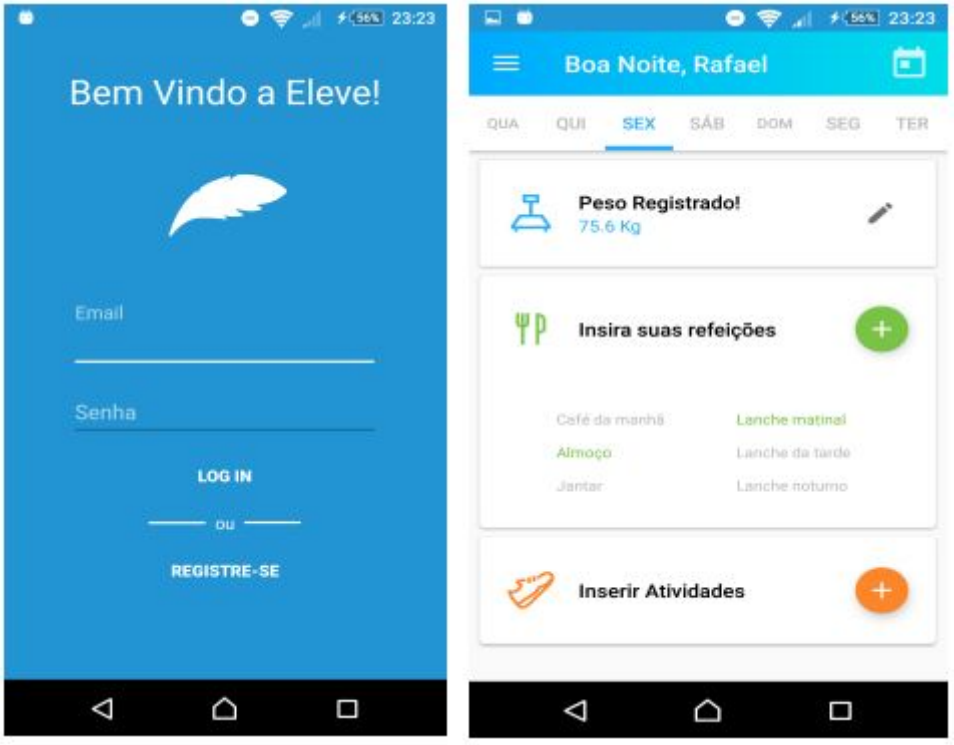


Figura - Nova tela de login e interação na tela principal

As maiores modificações feitas depois dos testes foram na tela de login/registro e na forma como os *cards* das telas são clicáveis. Com o *layout* da tela de login/registro anterior, as participantes do teste não viram que para entrar no aplicativo pela primeira vez teriam que clicar em um botão na parte de baixo da

tela(“É minha primeira vez”) para ir à outra tela de registro. Com essa dificuldade, elas inicialmente inseriam o email e senha para tentar fazer login e só depois quando um erro de autenticação era levantado as participantes percebiam que havia outra tela para registro. Outro problema que foi resolvido de maneira simples, foi a habilitação dos *cards* para serem clicáveis ao invés dos usuários só poderem adicionar um novo registro ao clicar no botão de adição que flutua sobre o *card*.

Com essas modificações feitas a partir dos resultados obtidos pelo teste com mulheres do mesmo perfil das participantes do programa Eleve, pudemos realmente divulgar o aplicativo nas mãos de algumas das participantes para as mesmas poderem monitorar de um jeito mais fácil.

4.7 Utilização do aplicativo por algumas participantes

Após a publicação do aplicativo na loja, o divulgamos para um grupo de três participantes, entre elas duas mais engajadas no monitoramento na última semana antes de lançarmos o aplicativo, para as mesmas utilizarem o *app* à medida que observamos algumas métricas para realizarmos um teste beta fechado. Nos testes betas fechados, o app, em versões estáveis, é testado por um grupo menor de usuário (podendo também ser da própria empresa).

Inicialmente, as métricas analisadas são algumas das quais podemos obter facilmente e que demonstram a utilização do aplicativo pelas participantes. Assim, nesse primeiro momento estamos observando:

- Quantidade de refeições registradas
- Quantidade de refeições inseridas com foto
- Quantidade de vezes em que o peso foi registrado
- Quantidade de atividades físicas que foram registradas
- Quantidade de erros reportados no Google Play Console
- Média de vezes por dia em que entravam para registrar no aplicativo

Dessas métricas, a única que não é obtida verificando o servidor do sistema é a quantidade de erros registrados, a qual é vista no Google Play Console. O Play Console serve como um local para os desenvolvedores de aplicativos publicarem e acompanharem o status dos aplicativos Android na loja, além de fornecer algumas estatísticas como número de instalações, dispositivos utilizando o app, avaliações e erros, entre várias outras funcionalidades.

Pela limitação de tempo entre a divulgação do aplicativo para essas participantes e o término da documentação deste trabalho, os dados obtidos ainda não foram grandes, porém as mesmas métricas estão sendo acompanhadas à medida que implementamos próximos ajustes e incrementos à ferramenta.

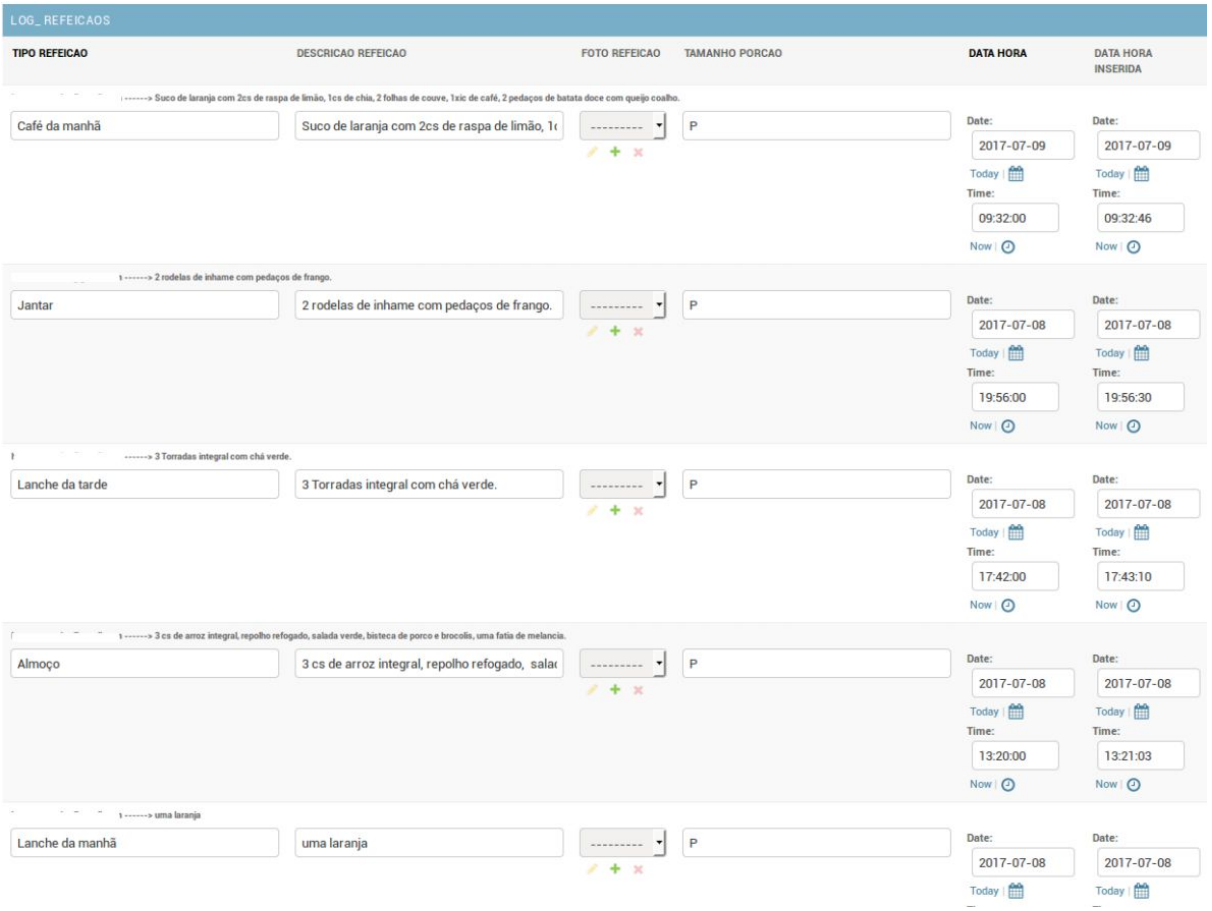


Figura - Acompanhamento de refeições no servidor

Da utilização do aplicativo pelas 3 participantes em que o divulgamos, foram registradas 52 refeições, sendo 6 delas acompanhadas de foto, o que mostra que

esta ainda não é uma opção tão acionada por elas. O monitoramento do peso foi feito um total de 13 vezes entre elas e 10 atividades físicas foram registradas.

Por participantes teríamos:

A. 16 Refeições - 3 com foto / 3 Registros de peso / 6 Registros de atividade física
B. 17 Refeições - 0 com foto / 7 Registros de peso / 4 Registros de atividade física
C. 20 Refeições - 3 com foto / 3 Registros de peso / 0 Registros de atividade física

Outro dado que obtivemos é que as participantes que utilizaram o app normalmente registravam as refeições várias vezes ao dia, só tendo alguns dias em que registraram diferentes refeições ao mesmo tempo, por exemplo à noite. E os registros de atividade física ocorriam normalmente depois da realização dos mesmos.

A medida em que divulgarmos o aplicativo para as outras participantes e o mesmo for mais utilizado, obteremos mais dados sobre o engajamento das usuárias. Com isso, poderemos utilizar mais métricas fornecidas pelo Firebase Console, como o número médio de sessão por usuária, tempo médio das sessões e quais telas são mais utilizadas, entre outras métricas que poderão ser coletadas ao entendermos mais as funcionalidades do Firebase Console.

5. Conclusão e trabalhos futuros

Este trabalho é o resultado da implementação e documentação do processo de desenvolvimento do aplicativo de monitoramento da Eleve. Graças a estruturação e construção da documentação foi possível aprender mais sobre alguns pontos importantes para o desenvolvimento de qualquer ferramenta, principalmente em relação às metodologias e arquiteturas. Através das pesquisas e estudos necessários para a escrita deste trabalho, pudemos revisar conceitos e enxergar pontos de melhoria nos dando *insights* para como prosseguir com o desenvolvimento da ferramenta para as próximas versões.

O aplicativo implementado descrito por este trabalho possibilita que o participante do programa digital da Eleve possa fazer o acompanhamento da sua evolução em relação a sua reeducação alimentar e criação de novos hábitos. Com ele, os usuários podem registrar a sua alimentação de maneira detalhada com descrição e foto, colocar o seu peso e inserir atividades físicas praticadas. Para acompanhar o seu progresso, há uma seção com gráficos de peso e atividades físicas e o usuário pode rever a alimentação de dias anteriores de maneira intuitiva.

Porém, o aplicativo mostrado por este trabalho é só uma primeira versão, representando o módulo de monitoramento, do que estamos ainda desenvolvendo. Os testes que foram feitos e relatados nesta documentação serviram para tirarmos algumas conclusões iniciais sobre a interação com o aplicativo e já nos possibilitou implantar simples melhorias antes mesmo do lançamento para as participantes.

Devido a limitação de tempo, algumas outras melhorias ficaram fora do escopo deste trabalho. Porém, essas adaptações e outras novas funcionalidades já estão sendo desenvolvidas para se integrar ao aplicativo de forma incremental.

Para as novas versões da ferramenta, estamos concluindo o módulo de Conteúdo, onde a participante do programa terá acesso a conteúdos publicados para ajudá-las a entender e colocar em prática o processo de reeducação alimentar e da criação da rotina de atividades físicas. Além disso, nas próximas versões desenvolveremos funcionalidades para as participantes receberem *feedbacks* em

relação ao que registram no aplicativo de modo instantâneo e notificações para ajudá-las criação dessa nova rotina, notificando, por exemplo, sobre sugestões de substituição de alimentos em horários próximos às refeições.

Em uma etapa posterior, desenvolveremos o módulo de Metas, onde os coaches do programa poderão criar metas com as participantes e elas poderão registrar e acompanhar a evolução dentro do aplicativo. Por último, integraremos o chat entre o coach e participante no aplicativo, assim como um feed onde o grupo de participantes poderão interagir.

Referência Bibliográficas

1. World Health Organization. **MHealth**: New horizons for health through mobile technologies, 2011.
2. RESEARCH2GUIDANCE. **MHealth App Developer Economics 2016:** The current status and trends of the mHealth app market, 2016.
3. World Health Organization. **Noncommunicable diseases country profiles 2014**, 2014.
4. ARMENIO, Letizia et al. **Trends of chronic diseases and partnership between Big Pharma and Patient Associations.** Gd: Fondazione Istud, 2013.
5. VIGITEL. **VIGITEL BRASIL 2016:** Vigilância de fatores de risco e proteção para doenças crônicas por inquérito telefônico, 2016.
6. HOLLIS, Jack F. et al. Weight Loss During the Intensive Intervention Phase of the Weight-Loss Maintenance Trial. **American Journal Of Preventive Medicine**, p. 118-126. ago. 2008.
7. KONG, Angela et al. Self-Monitoring and Eating-Related Behaviors Are Associated with 12-Month Weight Loss in Postmenopausal Overweight-to-Obese Women. **Journal Of The Academy Of Nutrition And Dietetics**, p. 1428-1435. set. 2012.
8. WASSERMAN, Anthony I.. **Software Engineering Issues for Mobile Application Developme.** Moffett Field, 2010.
9. EL-KASSAS, W. S. et al. **Taxonomy of Cross-Platform Mobile Applications Development Approaches**. Ain Shams Engineering Journal, p. 163-190. jun. 2015.
10. PEREZ, Sarah. **App downloads up 15 percent in 2016, revenue up 40 percent thanks to China.** Disponível em: <https://techcrunch.com/2017/01/17/app-downloads-up-15-percent-in-2016-revenue-up-40-percent-thanks-to-china/>. Acesso em: 17 jan. 2017.
11. STATISTA. **Number of available applications in the Google Play Store from December 2009 to June 2017.** Disponível em:

<https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>. Acesso em: 02 jul. 2017.

12. ROUSE, Margaret. **Systems development life cycle (SDLC).** 2009. Disponível em: <http://searchsoftwarequality.techtarget.com/definition/systems-development-life-cycle>. Acesso em: 05 jul 2017.
13. AGILEALIANCE. **What is Agile?** Disponível em: <https://www.agilealliance.org/agile101/>. Acesso em: 04 jul. 2017.
14. STEFFEN, Juliana Berossa. **O que são essas tais de metodologias Ágeis ?** 2012. Disponível em: <https://www.ibm.com/developerworks/community/blogs/rationalbrasil/entry/mas_o_que_s_c3_a3o_essas_tais_de_metodologias__c3_a1geis?lang=en>. Acesso em: 02 jul. 2017.
15. WELLS, Don. **Extreme Programming: A gentle introduction.** 2009. Disponível em: <http://www.extremeprogramming.org/>. Acesso em: 27 jun. 2017.
16. SUTHERLAND, Jeff; SCHWABER, Ken. **The Scrum Guide:** The Definitive Guide to Scrum : The Rules of the Game, 2016.
17. **SPRINT Backlog**. Disponível em: <https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/sprint-backlog>. Acesso em: 28 jun. 2017.
18. MORRIS, Bryan. **Yes We Kanban! Introducing an Agile Methodology to Manage Your Team**, 2012.
19. LOU, T. **A comparison of Android Native App Architecture MVC, MVP and MVVM.** Aalto, 2016.
20. POTEL, M. **MVP: Model-view-presenter the taligent programming model for C++ and java**, Taligent Inc, p. 20, 1996.
21. FOWLER, Martin. **Supervising Controller.** 2006. Disponível em: <https://martinfowler.com/eaaDev/SupervisingPresenter.html>. Acesso em: 29 jun. 2017.

22. FOWLER, Martin. **Passive View.** 2006. Disponível em: <https://martinfowler.com/eaaDev/PassiveScreen.html>. Acesso em: 29 jun. 2017.
23. **Android Architecture Blueprint.** 2016. Disponível em: <https://github.com/googlesamples/android-architecture>. Acesso em: 29 jun. 2017.
24. GOSSMAN, John. **Introduction to Model/View/ViewModel pattern for building WPF apps.** 2005. Disponível em: <https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/>. Acesso em: 30 jun. 2017.
25. **MVVM architecture with the data binding library**. 2016. Disponível em: <https://nullpointer.wtf/android/mvvm-architecture-data-binding-library/>. Acesso em: 05 jul. 2017.
26. **Implementing the MVVM Pattern Using the Prism Library 5.0 for WPF**. Disponível em: <https://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx>. Acesso em: 05 jul. 2017.
27. HIGA, Paulo. **95,5% dos smartphones vendidos no Brasil são Androids.** 2016. Disponível em: <https://tecnoblog.net/203749/android-ios-market-share-brasil-3t-2016/>. Acesso em: 10 abr. 2017.