UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

Bacharelado em Ciência da Computação

Processo Web Individual: Uma Investigação-Ação de Adequação de Práticas Ágeis para um Desenvolvedor Individual

Pedro Torchio

Recife, Dezembro de 2018

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

Processo Web Individual: Uma Investigação-Ação de Adequação de Práticas Ágeis para um Desenvolvedor Individual

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 de Bacharel em em Ciência da Computação

Aluno: Pedro Torchio
Orientador: Alexandre Marcos Lins de Vasconcelos

Dezembro/2018

Agradecimentos

Minha história acadêmica não me permitiria encerrá-la sem demonstrar gratidão aos meus colegas e amigos, que não apenas tornaram leves os momentos mais difíceis, como também ofereceram, ou foram coagidos a oferecer, caronas em dias de sol e chuva. Com mais relevância, agradeço aos meus pais. Meu pai por ter fortalecido em mim o gosto pelas ciências exatas e, acima de tudo e todos, minha mãe por ter me sustentado e fortalecido por nove anos de faculdade.

Resumo

O crescente alcance da infraestrutura da internet de alta velocidade impactou na demanda por desenvolvedores de websites ou de pequenas aplicações Web. Mais profissionais na área passaram a produzir mais conteúdo e guias técnicos, o que causou uma explosão de desenvolvedores medianos, que aprenderam a desenvolver por conta própria e trabalham sozinhos. Esta modalidade de profissionais possui pouca ou nenhuma educação formal em desenvolvimento de software e por esta razão nunca teve contato com Engenharia de Software. Este trabalho foca em elaborar um processo individual leve, baseado nos princípios da metodologia Ágil, que se adeque às necessidades deste novo profissional da internet.

Palavras-chave: Processo de Desenvolvimento, *Freelancer*, Desenvolvedor Web

Sumário

1	Introdução	6
2	Desenvolvimento Web	9
2.1	Arquitetura	10
2.2	Plataforma	12
2.3	Modularidade	13
2.4	Princípios	13
3	Manifesto Ágil e Extreme Programming	17
3.1	Extreme Programming	18
4	Investigação-Ação	25
4.1	PDCA	25
5	Adequação do Processo	27
5.1	Processo Inicial	28
5.2	Ciclos de Melhoria de Processo	30
5.3	Problema Proposto	31
5.4	Sprint 0 - Configuração	32
5.5	Sprint 1 - Pacientes	43
5.6	Sprint 2 - Autorização	54
5.7	Sprint 3 - Rotinas	66
5.8	Sprint 4 - Visualização	76
5.9	Sprint 5 - Preenchimento	87
5.10	O Processo Final	98
6	Conclusão	107

1 Introdução

Segundo estatísticas dos Estados Unidos e Europa, trabalhadores autônomos, também conhecidos como *freelancers*, trabalham majoritariamente sozinhos e devem se tornar maioria em até 10 anos (MANYIKA, 2016; NOGUCHI, 2018). Hoje já somam quase 50% dos trabalhadores das gerações entre 25 e 35 anos, os chamados *Millennials* (UPWORK, 2017). Também representam quase 30% de todos os trabalhadores (UPWORK, 2018).

Apesar de haver poucas amostras estatísticas tratando de engenheiros de software autônomos, é possível observar um enorme número destes profissionais em plataformas de trabalho *freelance* (foram observados https://www.upwork.com e https://www.freelancer.com). Este é um tipo de profissional compatível com projetos menores e menos arriscados, pois não contam com uma equipe ao seu lado, que poderia garantir a robustez do produto em tempo hábil. Desenvolvedores autônomos têm que lidar sozinhos com os mesmos prazos e níveis de qualidade exigidos de uma empresa provida de uma equipe completa. No entanto, a falta de planejamento e infraestrutura pode tornar o desenvolvimento com qualidade uma tarefa impossível para certos projetos, reduzindo as possibilidades do profissional *freelancer*, mesmo que este seja plenamente capacitado em conhecimentos técnicos.

Essas limitações podem ser bastante reduzidas com a utilização de metodologias de gestão de projeto de software (PRESSMAN, 2005). Porém, muitos desenvolvedores sequer seguem qualquer processo. Dentre os que seguem algum processo, a maioria adere a alguma adaptação de metodologias Ágeis, por serem menos burocráticas.

Existe um processo para indivíduos chamado Personal Software Process (PSP), o qual é baseado no CMM (o modelo antecessor ao CMMI) (GALIN, 2018), porém ainda é um tanto burocrático. Um grupo de estudantes da Faculdade de Matemática e Informática da Sofia University, Bulgária, descreveu brevemente como seria uma

interpretação Ágil do PSP, seguindo a metodologia Extreme Programming (XP), resultando no chamado PXP (DZHUROV, 2009). Este estudo, porém, não resultou num processo bem definido, mas sim num documento de recomendações muito breves, que apenas cita algumas fases, mas não ajuda o desenvolvedor sem conhecimento de processos mais complexos a adotar um processo individual baseado na XP.

Dentre as diversas áreas onde um desenvolvedor individual pode trabalhar com competência, está a de Aplicações Web. O nível técnico exigido de desenvolvedores Web é mais baixo que de desenvolvedores de aplicações nativas, devido a grande abstração entregue pelos navegadores. Somado a isso, houve um aumento do alcance da infraestrutura da internet. Consequentemente, houve também um aumento na demanda por desenvolvedores de websites ou de pequenas aplicações Web. Mais profissionais na área passaram a produzir mais conteúdo e guias técnicos, o que causou uma explosão de desenvolvedores medianos, que aprenderam a desenvolver por conta própria e trabalham sozinhos. O grupo Data USA estima um aumento constante de desenvolvedores Web nos próximos anos, acima do crescimento da força de trabalho total (DATA USA, 2018). Esta modalidade de profissionais possui pouca ou nenhuma educação formal em desenvolvimento de software (UPWORK, 2018) e por esta razão nunca teve contato com Engenharia de Software. Um processo individual leve, baseado nos princípios da metodologia Ágil, se adequa perfeitamente as necessidades deste novo profissional da internet.

O desenvolvimento para a internet se transformou muito com o surgimento de *smart phones*, celulares que na realidade são computadores de bolso. Muitos websites hoje são desenvolvidos para serem verdadeiros substitutos de seus equivalentes nativos, tanto para *desktop*, quanto *mobile*. PWAs (Aplicações Web cuja experiência do usuário se assemelha a de aplicações nativas, utilizando-se de recursos Web avançados) são a nova tendência em termos de tecnologia para software (AGRAWAL, 2018). Dessa forma, desenvolver programas para a web tem muito em comum com desenvolver programas desktop, porém por atingirem mais dispositivos, possuem algumas peculiaridades. Além disso, devido ao movimento *open source*, as tecnologias

web no geral evoluem com imensa rapidez, surgindo novos recursos e práticas a todo momento (THE NPM BLOG, 2018).

O propósito deste trabalho será descrever a execução da metodologia de investigação-ação em um projeto real, usando práticas conhecidas da metodologia ágil. O objetivo será refinar um processo de desenvolvimento voltado para o desenvolvedor Web individual. Ao final deverá ser possível extrair da investigação-ação fluxo de práticas e recomendações.

O capítulo 2 trará uma introdução ao desenvolvimento Web, com conceitos e especificidades, que o diferenciam do desenvolvimento nativo. No capítulo 3 há uma introdução à metodologia Ágil e a apresentação das práticas e recomendações mais relevantes para serem utilizadas por um indivíduo. O capítulo 4 traz um planejamento do processo investigativo que será executado. No capítulo 5 é executada a metodologia de investigação-ação, onde cada ciclo investigativo ocupará uma nova seção, resultando numa última seção com o processo final.

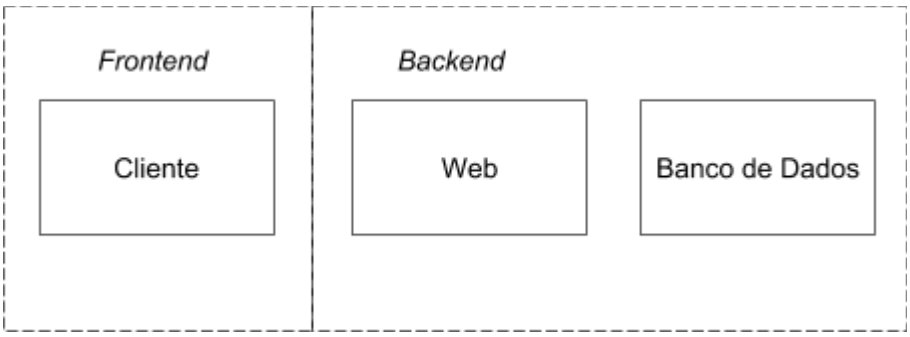
2 Desenvolvimento Web

A *World Wide Web*, mais conhecida pela sigla *WWW* ou apenas *Web*, é a rede de recursos interligados disponíveis sobre a infraestrutura da Internet. Em outras palavras, é o conjunto de documentos, mídias, ou qualquer outro formato de dados, também chamados de recursos, que se identificam por URLs e se conectam através de hiperlinks. A proposta da Web, escrita por Tim Berners-Lee em 1990, trouxe consigo importantes inovações tecnológicas. Além do identificador URL, também a linguagem descritiva HTML e o protocolo de distribuição de recursos HTTP (WEB FOUNDATION, 20--).

Qualquer tipo de recurso digital pode ser colocado na Web para ser facilmente alcançado, desde trabalhos artísticos, como imagens, vídeos e músicas, até trabalhos intelectuais, como monografias científicas e até software de computador (BERNERS-LEE, 2002). Estes, porém, são recursos que não foram criados exclusivamente para a Web, mas apenas foram distribuídos nela digitalmente. Existe um tipo de recurso mais específico para a Web, que exige conhecimento técnico bastante específico. Eram inicialmente os Websites e agora também, com mais relevância, Aplicações Web.

As tecnologias presentes tanto nos Websites como nas aplicações Web são as mesmas. Ambos são compostos por um conjunto de arquivos de código estrutural, em HTML, de descrição visual, em CSS e de procedimentos, scripts em Javascript. Tais arquivos são disponíveis em um servidor Web e repassados a um cliente, comumente um navegador Web, onde são interpretados. O maior ponto de discernimento está na interatividade com o usuário. Uma aplicação Web exige maior interação e, por esta razão, utiliza armazenamento em memória local e comunicação com o dispositivo físico (NATIONS, 2018; DROIDHEAD, 2017; WEST, 2013). Além disso, uma aplicação tende a envolver mais requisições ao servidor, para ler ou gravar dados, o que impacta diretamente na distribuição da arquitetura final do sistema.

Antes de apresentar uma definição mais técnica de aplicação Web, é necessário se fazer uma distinção entre dois conceitos que costumam ser considerados sinônimos, especialmente na tradução para português. Existem camadas, também conhecidos como *layers*, que representam uma separação lógica entre conjuntos de componentes de um software. Por outro lado, também existem *tiers*, que possui tradução semelhante, mas representa uma separação não apenas lógica, mas também física. Separação física não necessariamente geográfica, bastando encontrar-se em um diferente sistema de arquivos, como um container num mesmo computador (STACKIFY, 2017; MICROSOFT, 2014). Dessa forma, a camada de dados que utiliza um Sistema de Gerência de Banco de Dados (*DBMS*), pode ser considerada um tier. Aqui, apenas o consideraremos um *tier* quando assim for explicitamente utilizado. A Figura 2.1 ilustra uma arquitetura 3 tier, separando Cliente, Web e Banco de Dados e também uma 2 tier, separando *Frontend* e *Backend*.



Arquitetura 3-tier

2.1 Arquitetura

Um website é arquitetado em duas camadas superiores, chamadas *frontend* e *backend*. No *backend*, o servidor Web, ao receber uma requisição, executa um programa pré definido. Este busca a informação adequada no servidor de Dados, a processa e repassa para a camada de *frontend*, obtendo um arquivo final de resposta. Aqui, o cliente tem função reduzida, dotado apenas da capacidade de renderizar o arquivo de resposta, enquanto o servidor retém toda a lógica e dados. O *backend* é

equivalente às camadas de Negócio e Dados, enquanto o *frontend* equivale à camada de Apresentação.

Como apenas há busca por dados durante a requisição inicial pela página, não há grande demanda do *backend*. Dessa forma, ambas as camadas muito frequentemente se encontram num mesmo servidor, sendo uma aplicação *single-tier* (STACKIFY, 2017).

Como já comentado, aplicações Web possuem uma carga de trabalho no cliente muito maior que em simples websites, onde a camada de *frontend* representa sozinha um programa completo, não apenas uma camada lógica. Uma aplicação Web tende a envolver mais requisições de dados que um website. Isso não é verdade para todos os casos, pois uma aplicação pode executar inteiramente no cliente, como uma calculadora, ou editor de texto, sendo assim também um *single-tier*. Para a maioria das aplicações com algum nível de complexidade, porém, teremos dados fluindo entre o *frontend* e o *backend*. Será necessário pelo menos mais um *tier*, responsável pelas camadas de Negócios e Dados. Com dois *tiers*, *frontend* e *backend*, é possível fazer uma aplicação Web bastante funcional.

Devido às características distribuídas da internet e das divisões de responsabilidade, cada *tier* implementa uma aplicação independente das outras, podendo ser tratadas como projetos completamente separados. Uma organização pode atribuir cada projeto a uma nova equipe, que deve apenas seguir rigidamente os contratos de interface. Esta é uma grande diferença em relação ao desenvolvimento nativo. Desenvolvimento nativo é a criação de software visando sua execução em uma plataforma específica. Com isso deve-se seguir definições técnicas da plataforma, como linguagem e arquitetura de classes, o que restringe as possibilidades de distribuição de componentes. Nessa modalidade de desenvolvimento, é comum que se utilize padrões de camadas como o MVC, *Model-View-Controller,* para separar as responsabilidades entre os componentes de software. Na Web, os componentes de software não são apenas categorizados, mas são projetos diferentes, fisicamente separados, por vezes até em organizações em outros continentes. Além da escalabilidade já comentada e de

simplificar as fases de projeto, por reduzir o escopo da aplicação, essa arquitetura possui outras vantagens em flexibilidade, segurança e custos (STACKIFY, 2017).

2.2 Plataforma

Em termos de plataforma alvo, uma aplicação nativa é desenvolvida para ser executada em um único sistema operacional e utiliza *kits* de ferramentas de desenvolvimento, chamados SDKs. Na Web, tais kits não existem. Desenvolver sem o auxílio de *frameworks* seria como desenvolver uma interface gráfica em Java sem utilizar uma ferramenta como a *Swing*, que abstrai a criação de componentes gráficos. Software nativo também possui uma arquitetura mais monolítica, com mais possibilidade de maior acoplamento entre seus componentes que uma aplicação Web, devido à sua infraestrutura não distribuída. Na Web, as diferenças de sistemas operacionais são abstraídas pelo navegador, permitindo que um único código seja executado em qualquer dispositivo contendo um navegador Web. Não serão listados os méritos e deméritos de cada abordagem (CLEARBRIDGE MOBILE, 2018), mas vale salientar alguns pontos do desenvolvimento Web que devem ser levados em consideração:

1. A aplicação deve ser capaz de trabalhar com absolutamente qualquer formato de tela (ANDREW, 2016).
2. Devido à separação cliente-servidor, o fluxo de dados deve ser cuidadosamente planejado, para garantir que dados inseridos não apenas estejam na memória volátil do navegador, mas sejam constantemente enviados para o servidor.
3. Recursos remotos, como arquivos hospedados por terceiros, são constantemente utilizados. Deve-se levar em consideração que estes recursos podem eventualmente não estar acessíveis.
4. Pelo fato de haver um latência na internet, o tempo entre a requisição de um recurso e a sua resposta, além de erros de comunicação devem sempre ser levados em consideração (TECHTARGET, 2016).

5. Javascript é orientada a eventos e não trabalha diretamente com threads. Todo o paralelismo se dá com o registro de funções para serem executadas após determinados eventos (NETO, 2018).

2.3 Modularidade

Na medida que um software cresce, torna-se mais difícil isolar problemas, ou modificar partes sem que seja preciso modificar várias outras, em efeito dominó. Por mais que se planeje e arquitete bem um programa, um software grande contém muitos arquivos e muitas linhas de código, o que reduz a eficiência com que o programador pode trabalhar. Além disso é bastante comum se repetir partes de código em outros projetos, como elementos de interface gráfica, ou funcionalidades de validação de formulários (JACKSON, [2018]).

Módulos são pequenas partes de código altamente coesas e desacopladas do exterior, as quais uma aplicação pode ter como dependência para compor partes maiores de si. Devido ao desacoplamento em relação aos requisitos funcionais do projeto de software final, um módulo por si só pode ser considerado um projeto separado, que com uma interface e funcionalidade bem definidos pode ser amplamente reutilizado em diversos projetos. Esta é uma prática altamente comum entre desenvolvedores Web: não apenas criar para si, mas também publicá-los na Web com licença aberta (MODULECOUNTS, 2018). Para utilizá-los, estão disponíveis vários repositórios públicos e gerenciadores de pacotes, que disponibilizam e instalam facilmente módulos Javascript, CSS, ou qualquer outro componente reutilizável de um projeto de software (DIERX, 2018).

2.4 Princípios

Em termos simples, UX - *User Experience* é o estudo do impacto emotivo de um produto sobre o seu usuário. Fruto do UCD - User Centered Design, ele leva em consideração aspectos estéticos e práticos, como performance, ergonomia, acessibilidade e usabilidade (Figura 2.2), focado nas necessidades do usuário, vide

CEROS (2016, 2018). Um designer de UX deve conseguir planejar e executar a Arquitetura de Informação, Design de Interação, Design Visual, Usabilidade e Prototipagem (THINKWIK, 2010; CEROS, 2016).

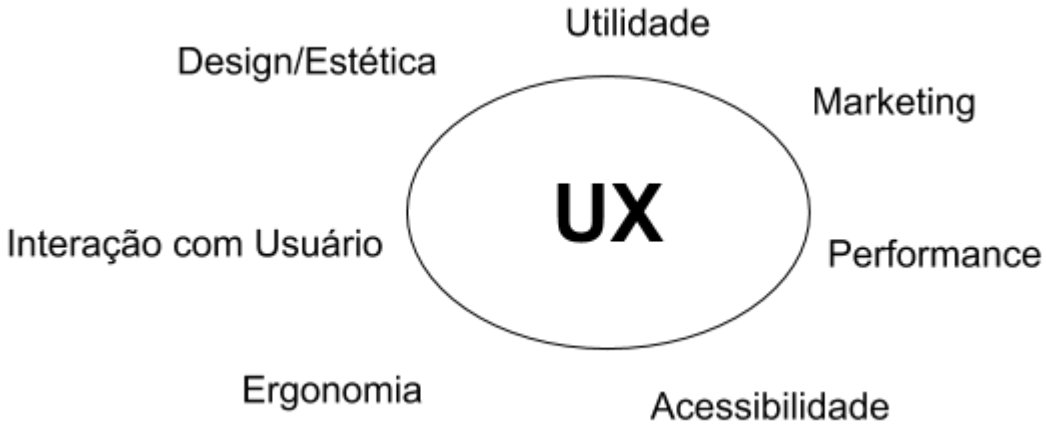


Fig. 2.2 - Aspectos do UX

Sendo a satisfação do usuário o foco deste ramo do design, sua prática afeta diretamente o nível de sucesso de um produto. Com a alta competitividade no mundo globalizado, um produto com boa usabilidade se sobressai e, caso também satisfaça as necessidades funcionais dos usuários, terá altos índices de satisfação. Baseado nas práticas do design de UX e das características técnicas da Web, é possível extrair princípios básicos para o desenvolvimento neste universo.

Um estudo realizado por Ping Zhang, da universidade de Syracuse, Estados Unidos, identificou quantitativamente as principais expectativas de um usuário ao navegar no ambiente Web. O trabalho focou no estudo sobre websites simples, porém extrai características mais gerais para a Web como um todo. Do total de 42 características listadas, nem todas tratam de características do funcionamento do produto. Algumas se originam de necessidades do contexto social, como preconceito de raça e gênero (ZHANG, 2002). Dessa forma, foi possível selecionar e mesclar algumas, baseado na relevância. Ao final, o resultado foi um conjunto de dezesseis características categorizadas. São elas:

1. Informação
    a. Nível de detalhamento apropriado
    b. Informação atualizada
    c. Informação relevante
2. Privacidade
    a. Burocracia no acesso (pagar para usar, cadastro complexo...)
    b. Garantia de segurança da informação entregue
3. Empoderamento
    a. Controle sobre a complexidade da informação visível
    b. Controle de navegação simples
    c. Integração com outros serviços
4. Facilidades
    a. Divertimento
    b. Multimídia
    c. Indicação de localização (Rastros, ou *Breadcrumbs*)
    d. Indicação de estado de espera (carregando)
5. Visual
    a. Atratividade visual
    b. Espaçamento suficiente
6. Suporte
    a. Suporte para plataformas diversas
    b. Estabilidade (conteúdo sempre acessível)

Uma grande diferença entre a plataforma Web e as plataformas nativas é o tempo de carregamento. No geral, Aplicações Web devem ser carregadas novamente para a memória temporária do cliente toda vez que forem utilizadas. Em contraste, aplicações nativas são explicitamente carregadas e instaladas no dispositivo uma única vez. O tempo de carregar todos os recursos até a aplicação estar pronta para uso é de suma importância na satisfação do usuário (ANDERSON, 2018). Este é um requisito

funcional para qualquer aplicação, ou website, na Web. Sem resposta rápida, a tendência é a perda total de usuários (WAGNER, 2018).

3 Manifesto Ágil e Extreme Programming

Em 2001, um grupo de dezessete desenvolvedores de software, com representantes de diferentes metodologias de desenvolvimento, se reuniu em uma estação de ski norte americana visando discutir dificuldades observadas. Para eles, as metodologias existentes possuíam uma abordagem muito burocrática e havia demanda por uma alternativa mais *leve*. Após dois dias de discussões e debates, o grupo, chamado A Aliança Ágil (*The Agile Alliance*), produziu um documento conhecido popularmente como Manifesto Ágil.

Este manifesto trás um conjunto de valores e princípios como sugestões para o desenvolvimento de software eficaz. Seus valores contradizem os processos produtivos mais comuns em outras áreas, mas se adequam bem à forma com que desenvolvedores se mostram mais eficientes (SERRADOR, 2015). Mais próximo de uma filosofia que de uma metodologia, as ideias do Manifesto Ágil permeiam o princípio de que num projeto pessoas são mais importantes que ferramentas. Seus valores são:

1. Comunicação entre indivíduos é mais importante que processos e ferramentas
2. Produto funcional é mais importante que documentação abrangente
3. Colaboração é mais importante que contratos
4. Atender à mudanças é mais importante que seguir um plano

Valores são muito abstratos e não podem ser diretamente aplicados à prática. Para tornar o Manifesto mais efetivo, foram criados sugestões de princípios já aplicáveis em um projeto de desenvolvimento, porém sem definir nenhuma prática de processo.

1. Garantir a satisfação do consumidor entregando rapidamente e continuamente software funcionais;
2. Até mesmo mudanças tardias de escopo no projeto são bem-vindas para garantir a vantagem competitiva do cliente;

3. Software funcionais são entregues frequentemente (semanas, ao invés de meses);
4. Cooperação diária entre *stakeholders* e desenvolvedores. *Stakeholders* são pessoas interessadas no produto. Neste caso são a equipe de clientes;
5. Projetos surgem através de indivíduos motivados, entre os quais existe relação de confiança.
6. A maneira mais eficiente e efetiva de transmitir informações é conversar cara a cara;
7. Software funcionais são a principal medida de progresso do projeto;
8. Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes para manter um ritmo constante indefinidamente;
9. Simplicidade é essencial;
10. As melhores arquiteturas, requisitos e projetos emergem de equipes auto-organizadas;
11. Em intervalos regulares, a equipe reflete sobre como tornar-se mais eficaz e então sintoniza e ajusta seu comportamento apropriadamente.

3.1 Extreme Programming

É uma metodologia de desenvolvimento muito bem sucedido por ajudar a criação de sistemas com qualidade em menor custo. A XP traz consigo cinco valores para o Desenvolvimento:

1. Comunicação
2. Coragem
3. Feedback
4. Respeito
5. Simplicidade

Comunicação

A comunicação entre cliente e desenvolvedores é primordial para haver entendimento na solução de um problema. Essa comunicação deve ser frequente e o mais próxima possível. A comunicação escrita é um meio de mão única. Conversas possuem feedback imediato, impossível por meio de documentos.

Coragem

Para que o produto entregue ao cliente não seja limitado por temores, deve-se ter a segurança em assumir que os riscos existem e saber lidar com suas consequências com praticidade. Deve-se abraçar a possibilidade de haver mudanças nos requisitos, no lugar de burocraticamente congelar o projeto.

Feedback

O relacionamento com o cliente deve sempre ser atualizado com pequenas entregas, consolidando confiança.

Respeito

Deve existir em qualquer relacionamento humano saudável, mas em especial quando não há restrições e regras que garantam o funcionamento para além das condições do relacionamento.

Simplicidade

Deve-se investir apenas no que é estritamente e definitivamente necessário, sem presunções.

Dos cinco valores desenvolvem-se algumas práticas, das quais para o propósito deste trabalho vale salientar as seguintes:

Ambiente Informativo

O ambiente de trabalho de uma equipe XP deve refletir o projeto, de forma que poucos segundos na sala de produção bastem para se ter uma noção do que é Proposto.

Build de Dez Minutos

Tempos de build e de execução de testes devem ocorrer em um tempo mínimo, de forma que não custe tempo e esforço.

Ciclo Semanal

O software é desenvolvido de modo iterativo e incremental, com período de uma semana. Dessa forma, semanalmente desenvolvedores e stakeholder se encontrarão numa reunião, chamada Jogo do Planejamento, para priorizar um conjunto de funcionalidades para ser implementadas até o próximo ciclo, também chamado de *sprint*.

Desenvolvimento Orientado a Testes (TDD)

Comumente, o desenvolvimento de software descuidado pode ser considerado "Orientado a Falhas", tamanho é o tempo gasto observando erros e corrigindo-os. Desenvolvimento Orientado a Testes prioriza a confirmação de corretude de componentes de software que implementam uma funcionalidade antes de partir para outra funcionalidade. Em TDD os testes são desenhados baseados nas definições dos Casos de Uso antes da implementação das soluções.

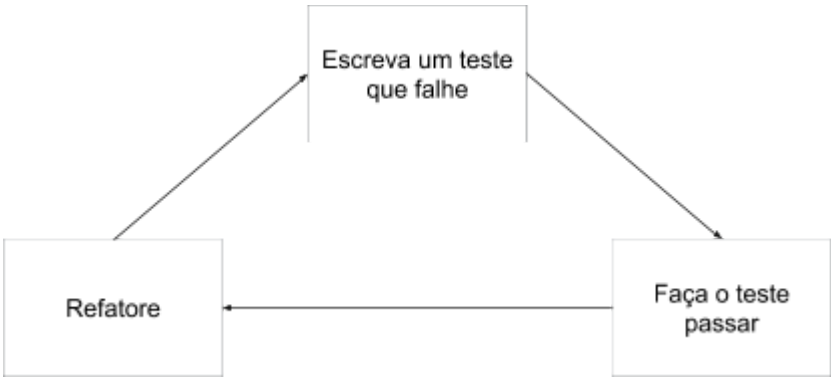


Figura 3.1.1 - Ciclo de TDD

1. Escreva um teste que falhe: Analise um dos requisitos do sistema, pense nas verificações que o teste deve fazer para que sua funcionalidade seja validada. Escreva um teste que falhe, pois ainda não há nada implementado.
2. Programe o mínimo: Codifique o mínimo possível para que ele passe pelo teste mesmo que seu método retorne apenas o valor esperado pelo teste. O objetivo aqui é passar o teste. Seja Simples.
3. Refatore o seu código de teste: Deve-se refatorar o código de teste desenvolvendo novos testes de modo que cubra mais possibilidades de erros. O Código deve continuar atendendo ao requisito e passando pelos testes.

Design Incremental

O objetivo é sempre criar a solução mais simples possível que satisfaça às funcionalidades priorizadas em cada iteração. Características para apoiar funcionalidades futuras apenas serão implementadas no seu devido momento.

Equipe Integral

A equipe de desenvolvimento deve contar com o máximo de disponibilidade dos Stakeholders.

Histórias de Usuário

São uma forma de representar em alto nível, na linguagem do cliente, as funcionalidades debatidas em uma reunião. Os desenvolvedores devem desvendar detalhes de cada história durante o Jogo do Planejamento, além de estimar o tempo baseado em experiências anteriores. Com essa estimativa, o Stakeholder pode priorizar as histórias.

Integração Contínua

Utilizando sistemas de versionamento, onde se armazenam versões de código fonte, cada desenvolvedor se responsabiliza a constantemente enviar suas alterações para o repositório central.

Trabalho Energizado

É extremamente importante que os desenvolvedores tenham momentos de folga, sem fazer muitas horas extras, de forma que estejam sempre aptos para programar com o mais alto nível de qualidade.

Contratos de Escopo Negociável

Contratos tradicionais definem previamente os seguintes valores:

1. Custo
2. Prazo
3. Escopo
4. Qualidade

Onde supõe-se tanto que o cliente é capaz de prever todas as funcionalidades que o produto de software deverá ter, quanto que os desenvolvedores serão capazes de entender e prever plenamente esses requisitos. Essa suposição é equivocada, já que mesmo que o cliente saiba corretamente exatamente o que quer, ele não será capaz de explicitá-lo por completo em nível técnico. Baseado nisso, o XP recomenda o

uso de contratos de escopo flexível, com tolerância a mudanças de escopo, onde o produto é incrementado a cada ciclicamente, onde em cada ciclo se verifica demandas por alterações.

Nas primeiras reuniões, antes do início do desenvolvimento, há um entendimento geral do problema, para poder estimar prazo, custo e nível de qualidade razoáveis para o projeto. O produto é implementado a cada ciclo e os pagamentos podem também acompanhar cada entrega. O cliente pode sempre cancelar o contrato ao final de cada ciclo sem custos adicionais.

Código e Testes

Código e testes devem ser os únicos artefatos permanentes. Artefatos são produtos resultantes de alguma prática, como arquivos, documentos e registros. Clientes pagam pelo que o sistema faz ou fará com qualidade. Quaisquer artefatos que contribuam para essas duas fontes de valor são também valiosos. O resto é desperdício.

Versionamento

O controle de versionamento de documentos é uma prática muito comum em editores de texto e aplicações de armazenamento de arquivos. O sistema de versionamento (*VCS*) mantem uma linha do tempo (histórico) com registro de características, conteúdo e alterações em um arquivo. No desenvolvimento de software, *VCSs* possuem funções bastante significativas. Com uma grande margem, o *VCS* mais utilizado é chamado Git e dentre as suas funções mais utilizadas estão:

1. Ramificação da linha do tempo, para isolar novas alterações, que poderiam pôr em risco a linha do tempo principal, entrando em conflito com outras alterações acontecendo em paralelo
2. Mesclagem de históricos, de forma a unificar alterações feitas em outras linhas do tempo

3. Comparação de alterações, para identificar onde um desenvolvedor alterou cada arquivo
4. Recuperação de estados anteriores, que permite desfazer alterações para qualquer momento no passado

Cada registro, ou marco, no histórico do código de um projeto é chamado *commit* e deve ser feito manualmente pelo desenvolvedor. Cada ambiente de desenvolvimento possui uma cópia de todos os históricos de um projeto, o repositório local. As alterações feitas em um repositório local devem ser então enviadas para uma central, o repositório remoto, através dos chamados *Pull Requests*.

Apesar de ser utilizado por toda a indústria de desenvolvimento e de não ser uma prática definida por métodos Ágeis, um sistema de versionamento é imprescindível no desenvolvimento Ágil por permitir uma forma estruturada de executar a prática *Integração Contínua* e alterações de última hora, sem quebrar todo o código.

4 Investigação-Ação

Os capítulos anteriores apresentaram os princípios mais importantes e relevantes para o desenvolvimento de aplicações Web seguindo uma metodologia ágil, focando tanto nos aspectos técnicos como nas expectativas do cliente, o que impacta diretamente na formulação de um processo de desenvolvimento. Este capítulo apresenta a descrição do método de Investigação-Ação e das fases do ciclo de melhoria de processos *PDCA*.

Em tradução livre, "A Investigação-Ação é, antes de mais, uma metodologia que procura superar o habitual dualismo entre teoria e prática" (Noffke e Someck, 2009). A Investigação-Ação é um conjunto de métodos cíclicos e sistemáticos que buscam o refino de informações quanto a algum objeto de estudo através da sua aplicação prática. Tradicionalmente, seus métodos seguem uma epistemologia mais interpretativa, voltada para ciências sociais (BALDISSERA, 2001). Em paralelo, existem métodos mais positivistas, focados em ciências exatas, que seguem uma filosofia de melhoria progressiva muito similar a da Investigação-Ação, como o *PDCA*. Neste trabalho de Investigação-Ação utilizaremos o método *PDCA* para encontrar um processo de desenvolvimento Web adequado a um indivíduo, a partir de práticas e recomendações do método de desenvolvimento XP tradicional.

4.1 *PDCA*

Planejar, Executar, Checar e Agir (do inglês: *Plan, Do, Check e Act)* são as etapas do PDCA. É um método de melhoria de processos recomendado pela ISO 9001:2015 que segue a filosofia da Investigação-Ação, de refino através da prática (HAHN, 2012). Suas quatro fases são definidas como:

Planejar

Desenvolver um plano de avaliação da qualidade do processo. Identificar critérios e objetivos.

Executar

Executar processo a ser avaliado, extraindo artefatos relevantes para posterior avaliação.

Avaliar

Avaliar resultados, comparando-os com objetivos definidos anteriormente. Deve-se identificar as causas de qualquer problema encontrado.

Agir

A partir do que foi descoberto na etapa anterior, modificar processo para evitar a recorrência do problema, visando resultar num processo de mais qualidade.

5 Adequação do Processo

Visando entregar um produto como recomendam as metodologias Ágeis, o desenvolvimento deve ser incremental. As funcionalidades deverão ser completadas dentro de ciclos curtos, chamados *sprints* e entregue funcionando, sem quebrar entregas anteriores. Em cada *sprint* executa-se um processo de desenvolvimento com fases bem definidas. Essa ciclicidade permite uma correspondência entre ciclos do *PDCA* e os ciclos de desenvolvimento, de forma a fundir as etapas de ambos e garantindo um ciclo uniforme de desenvolvimento e refino do processo de desenvolvimento simultaneamente.

A partir de um processo XP comum, o processo de desenvolvimento individual será obtido pela aplicação do *PDCA* para identificação e aplicação de adequações. Para tal, é necessário se estabelecer um processo inicial baseado em práticas definidas no XP, além de critérios de adequação para cada prática, que indiquem as qualidades necessárias para sua utilização por um indivíduo. Tal processo será utilizado na fase *executar* do *PDCA*, que será repetida a cada *sprint*, junto com uma reavaliação do processo, na fase *avaliar*, escolha de mudanças e adequações necessárias no processo, na fase *agir* e possivelmente redefinição dos critérios de adequação, na fase *planejar*. Os critérios de adequação serão inicialmente definidos junto às práticas e bem como elas próprias, poderão ser modificados. A fase *planejar* trará apenas as mudanças nos critérios, caso existam. Caso não haja novos critérios a ser definidos anteriormente ao início de um ciclo, esta fase será omitida.

Para melhor organizar a sua descrição, o processo de desenvolvimento será particionado em três etapas: Análise, Desenvolvimento e Entrega. Na fase de Análise acontece a reunião com o *PO*, onde há o entendimento dos requisitos para a *sprint*, seleção de funcionalidades e projeto de implementação, com a modelagem de componentes do sistema a serem criados. Na fase de Desenvolvimento há a implementação dos componentes modelados na fase anterior. Na fase de Entrega há a publicação dos componentes implementados em uma plataforma disponível para acesso dos clientes, de modo que eles acompanhem de perto o desenvolvimento da

solução sendo contratado. A Figura 5.1 mostra como o método *PDCA* será utilizado para melhorar o processo de desenvolvimento.

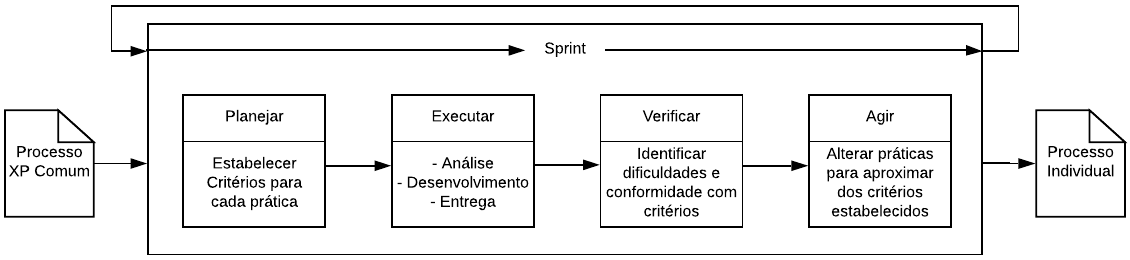


Figura 5.1 - Fases do *Sprint* com *PDCA*

5.1 Processo Inicial

Baseado em critérios de individualidade e simplicidade, foram selecionadas práticas utilizadas no XP para serem utilizadas como um processo inicial. Na busca pela melhor adequação, foram também definidos critérios de adequação de tais práticas para a utilização de um indivíduo. Nas Tabelas 5.1, 5.2 e 5.3 temos as práticas XP e critérios de validação para verificação de conformidade em cada uma das 3 etapas de desenvolvimento.

Análise

Prática	Critérios
Casos de Uso (UC)	Funcionalidades agrupadas em categorias independentes. Deve ser rapidamente construído e validado pelo *PO* durante a reunião do *sprint*.
Estórias de Usuário (US)	*PO* deve ser capaz de imediatamente confirmar os objetivos de cada *funcionalidade* encontrada apenas pela descrição no formato de *US*, descritos sob perspectiva do usuário
Wireframes	Deve ser possível validar e identificar componentes chave da interface gráfica de modo a permitir uma boa estimativa de tempo de criação da página
Design Simples/Incremental	Funcionalidades descritas devem ser pequenas e isoladas, de forma a entregar valor imediatamente

Jogo do Planejamento/Ciclo Semanal	Funcionalidades selecionadas devem ser pequenas o suficiente para ser possível concluir todas *sprints* em ciclos de mesma duração
Trabalho Energizado	É possível trabalhar com limite baixo de horas diárias trabalhadas
Envolvimento do Cliente	É possível trabalhar diariamente com a presença do *PO*
Contrato de Escopo Negociável	*PO* aceita e se demonstra satisfeito com contrato flexível
Código e Testes	Apenas arquivos relativos ao código fonte devem ser mantidos, incluindo arquivos de teste e o mínimo de documentação
Metáfora	Deve auxiliar comunicação entre *PO* e desenvolvedor

Tabela 5.1 - Práticas e Critérios da fase de Análise

Desenvolvimento

Prática	Critérios
Base de Código Unificada	Ramos de versionamento não devem durar mais que algumas horas
Integração Contínua	Versionamento deve ser constantemente atualizado com código novo, a cada pequena mudança, criando marcos na sua linha do tempo. Grandes mudanças a em um único marco devem ser evitadas.
Build de dez minutos	Código alterado deve ser rapidamente transformado em código executável
Desenvolvimento Orientado a Testes	A criação de testes não deve atrasar entrega de funcionalidades
Refatoração	Não deve ser feito em código ainda não funcional e não pode quebrar um código pronto. Deve entregar valor ao cliente e não apenas ao desenvolvedor.

Tabela 5.2 - Práticas e Critérios da fase de Desenvolvimento

Entrega

Prática	Critérios

Implantação Diária	Em hipótese alguma deve quebrar sistema em produção, ou entregar funcionalidades incompletas, de forma que não façam sentido para o usuário.
Entrega Rápida	Sincronização entre hospedagem e repositório deve ser automática ou com poucos comandos

Tabela 5.3 - Práticas e Critérios da fase de Entrega

5.2 Ciclos de Melhoria de Processo

Esta seção demonstra a execução de um processo de melhoria contínua baseada nas fases do *PDCA*, mesclando suas etapas com as etapas de desenvolvimento. Essa mesclagem é feita através da substituição da fase *Executar* do *PDCA* pelas fases do processo de desenvolvimento *Análise*, *Desenvolvimento* e *Entrega*. Nestas fases, a execução das práticas do processo poderá gerar artefatos, em maior parte diagramas e imagens, cujo conteúdo é transcrito, ou fotografado, e faz parte deste trabalho. Cada prática também trará comentários quanto à experiência que trouxe a sua execução.

Ao invés de utilizar verbos genéricos, como no *PDCA*, as fases serão identificadas de forma mais descritiva, com substantivos que representam o seu objetivo neste trabalho. A fase *Planejar* será chamada *Critérios*, onde poderão ser estabelecidos critérios de adequação mais relevantes para a *sprint* seguinte. Tais critérios deverão definir as expectativas em relação à prática, de forma que ao final da *sprint*, na fase *Verificação*, seja possível avaliar sua adequação com o desenvolvimento individual. Talvez não seja necessário redefinir critérios em todas as *sprints* e esta fase poderá ser omitida em alguns dos ciclos. *Executar* será substituída por três fases, *Análise*, *Desenvolvimento* e *Entrega*. Em *Análise* ocorrerá o planejamento da *sprint* seguinte, seleção de funcionalidades e projeto de execução. Em *Desenvolvimento* haverá a produção de código e testes. *Entrega* trará o envio do código desenvolvido ao ambiente de produção, pronto para uso. A fase *Verificar* será chamada *Verificação*, onde será desenvolvida a análise da adequação das práticas a partir dos critérios definidos e das dificuldades em executá-las durante a *sprint*. *Agir* será *Adequação*, onde as práticas devem ser readequadas, adicionadas, ou removidas. As *sprints* são

numeradas e rotuladas pela funcionalidade que entregam. Ao início de cada *sprint* pode haver um comentário sobre o que se espera do ciclo que se iniciará.

5.3 Problema Proposto

Um problema recorrente na terceira idade é o mal uso do tempo. Muitos idosos não estimulam seus cérebros com atividades adequadas, por vezes passando muito tempo dormindo, ou assistindo televisão passivamente. Uma atividade passiva é aquela que não estimula o indivíduo a interagir, pensar ou questionar. Muitos cuidadores, especialmente de idosos com algum nível de demência, estimulam seus pacientes a serem passivos, de forma a evitar aumento de trabalho, causando agravamento severo na doença (HUNT e MACKAY, 2014).

Neste contexto, uma empresa de terapia ocupacional, em Recife, chamada Reato - Reabilitação em Terapia Ocupacional, precisa de uma solução para o problema de rastrear, exibir e identificar as atividades problemáticas na rotina dos pacientes. Atividades problemáticas são atividades com alto nível de passividade ou dependência. Dependência é a incapacidade de completar uma atividade sem auxílio de terceiros. Além disso, os cuidadores, que na maioria dos casos são os fornecedores do relatório de atividades, muitas vezes possuem baixa escolaridade e pouco interesse em preencher corretamente as informações necessárias. A solução buscada deve também solucionar o problema de preenchimento do relatório.

5.4 Sprint 0 - Configuração

Esta *sprint* é diferente das demais, pois no lugar de escolher funcionalidades e implementá-las, nela se analisa o problema, visando identificar *atores*, encontrar *Épicos* e requisitos funcionais e não funcionais, além de decidir que tecnologias são adequadas para solucionar o problema. Com essas informações, formula-se um acordo flexível, com estimativas de custos baseadas no conhecimento superficial do problema e das tecnologias. Também prepara-se o ambiente de desenvolvimento para os ciclos seguintes, que entregarão as funcionalidades desejadas. Nesta *sprint* será possível determinar qual a duração adequada para uma única *sprint* no projeto.

5.4.1 Critérios

Os critérios de adequação para este *sprint* são um pouco específicos, por se tratar de uma fase inicial de entendimento do problema. A Tabela 5.4 trás os critérios para as práticas *Descrição* de *Casos de Uso* e *Descrição* de *Estórias de Usuário* nesta *sprint*.

Prática	**Artefatos**	**Critérios**
Descrição de Casos de Uso (UC)	Diagrama de UC	Funcionalidades devem ser agrupadas por Épicos Independentes e diagrama deve ser rapidamente construído durante a reunião do *sprint*.
Descrição de Estórias de Usuário (US)	Cartões de US	*PO* deve ser capaz de compreender os objetivos de cada US apenas olhando seus cartões.

Tabela 5.4 - Adaptação de critérios de adequação para esta *Sprint*

5.5.2 Análise

O escopo de negócio do problema proposto ainda precisava ser conhecido. Faltava identificar a situação na qual o problema é percebido. Além disso, faltava também determinar quais obstáculos o problema cria e estabelecer um vocabulário

comum a ambos os lados, desenvolvedor e *stakeholders*. Ainda antes de responder a estas questões, a primeira definição a ser feita foi qual das duas sócias seria responsável pela comunicação entre a empresa e o desenvolvedor, passando especificações e decisões quanto à solução. No Scrum, esse *stakeholder* é chamado *product owner,* ou *PO*.

Uma conversa de cerca de duas horas se desenrolou com a *PO*, no seu ambiente de trabalho, com o objetivo de identificar respostas para as perguntas mencionadas, referentes ao escopo do problema. Da conversa foi extraído o conteúdo para gerar o primeiro artefato "Escopo e Propósito", que é parte do *backlog* do projeto. O *backlog* do projeto é o conjunto de artefatos referentes ao projeto inteiro, enquanto *backlog* de *sprint* é o conjunto de artefatos referentes a uma *sprint.*

Com o problema já compreendido, iniciou-se uma fase de entendimento de funcionalidades. Tradicionalmente nas metodologias Ágeis, Épicos são Estórias de Usuário de alto nível. Aqui foi usada a liberdade de utilizar o termo para agrupar Casos de Uso por temática. Os Casos de Uso como estritamente definidos pela *UML* demonstraram não ser a forma mais adequada para se agrupar e compreender o fluxo e a dependência entre recursos, devido à ausência de sinalização de eventos e prioridades. Porém seu diagrama pode ser rapidamente construído durante a conversa, sem interrompê-la, além de ser visualmente simples de se trabalhar com a *PO*. Desta forma, um mapa mental baseado no Diagrama de Casos de Uso foi construído. A lógica por trás do mapa mental era conectar o símbolo do usuário a ações que poderiam ser imediatamente implementadas e gradualmente conectar a elas novas ações que delas dependessem. Foi possível, então, identificar quatro categorias de funcionalidades, que são os *Épicos*, e Casos de Uso de alto nível, que serão selecionados para implementação nas *sprints*. O "Diagrama de *Épicos*" foi criado manualmente durante a reunião e demonstra algumas funcionalidades em alto nível, bem como suas dependências com outras funcionalidades e também faz parte do *backlog* do projeto, representado na Figura 5.1.

Após terminado o artefato "Diagrama de Épicos", ficou bastante claro para o desenvolvedor que a solução deve oferecer grande segurança aos dados, de forma que informações particulares não estejam disponíveis publicamente. Dessa forma, o Usuário deve ser autenticado, antes de poder visualizar qualquer informação restrita. Em consequência, foi criado um novo *Épico* com novos UCs relativos a autenticação, representados diagrama na Figura 5.2.

Com o intuito de formalizar e confirmar a adequação dos UCs junto à *PO*, eles foram convertidos para o formato de cartões de *Estórias de Usuário (US)*, que no lugar de utilizarem um paradigma visual, utilizam um paradigma verbal, convertendo o diagrama em frases mais facilmente compreendidas do ponto de vista do usuário. A construção de *Estórias* se mostrou um pouco lenta, se comparada com o "Diagrama de Épicos", porém foram bem sucedidas. Com as *Estórias* aprovadas imediatamente, sem nenhum conflito de ideias, foi possível identificar recursos, ações e requisitos de qualidade relativas aos *Épicos*, que na Lista 5.1 estão destacados em itálico, sublinhado e negrito, respectivamente. Detalhes sobre a análise de requisitos na seção Funcionalidades. Com essas informações ficou claro que a melhor solução seria na plataforma WEB e também identificar quais tecnologias eram necessárias para implementá-la. Maiores informações sobre a escolha das tecnologias na seção Tecnologias.

O próximo passo foi construir *wireframes* manuais de como as páginas se pareceriam. *Wireframes* são simplificações da interface gráfica, contendo apenas caixas e nenhuma cor ou especificação de fonte. Como haveria muitas páginas similares, foram criados apenas três *wireframes*, um para as páginas de listagem de recursos, como listagem de Pacientes ou de Rotinas, outro para formulários de criação ou edição de recursos e outro para a visualização das Atividades de Rotinas, que é um dos principais objetivos da aplicação. Esses *wireframes* fazem parte do *backlog* do projeto.

Conhecendo a velocidade com que trabalha quando utiliza as tecnologias escolhidas, foi estimado quanto tempo levaria para criar operações CRUD simples no *backend* para cada recurso identificado nas *Estórias*. Todo recurso deveria também ter

uma comunicação entre o *backend* e o *frontend*, desenvolvida em Javascript. As páginas do *frontend* foram um pouco mais complicadas de estimar, já que não havia nenhum projeto de design de baixo nível, que entregasse maiores detalhes da implementação. Baseado nos *wireframes* e em pequenas suposições quanto aos componentes da interface gráfica, foi possível superestimar genericamente o tempo de desenvolvimento para cada página. Tal processo está descrito na seção Estimativas.

Somando todas as estimativas, a criação de cada recurso completo levaria uma semana, com seis a oito horas diárias, onde haveria a criação de uma tabela no Banco de Dados, cinco operações na API de *backend*, a comunicação entre *frontend* e *backend* e duas páginas, para listagem e edição/criação de recursos. As cinco operações na API de *backend* sobre um recurso seriam quatro *CRUD* (Criação, Leitura, Atualização e Remoção) e uma de listagem de todos os recursos daquele tipo. Esta estimativa seria a referência para todas as outras. Ficou acertado com a *PO* que as funcionalidades seriam definidas semanalmente, com a possibilidade de se redefinir prioridades ou adicionar mudanças, contanto que estivessem no escopo dos *Épicos* já definidos, como recomenda a prática *Contrato de Escopo Flexível*.

Funcionalidades

Os artefatos para a *sprint* 0 tratam especialmente do escopo do projeto como um todo e por esta razão fazem parte do *Backlog* do Projeto e não do *Backlog* do *sprint* 0. O primeiro artefato, chamado "Escopo e Propósito", focou no entendimento e em definições iniciais para o problema encontrado pelas clientes. É um documento onde se descreve os atores envolvidos, a situação problemática e quaisquer conceitos relevantes para uma comunicação efetiva entre interessados.

Como já descrito anteriormente, para identificação de *Épicos* foi utilizado um mapa mental de funcionalidades, que possibilitou categorizar e identificar dependências entre diferentes funcionalidades, resultando no "Diagrama de Épicos" (Figura 5.1). Após a análise de tal diagrama, foi percebido a necessidade da adição de dois novos Casos de Uso, tal qual a Figura 5.2.

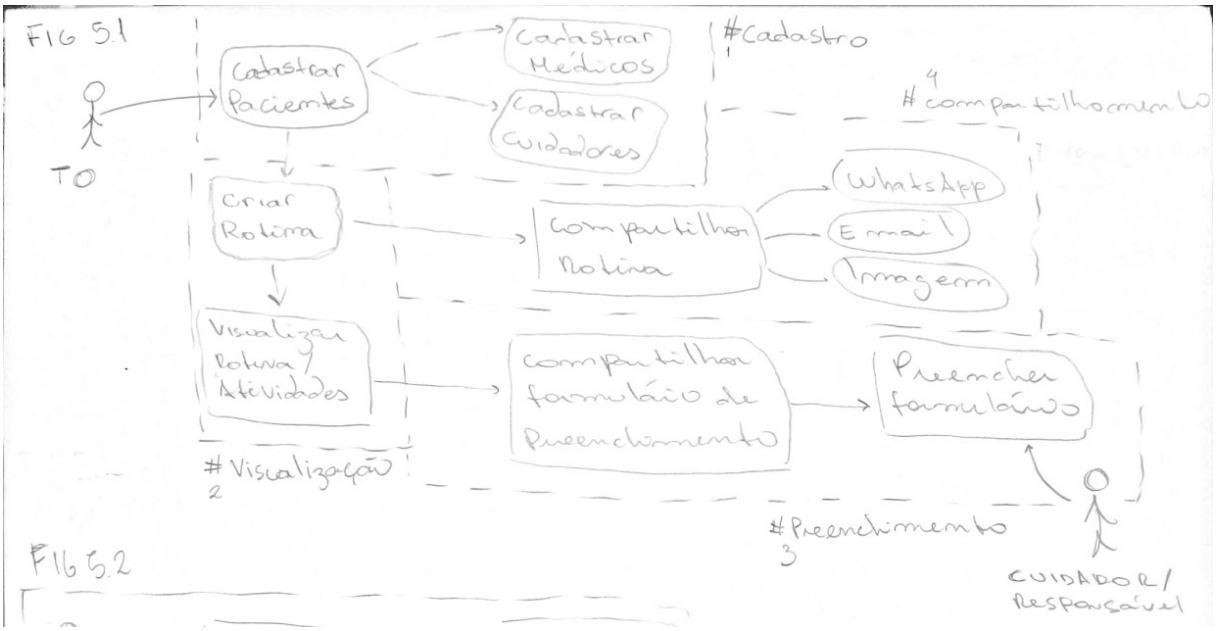


Figura 5.1 - Diagrama de Épicos

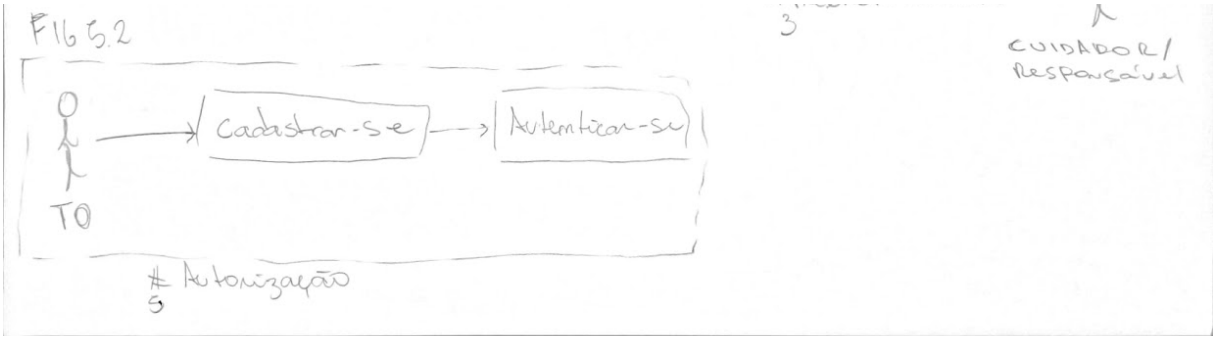


Figura 5.2 - Casos de Uso adicionais

Dos *Épicos* identificados, foi possível construir cinco *Estórias de Usuário*, para validação da *PO*. Nele estão destacados recursos, ações e requisitos de qualidade em sublinhado, itálico e negrito, respectivamente:

1. Cadastros: COMO TO, EU QUERO *cadastrar* pacientes, junto com seus médicos e cuidadores, para poder *registrar* e *compartilhar* sua rotina

2. Visualização: COMO TO, EU QUERO *visualizar* a rotina de meus pacientes de forma **simples** e **intuitiva**, PARA QUE consiga *identificar* que atividades não são **adequadas** e devem ser modificadas
3. Preenchimento: COMO TO, EU QUERO *compartilhar* com cuidadores um formulário de **fácil preenchimento**, PARA QUE cuidadores me *entreguem* os dados de rotina necessários
4. Compartilhamento: COMO TO, EU QUERO *imprimir/compartilhar* os dados de rotina, PARA QUE interessados possam *visualizar* **sem dificuldade**
5. Autorização: COMO TO, EU QUERO *garantir* a privacidade dos meus pacientes, mantendo seus dados **seguros**, PARA QUE apenas eu possa *acessá-los*

Lista 5.1 - Estórias do Usuário de Épicos

Tecnologias

Além do entendimento do problema, a *sprint* 0 trás a escolha, instalação e configuração de ferramentas e *frameworks* no ambiente de desenvolvimento. Após a reunião inicial, onde foram produzidos os artefatos "Escopo e Propósito" e "Diagrama de Épicos", o desenvolvedor retornou para seu local de trabalho e identificou algumas características técnicas do problema. Dentre elas:

1. Autorização de acesso aos dados
2. Armazenamento em bancos de dados
3. Compartilhamento em diversas mídias
4. A plataforma do dispositivo do usuário é irrelevante
5. Não há necessidade de acesso a funcionalidades específicas de dispositivos físicos, como *Bluetooth*, *Sensores de Inclinação*, etc.

Lista 5.2 - Qualidades Técnicas do Problema

Os três primeiros pontos podem ser entregues tanto por aplicações nativas, quanto por aplicações Web. Os dois últimos, porém, demonstram que não há

necessidade de acesso via aplicação nativa. Dessa forma, para alcançar o maior número de usuários, em qualquer ambiente de trabalho com acesso a Internet, decidiu-se por uma solução em um sistema Web. Restava agora preparar o ambiente de desenvolvimento. Para o ambiente de desenvolvimento, se decidiu utilizar as seguintes tecnologias de desenvolvimento:

1. Laravel, para *backend* em PHP
2. MySQL, para banco de dados SQL
3. VueJS, para *frontend* em Javascript
4. GIT, para versionamento
5. Github.com, como repositório GIT remoto

Lista 5.3 - Tecnologias de Desenvolvimento

A escolha delas se deu majoritariamente pela experiência do desenvolvedor e pelo suporte *online* para o seu uso. Com o *framework* de *backend* Laravel é possível desenvolver rapidamente um sistema de autenticação e acesso aos dados via protocolo HTTP, além de possuir integração simplificada com o banco de dados MySQL. Dessa forma, o Laravel provê as principais características necessárias para o gerenciamento e acesso aos dados para a solução do problema proposto. Já no *frontend*, o *framework* VueJS é simples e rápido. Diversas ferramentas seriam viáveis para o *frontend*, então a escolha se deu em especial pela experiência do desenvolvedor.

Estimativas

Para melhor estimar o tempo de desenvolvimento do *frontend*, foram prototipadas páginas genéricas em alto nível (*Wireframes*) para as operações mais básicas definidas no "Diagrama de Épicos". As operações mais básicas são de listagem de recursos e criação/edição de seus dados. Para estimular a expectativa das clientes e entregar alguma noção do produto final, foi também criado um *Wireframe* para a visualização das Atividades. Estes protótipos foram desenhados digitalmente na Figura 5.3 e enviados para a *PO* via *chat*.

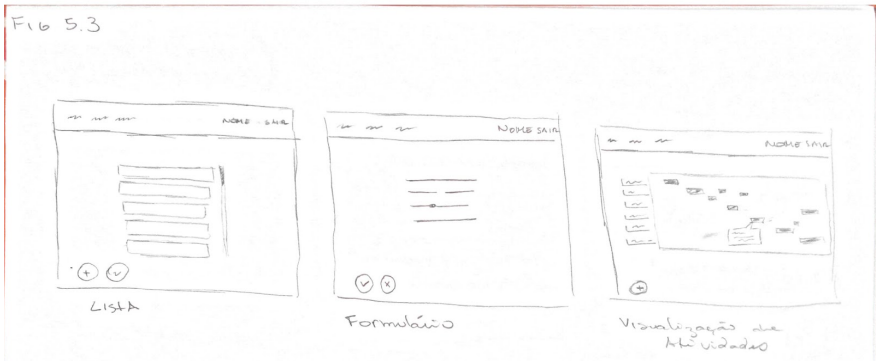


Figura 5.3 - *Wireframes* iniciais

Para estimar custos, o desenvolvedor utilizou o seu conhecimento quanto à geração de APIs Restful em Laravel. Estas APIs são interfaces do protocolo HTTP que representam operações CRUD (operações de Criação, Leitura, Atualização e Remoção de recursos, respectivamente) sobre recursos em um servidor Web, possivelmente asseguradas por autenticação. Como comentado no Capítulo 2, recursos em HTTP são a informação retornada pelo servidor a partir de uma requisição do cliente a uma URL. No problema proposto neste trabalho, os recursos foram identificados sublinhando substantivos nas Estórias do Usuário na Lista 5.1 (pacientes, rotinas, médicos, entre outros) e serão encontrados em URLs definidas no projeto da API de *backend*. Conhecendo a velocidade com que trabalha, o desenvolvedor estimou que pode criar recursos em duas horas, operações CRUD simples em até quatro horas e criar relações entre recursos em mais quatro horas. Dessa forma, poderia criar o *backend* para cada recurso em até dois dias. Baseado nos *wireframes* e em pequenas suposições quanto aos componentes da interface gráfica, foi possível superestimar que cada página de listagem ou de formulário, que são bastante simples, levaria até um dia para ser completada. A comunicação do *frontend* com o *backend*, definida em Javascript, levaria cerca de duas horas. A partir dos substantivos sublinhados nas *Estórias de Usuário*, foram extraídos inicialmente sete recursos:

1. Usuário
2. Paciente
3. Médico

4. Cuidador
5. Rotina
6. Atividade
7. Formulário

Dado que a implementação de cada um dos recursos foi superestimada para ser concluída em até uma semana, a estimativa inicial para completar os sete recursos seria de sete semanas de seis a oito horas diárias.

5.4.3 Desenvolvimento

Não é relevante entrar em detalhes da instalação de cada uma das ferramentas, mas dois projetos foram criados, um para *backend* e outro para *frontend*, chamados API e *Webclient* respectivamente. Também dois sistemas de versionamento GIT, cada um com um repositório remoto diferente. Durante a fase inicial de desenvolvimento e validação do software, não há necessidade de se contratar um servidor dedicado de grande porte. Como as tecnologias definidas na fase de Análise estão presentes na maioria dos servidores Web compartilhados, essa seria a opção com melhor custo-benefício. Mais detalhes na seção Servidor, adiante. Por hora, o próprio servidor do website da empresa Reato já seria suficiente.

Após instalados os *frameworks* em seus devidos projetos, houve um *commit* para definir um ponto de restauração inicial no sistema de versionamento. Só o ambiente de desenvolvimento não entregava valor algum às clientes. Para elas, não há valor em saber que o computador do desenvolvedor e o servidor estão configurados corretamente. Dessa forma, para entregar algum valor no *sprint* 0, foi criado um novo ramo de versionamento onde foi criada uma página de recepção com a mensagem "Em breve aqui, uma análise inteligente da rotina de seus pacientes".

Servidor

Existe um conjunto padrão de tecnologias para a WEB, que é usado nos servidores mais baratos. Este conjunto é conhecido como pilha *LAMP* de tecnologias de

serviço Web (Linux, Apache, MySQL e PHP). Esta pilha foi escolhida para a implementação deste projeto.

5.4.4 Entrega

Nesta primeira entrega havia muitos arquivos para enviar para o servidor, indo contra a prática *Entrega Rápida*. O processo de envio foi feito durante a pernoite, para evitar desperdício de tempo útil de desenvolvimento.

5.4.5 Verificação

Foi utilizada uma flexibilização do Diagrama de Casos de Uso, de modo a representar relacionamentos entre recursos e entre funcionalidades. Este uso improvisado do Diagrama de Casos de Uso é inadequado, por confundir seus objetivos com o de outros diagramas. Deve se identificar um diagrama mais adequado para o relacionamento entre recursos. Ainda assim, foi possível identificar *Épicos* com sucesso para este projeto.

Estórias de Usuário se demonstraram lentas, mas por sua vez sugerem relacionamentos entre recursos. Uma solução para o relacionamento de entre recursos pode solucionar o uso de Casos de Uso e eliminar a prática Estórias de Usuário.

Contratos flexíveis foram bem aceitos pelas clientes, em especial por terem um contato mais periódico com o que está sendo desenvolvido e poderem intervir com mais frequência. Por outro lado, um desenvolvedor individual não possui ambiente de trabalho para contar com a presença diária de um representante de seus clientes. Além disso, a própria natureza mais barata de um projeto individual indica que não haverá funcionários disponíveis para tal papel.

Ainda não foi possível avaliar a eficácia dos *Wireframes* nas estimativas e por esta razão ainda não houve nenhuma verificação para esta prática. A tabela 5.5 trás um resumo da avaliação das práticas mais relevantes utilizadas nesta *sprint*.

Prática	Está Adequado?

Descrição de Casos de Uso (UC)	Não. Na ausência de diagramas mais adequados, foi necessário uma adaptação das definições do diagrama de *UC* em mapas mentais. As dependências entre casos de uso de alto nível puderam ser extraídas de forma superficial, necessitando serem trabalhadas em nível mais baixo posteriormente. No novo formato, *Épicos* foram extraídos rapidamente e com sucesso.
Descrição de Estórias de Usuário (US)	Sim. Apesar de lentas, Estórias de Usuário auxiliaram na identificação de recursos e relações entre eles.
Envolvimento do Cliente	Não. Não será possível contar diariamente com a presença da *PO.*
Contrato de Escopo Negociável	Sim. A *PO* se demonstrou satisfeita com prazos e possibilidades de um acordo flexível.
Entrega Rápida	Entrega inicial se demonstrou demorada, pelo grande número de novos arquivos. Esse problema não deverá se repetir nas próximas entregas, onde a quantidade de novos dados será menor.

Tabela 5.5 - Avaliação das práticas executadas

5.4.6 Adequação

Um desenvolvedor individual não costuma possuir estabelecimento fixo, por não necessitar de um ambiente de trabalho para uma *equipe integral*. Também não há equipe para atribuir a responsabilidade do contato com os clientes e o desenvolvedor não dispõe de tempo para o constante deslocamento, dado que a atividade de desenvolvimento, que produz o maior valor, deve ser prioritária. Assim, foi necessário adaptar a prática *Envolvimento do Cliente*, de forma que a presença física não fosse necessária. O contato foi diário, porém por meio digital. A adequação da prática está registrada na Tabela 5.6.

Prática	Critérios
Envolvimento do Cliente	Contato diário por meio digital

Tabela 5.6 - Adequações das Práticas após *sprint* 0

5.5 Sprint 1 - Pacientes

5.5.1 Análise

O *Jogo do Planejamento* a partir desta *sprint* ocorreria através de reuniões remotas, via chat, exceto no caso de funcionalidades muito elaboradas. Do artefato "Diagrama de Épicos", no *backlog* do projeto, foi identificada a próxima funcionalidade a ser implementada, que é pré-requisito para a implementação de outras funcionalidades. Tal funcionalidade ainda estava detalhada em alto nível e não era possível extrair atividades de implementação. Foi necessário então dividir o seu Caso de Uso em Casos de Uso mais específicos, descritos na Figura 5.4, na seção Funcionalidades. Além das questões levantadas na Avaliação da *sprint* anterior, representar Casos de Uso muito específicos no formato de Estórias de Usuário é muito desinteressante, especialmente numa reunião via chat e não entrega maiores novidades ao processo de análise.

Com os Casos de Uso selecionados, foi necessário quebrá-los em atividades de desenvolvimento bastante específicas, de forma a encaixá-las em um cronograma da *sprint*. Na seção Atividades, a Tabela 5.7 mostra as atividades extraídas de cada Caso de Uso.

Funcionalidades

A lógica de negócio do cliente sugere explicitamente uma dependência entre o recurso *Paciente* e todos os outros recursos. Tal dependência foi explicitada no "Diagrama de Épicos", indicando um fluxo de valor iniciado no Cadastros de Pacientes. Desta forma, foi decidido junto à *PO*, via *chat*, que este seria o primeiro Caso de Uso a ser implementado no sistema.

Os usuários precisam **criar** novos *Pacientes*, **atualizar** e **acessar** seus dados, além de **remover** *Pacientes*. Os relacionamentos entre Pacientes e Médicos e entre Pacientes e Cuidadores são de um-para-muitos, ou seja, cada Paciente pode possuir vários Médicos e vários Cuidadores, mas cada Médico ou Cuidador pertence a apenas

um Paciente. Pelo fato de um Paciente não pertencer a nenhum outro recurso, *Cadastrar Pacientes* não depende de outros Casos de Uso, como já identificado no "Diagrama de Épicos" da Figura 5.1, sendo composto apenas pelos Casos de Uso de operações *CRUD* simples, tanto no *backend*, quanto no *frontend*. Então, apenas esses quatro casos de uso e a visualização de todos os pacientes foram selecionados para esta *sprint*. Uma prática recorrente no desenvolvimento de sistemas para melhor experiência do usuário é substituir operações de deleção por arquivamento. Desta forma, um recurso não é perdido após sua remoção, mas fica disponível em um arquivo separado, como a ferramenta Lixeira, utilizada pelos sistemas operacionais mais conhecidos. Esta prática será utilizada neste projeto para todos os Recursos que não armazenem dados altamente confidenciais, o que será definido junto à *PO*. A Figura 5.4 demonstra os *UCs* para esta *sprint*.

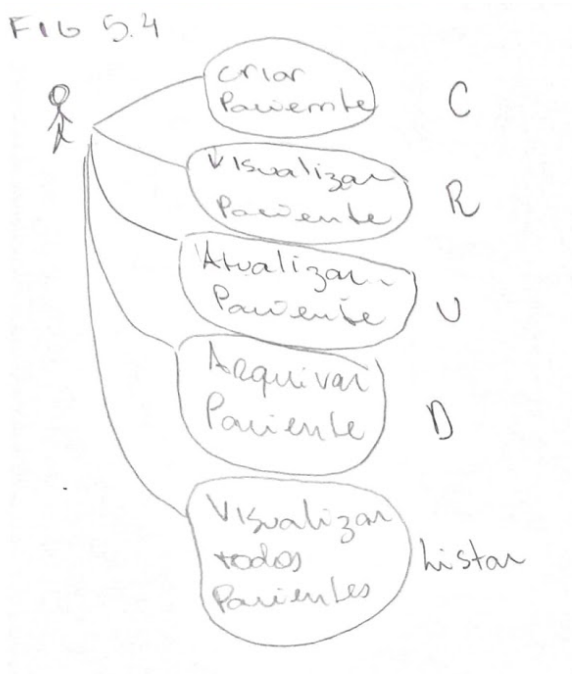


Figura 5.4 - Casos de Uso de Cadastro de Pacientes

Atividades

Os cinco Casos de Uso foram quebrados em nove atividades de desenvolvimento. Para cada operação dos Casos de Uso foi criada uma atividade no

projeto de *backend*, além de uma atividade para configuração do banco de dados para o recurso Paciente. No *frontend* foi criada uma atividade para criação da comunicação entre cliente e servidor e uma para cada página, de listagem e de edição/criação de dados de Paciente. As estimativas foram elaboradas a partir da experiência do desenvolvedor e de guias e relatos em fóruns de desenvolvedores.

#	Título	Descrição	Estimativa (hrs)
1	**API *Backend***	**CRUD *Restful* em PHP**	**7**
1.1	Criação do Banco de Dados		2
1.2	*CREATE*	Criação de Pacientes via *POST*	1
1.3	*READ all*	Visualização de todos os Pacientes	1
1.4	*READ one*	Visualização de um Paciente	1
1.5	*UPDATE*	Alteração de Paciente	1
1.6	*DELETE*	Arquivamento de Paciente	1
2	**API *Frontend***	**Comunicação Cliente-Servidor em Javascript**	**2**
3	**Páginas *Frontend***	**Páginas em VueJS**	**14**
3.1	Listagem	Visualização de todos os Pacientes	6
3.2	Formulário	Criação, visualização e alteração de Pacientes	8

Tabela 5.7 - Atividades para *Sprint* 1

5.5.2 Desenvolvimento

Utilizando a abordagem *bottom-up*, de forma a fornecer acesso aos dados no *backend* antes de obter qualquer interatividade no *frontend*, a primeira atividade selecionada foi a criação de Banco de Dados, seguida então pelas operações *CRUD*. A escolha da abordagem *bottom-up* se deu por oferecer maior independência entre os dados e o cliente, resultando então em um serviço de dados que pode ser utilizado por qualquer cliente, como no caso de um possível aplicativo na versão mobile nativa para solucionar o mesmo problema.

Como comentado na seção 3.1, para seguir a prática *Desenvolvimento Orientado a Testes* (*TDD*) com exatidão, durante a produção de código seria necessário executar o seguinte ciclo de desenvolvimento, dentro de uma *sprint*:

1. Desenvolver testes para a atividade em desenvolvimento
2. Executar testes e observar falhas
3. Escrever pouco código para procurar solucionar falhas encontradas
4. Executar testes e observar novas falhas
5. Refatorar
6. Repetir a partir de 2, até que não haja mais falhas

No entanto, aplicar tal ciclo a algumas atividades se mostrou pouco benéfico, por atrasar a conclusão da atividade e nem sempre entregar grandes certezas quanto à corretude do código, como no caso de práticas já bastante estabelecidas e testadas. Esse problema se intensifica quando há múltiplos tiers, com diferentes linguagens de programação e *frameworks*, pois há várias ferramentas de testes para se aprender a usar. Por exemplo, no desenvolvimento das operações *CRUD* em PHP, além da ferramenta de testes PHPUnit, utilizada no Laravel, ser ainda desconhecida, o próprio desenvolvimento das operações já é entregue quase completo após a instalação. Tais operações já foram testadas por grande parte da comunidade desenvolvedora e basta apenas configurar o acesso a cada recurso. Dessa forma, os testes para *backend* nesta *sprint* se tornaram muito dispendiosos e foram interrompidos. A validação das operações foi manual e feita através de uma ferramenta de acesso a APIs HTTP chamada Postman.

Um passo importante na arquitetura de APIs Restful, como a implementada no projeto de *backend* neste trabalho, é a definição dos chamados *endpoints*, que são os endereços *URL* de acesso e operações para cada recurso (Ximenes, Juvenal, 2017). A elaboração de *endpoints* deve estar presente na fase de Análise e não deve ser executada durante o desenvolvimento. Deverá haver uma diagramação específica para

definição de *endpoints*. A Tabela 5.8 na seção Endpoints, lista os *endpoints* criados nesta *sprint*.

Com o desenvolvimento do *backend* completo, foi necessário estabelecer uma comunicação entre cliente e servidor. O desenvolvedor notou que seria necessário criar um módulo dentro do sistema para implementar tal funcionalidade. Este módulo não faz parte do Caso de Uso selecionado para esta *sprint* mas é necessário para sua implementação. Tal dependência deveria ter sido evidenciada na fase anterior, de análise do Caso de Uso. Algumas práticas do processo devem ser readequadas para que se encontre maior detalhamento de componentes e dependências do sistema previamente à fase de desenvolvimento. Para o desenvolvimento do módulo de comunicação foi utilizada a prática TDD. A prática não apenas validou a corretude do módulo, mas também explicitou obstáculos que poderiam surgir tardiamente, causando reajustes futuros em código já implementado e funcional. Maiores detalhes na seção Comunicação.

Após todos os testes do módulo de comunicação terem passado com sucesso, ele foi considerado concluído. A seguir houve o desenvolvimento das páginas correspondentes à Listagem e Formulário de Pacientes. Novamente não compensou investir tempo no processo *TDD*. A razão é que no caso da interface gráfica, os componentes de tela finais não podem de antemão ser facilmente identificados e selecionados para testes, especialmente pelo uso de *frameworks* gráficos, que geram elementos automaticamente. Para a construção das páginas, o processo *TDD* se focou nas funções do controlador e pouco na interface gráfica, onde a validação manual posterior foi suficiente. As estimativas de tempo foram um pouco imprecisas, o que impactará nas próximas estimativas.

Endpoints

É necessário definir os *endpoints* e que operações são efetuadas em cada um deles. Eles foram descritos na Tabela 5.8, com o método HTTP seguido da parte relevante do endereço URL e a operação correspondente. O *endpoint* final para

Visualizar um único Paciente com id igual a 23 seria, por exemplo, GET www.toreato.com.br/diario-data/api/pacientes/23.

/pacientes	
GET /	Visualizar todos os Pacientes
GET /<id>	Visualizar Paciente com id <id>
PUT /<id>	Alterar Paciente com id <id>
POST /	Criar novo Paciente
DELETE /<id>	Arquivar Paciente com id <id>

Tabela 5.8 - *Endpoints* para o recurso Pacientes

Comunicação

No projeto de *frontend* foi desenvolvido um módulo de comunicação entre cliente e servidor chamado APIJS, que implementou a interface HTTP em Javascript, ou seja, os métodos Get, Post, Update e Delete, correspondentes aos métodos homônimos básicos do protocolo HTTP. Um módulo é um membro quase completamente isolado do restante do código e por esta razão foi implementado em um diretório próprio no código fonte do projeto *Webclient*, como um subprojeto. Os casos de uso para ele estão demonstrados na Figura 5.5.

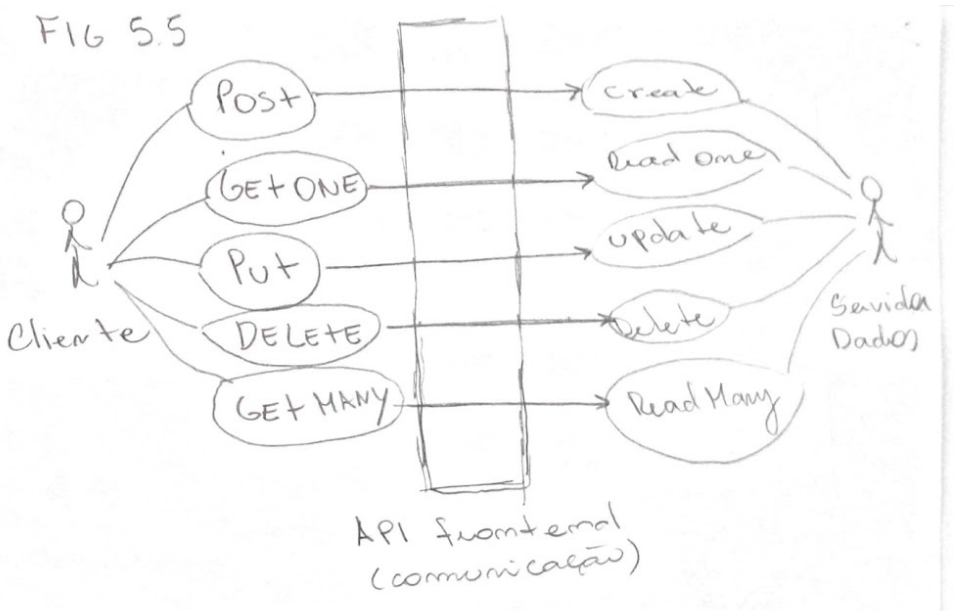


Figura 5.5 - Casos de Uso para módulo Api no *frontend*.

Qualquer biblioteca de comunicação HTTP seria capaz de fornecer uma solução direta para este problema. Porém utilizar uma biblioteca dessa forma, diretamente no código fonte, é uma péssima prática por criar um forte acoplamento entre o projeto e a biblioteca. Para evitar este problema, bastou que o módulo seguisse o padrão de projeto *adapter*, de forma a desacoplar o módulo de *frontend* e a biblioteca de comunicação sendo utilizada. Foi escolhido utilizar a biblioteca Axios, por ser bastante popular e, dessa forma, existir muita experiência a seu respeito na Web, além de possuir suporte suficiente. Para o módulo, foi decidida uma interface com os métodos *get*, *put*, *post* e *delete*.

Para seguir o *TDD*, foram criados testes para os quatro métodos citados, utilizando a biblioteca Jest, para testes em Javascript. Como não havia dados cadastrados no servidor em desenvolvimento, para os testes foi utilizado um outro servidor, chamado Json Placeholder, que fornece dados no formato padrão da Web, para aplicações que necessitam de dados, mas ainda não os possuem. Durante a criação de testes surgiram dúvidas quanto ao funcionamento do Axios e de que forma seriam identificados erros durante as requisições. O Axios modifica os dados retornados para incluir informações extra, inclusive relativas a erros. Os dados devem ser

convertidos para um formato padronizado, independente de bibliotecas, durante o mapeamento entre os métodos do módulo e do Axios. No caso de erros, os seus dados devem também seguir um padrão compreensível pela aplicação para que ela tenha a possibilidade de oferecer a melhor experiência ao usuário.

Outra questão que se demonstrou importante foi o desacoplamento entre o endereço do servidor de dados e o módulo Api. Se o endereço fosse definido no código do módulo, em caso de mudança de servidor, seria necessário alterar o código original, o que aumentaria o risco de erros retroativos. Foi definido um novo método para a sua interface: *setBaseUrl*, que define o endereço base para as requisições à Api de dados, restando apenas indicar qual *endpoint* acessar a partir daquela base.

5.5.3 Entrega

Nesta *sprint* a entrega foi bastante ágil, se comparada à entrega anterior, porém ainda utilizando o protocolo FTP para transferência de arquivos entre os ambientes de desenvolvimento, no computador do desenvolvedor e o ambiente de produção, no servidor *Hostgator*, o que não é ideal. Esta dificuldade permanecerá em aberto nesta *sprint*. Caso não haja uma solução com o ambiente em produção atual, deve se considerar o custo-benefício de se utilizar um ambiente de entrega mais ágil em projetos futuros.

5.5.4 Verificação

Foi observado que para este projeto a prática de Estórias de Usuário não é a mais adequada para a análise de requisitos. Casos de Uso entregam informações similares quanto ao uso do sistema pelo usuário final, mas, apesar de menos específicos que Estórias de Usuário, Casos de Uso podem ser mais rapidamente diagramados e validados pela *PO*. A fase de Análise também deve entregar um projeto de *endpoints* para o desenvolvimento do *backend*.

Uma das dificuldades do desenvolvedor individual é que, enquanto uma equipe pode atribuir a um de seus membros a responsabilidade de solucionar problemas, como

o de entrega automática, um desenvolvedor individual, ao respeitar a prática *Trabalho Energizado,* para procurar soluções, deve utilizar das horas destinadas à produção do produto final, atrasando então o desenvolvimento. Como problemas não estão sempre dentro do planejamento, esta situação criaria atrasos e geraria um débito de prazo ou pagamento com o cliente. Para manter o cronograma, talvez seja necessário prolongar um pouco a jornada de trabalho, mesmo não sendo o ideal.

No caso específico do problema de entrega automática, neste *projeto* ela não será utilizada e consequentemente a prática *Entrega Rápida* será abandonada. Se o custo do processo de entrega manual for muito alto, em próximos projetos deve se contratar servidores de hospedagem com sincronização automática com o repositório de versionamento remoto.

Alguns ramos de versionamento incluiram implementações muito longas, de diversas funcionalidades, contrariando a prática *Base de Código Unificada* e *Integração Contínua*. Essas práticas são importantes de modo a garantir que funcionalidades implementadas corretamente não sofram interferência da implementação de outras funcionalidades no mesmo ramo. Isolando implementações um desenvolvedor pode apenas excluir uma implementação indevida, ou pode se mover pela linha do tempo sem correr riscos em funcionalidades já concluídas.

A prática de *Entrega Diária* se mostrou inadequada, ao menos enquanto seguindo a abordagem *bottom-up*, pois na maioria dos dias não se concluiria atividades suficientes para entregar novo valor ao cliente. Deve-se tentar isolar o desenvolvimento a cada Caso de Uso. No lugar de implementar todo o *backend* e posteriormente todo o *frontend*, pode-se implementar cada Caso de Uso por completo, antes de passar ao próximo. Desta forma, será possível concluir um Caso de Uso completo e diariamente entregar valor ao cliente.

Prática	Está Adequado?
Descrição de Casos de Uso (UC)	Casos de Uso foram adequados, mas será necessário haver práticas que evidenciem componentes e relacionamentos no código.

Descrição de Estórias de Usuário (EU)	Não. Utilizar EUs se demonstrou mais oneroso que proveitoso ao projeto.
Base de Código Unificada	Alguns ramos se demonstraram demasiadamente grandes, causando bifurcações muito duradouras.
Integração Contínua	É viável integrar mudanças continuamente num processo individual, mas é preciso utilizar ramos mais curtos.
Implantação Diária	Não. Utilizando a abordagem *bottom-up*, o código apenas foi entregue ao final da *sprint*. Outra abordagem deve ser utilizada para tentar a Implantação Diária.
Entrega Rápida	Não, o servidor contratado não oferece ferramentas para isso.
TDD	Foi utilizado apenas em algumas atividades.
Trabalho Energizado	Pode ser necessário prolongar a carga de trabalho.

Tabela 5.9 - Avaliação das práticas no *sprint* 1

5.5.5 Adequação

Nesta *sprint* houve uma sobreposição das fases de Análise e Desenvolvimento, onde processos que deveriam estar presentes na Análise terminaram executados durante a fase de Desenvolvimento. A ausência do módulo de comunicação com o *backend* na Análise da *sprint* demonstrou que a prática de elaboração de Casos de Uso é insuficiente para projetar o sistema, de forma a evidenciar uma visão geral dos componentes do sistema e seus relacionamentos. Dessa forma, as estimativas e planejamento se tornaram arriscados e imprecisos. Dado que projetar componentes não é a função do Diagrama de Casos de Uso, um Diagrama de Modelagem deverá ser utilizado. Em projetos de sistemas, Diagramas de Sequência e de Classes são as soluções costumeiras para este problema, por representar tanto dinamicamente, quanto estaticamente, os modelos de componentes e seus relacionamentos. Para a Web, há uma versão extendida, mais adequada, do Diagrama de Classes, chamada *WAE* (*Web Application Extension*), que será utilizada. Uma nova prática foi adicionada à lista de práticas, chamada Modelagem, que inclui os Diagramas citados, além do já utilizado

Wireframe, com critério de adequação "É possível identificar os componentes relevantes e seu relacionamento com outros componentes", vide Tabela 5.10. É importante salientar que um desenvolvedor individual não necessita comunicar ideias técnicas com ninguém. Por esta razão, se faz menos necessário o uso de uma linguagem de diagramação muito rigorosa, que custaria um tempo desnecessário para produzir. O enfoque é a compreensão do indivíduo.

Inicialmente se utilizou ramos de versionamento para as páginas inteiras, o que resultou na quebra das práticas de *Base de Código Unificada* e *Integração Contínua*, pois o ramo iniciado em um dia se manteve em aberto para ser concluído no dia seguinte, sendo então considerado muito duradouro, segundo as recomendações do XP. Ao invés de eliminar as práticas, este problema pode ser solucionado com a utilização de ramos com enfoques menores e mais específicos. No lugar de um ramo para uma página inteira, serão criados ramos para, por exemplo, a criação da infra estrutura da página, que conecta o código da página ao resto do framework, ou ramos para componentes da interface gráfica, como um menu, ou um formulário.

Prática	Critérios
Modelagem 1. Diagrama WAE 2. Diagrama de Sequência 3. Wireframe	É possível identificar os componentes relevantes e seu relacionamento com outros componentes, sem que se desperdice tempo com normas de modelagem.
Base de Código Unificada e Integração Contínua	Ramos para funcionalidades pequenas podem ser integrados no máximo em um mesmo dia de trabalho. Enfoque em funcionalidades mais específicas. Ex: no lugar de um ramo para uma página, um ramo para cada componente da página.

Tabela 5.10 - Adequação das práticas após *sprint* 1

5.6 Sprint 2 - Autorização

5.6.1 Análise

Utilizando o "Diagrama de Épicos", a PO decidiu quais funcionalidades seriam mais interessantes para serem entregues nesta *sprint*. A seção Funcionalidades detalha este momento de decisão. Dessa vez seria necessário desenvolver a modelagem dos Casos de Uso escolhidos, de forma a identificar componentes do sistema e posteriormente atividades de desenvolvimento. Antes de construir Diagramas WAE, percebeu-se a necessidade de se analisar o fluxo de dados para a correta identificação de componentes. Com tal informação, ficaria mais claro quais são os componentes e de que forma eles estavam envolvidos em cada operação. Por esta razão, decidiu-se então pelo uso primeiro do Diagrama de Sequência (Figuras 5.6 e 5.7). A partir destes Diagramas foi identificado um novo componente do sistema, que armazena informações globais. Com os componentes corretamente identificados, foi possível construir um Diagrama WAE para cada UC (Figuras 5.8 e 5.9). Os dois diagramas foram muito similares e é mais proveitoso utilizar um único diagrama com todos os componentes da *sprint*. Duas páginas foram identificadas nos Diagramas WAE e para cada uma foi desenhado um *Wireframe* como referência para estimativas e validação da PO, na Figura 5.10.

Toda esta modelagem pôde ser elaborada na metade de um dia de trabalho e possibilitou a identificação e o projeto de componentes do sistema. Desta vez os componentes necessários para a implementação dos Casos de Uso escolhidos puderam fazer parte do planejamento da *sprint* e sua implementação também foi transformada em atividades de desenvolvimento, vide Tabela 5.10, na seção Atividades.

Funcionalidades

Com o cadastro de pacientes funcionando, já com alguns pacientes cadastrados, a *PO* passou a demonstrar um receio recorrente, pois a incomodava ver publicamente todos os pacientes cadastrados, sem nenhuma restrição por credenciais, de forma a

proteger as suas informações. Em outras palavras, as páginas de Pacientes deveriam ser Privadas. Páginas são consideradas públicas quando não é necessário estar autenticado para acessá-las. Caso contrário são consideradas páginas privadas. Além do incômodo declarado pela *PO*, após implementar os Casos de Uso de *Autorização,* seria necessário se atrelar os pacientes e qualquer outro recurso cadastrado ao usuário correto. Por esta razão, decidiu-se por implementar os Casos de Uso de *Autorização* nesta *sprint*, *Cadastrar-se* e *Autenticar-se*. Como ambos são Casos de Uso simples, não há como quebrá-los de forma a obter UCs mais específicos. Ambos são suficientemente específicos, de forma a poderem ser implementados como estão.

Modelagem

As Figuras 5.6 e 5.7 mostram os Diagramas de Sequência desenvolvidos para Cadastro e Login.

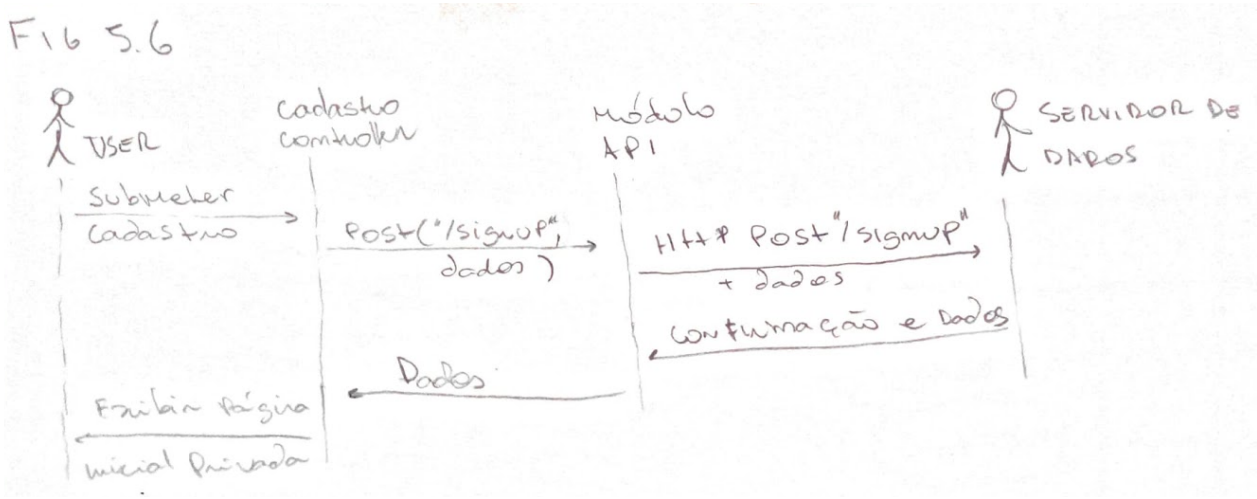


Figura 5.6 - Diagrama de Sequência de Cadastro

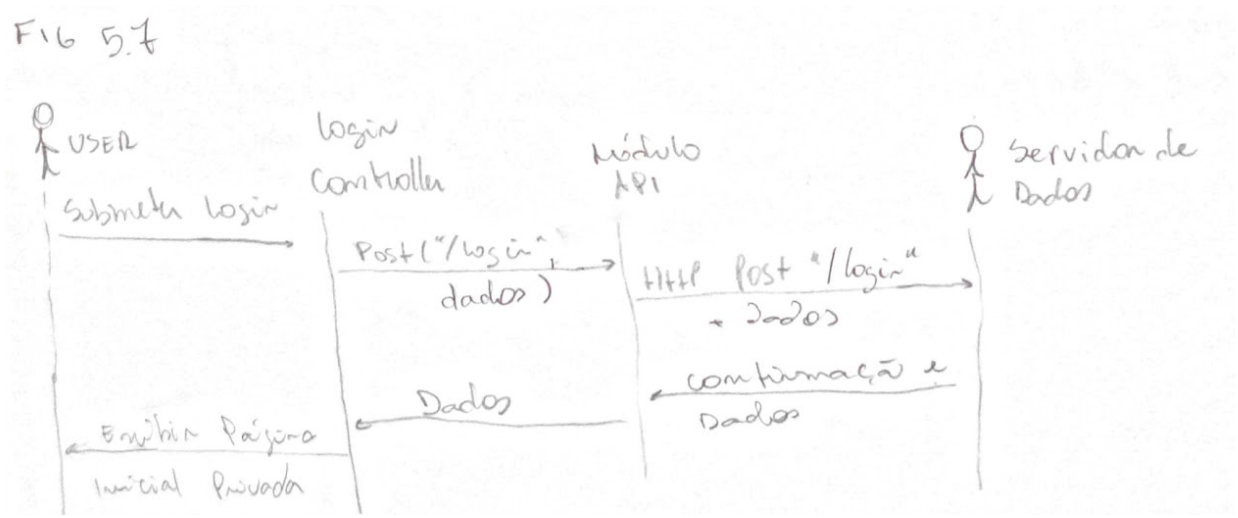


Figura 5.7 - Diagrama de Sequência de Login

Uma questão importante surgiu ao se analisar tais diagramas. Após autenticado e redirecionado para a página privada inicial, os dados retornados durante o login seriam perdidos, pois estavam armazenados apenas em uma variável local em *LoginController*. Este é um problema recorrente no desenvolvimento Web e já possui diversas soluções diferentes. A solução recomendada para o *framework* VueJs é o *plugin* Vuex, que armazena dados globalmente, mas com acesso estruturado e organizado, através de métodos bem definidos. Esta forma de utilizar dados globais, compartilhados entre múltiplos componentes é chamado *State Management Pattern* e usa a prática *Single Source of Truth* (FILGUEIRA, [2018]). Foram então construídos os Diagramas WAE, incluindo o gerenciador de estado global, chamado *Global Store*, resultando nas Figuras 5.8 e 5.9, para cadastro e login, respectivamente.

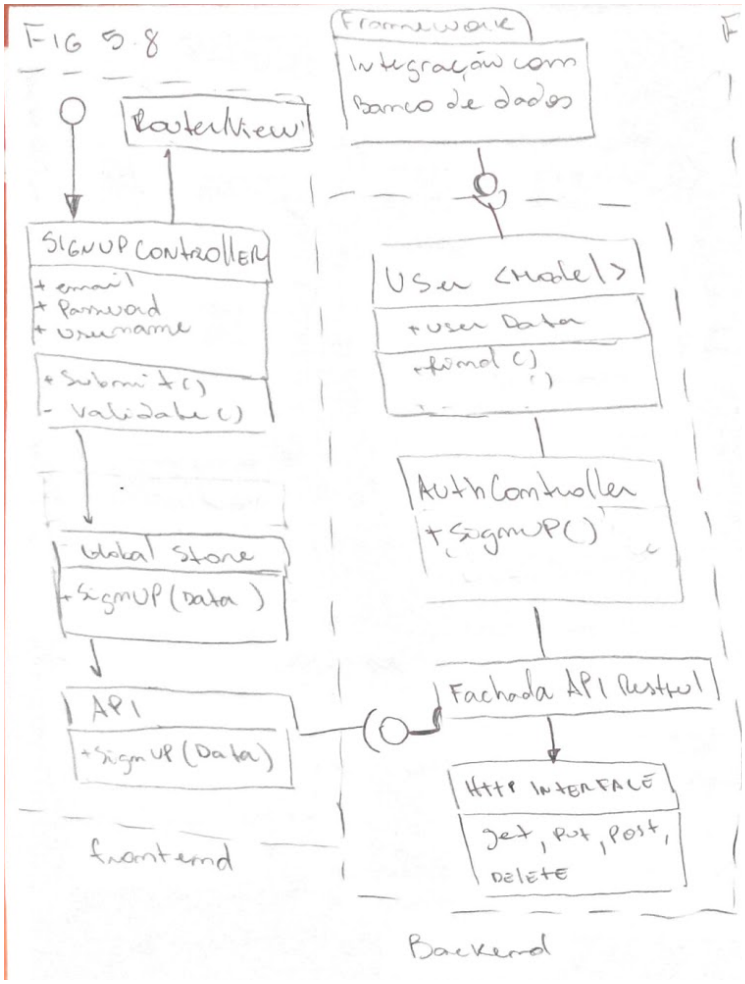


Figura 5.8 - Diagrama WAE para o Caso de Uso Cadastrar-se

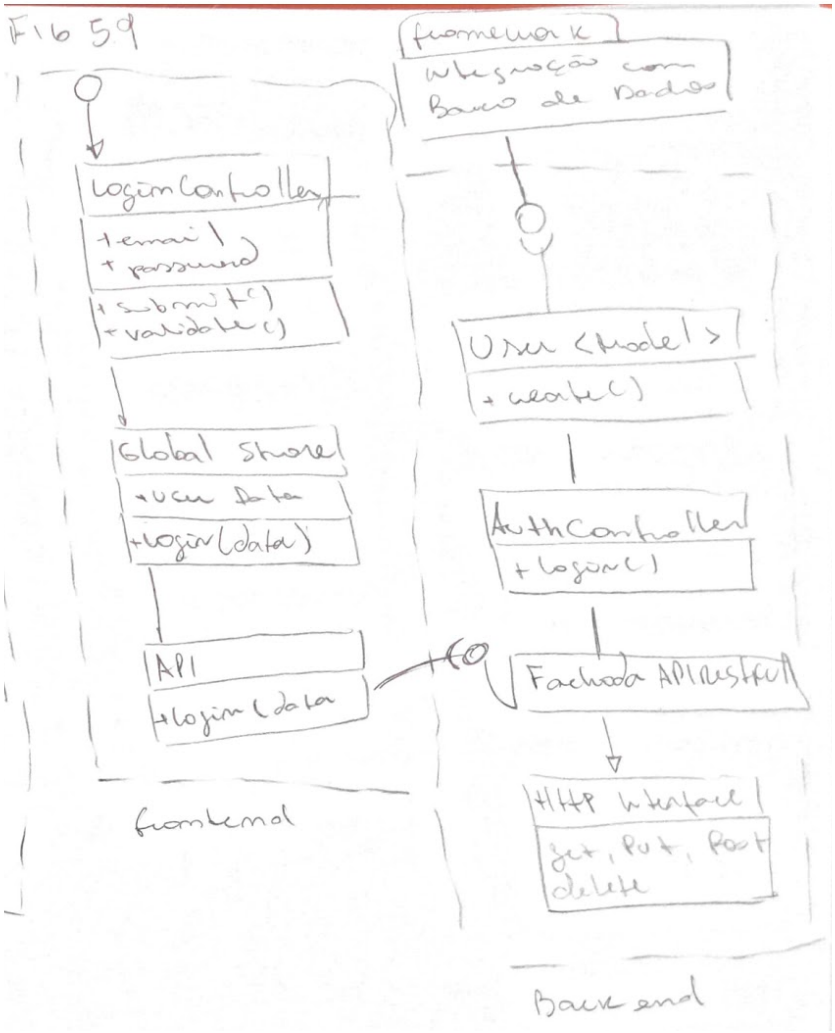


Figura 5.9 - Diagrama WAE para o Caso de Uso Autenticar-se

Para projetar a construção dos componentes *LoginView* e *SignupView* foi necessário um pequeno esboço do resultado final com a prática *Wireframes* de forma a identificar o nível de complexidade do desenvolvimento dos mesmos. A proposta para ambos é que exija o menor esforço possível do usuário, de forma que rapidamente possa efetuar um cadastro, ou fazer *login*. O cadastro exige apenas as informações estritamente necessárias e ambos os *wireframes* estão representados na Figura 5.10 e foram adicionados ao *Backlog* do *sprint*, junto aos outros diagramas criados durante o *Jogo do Planejamento* desta *sprint*.

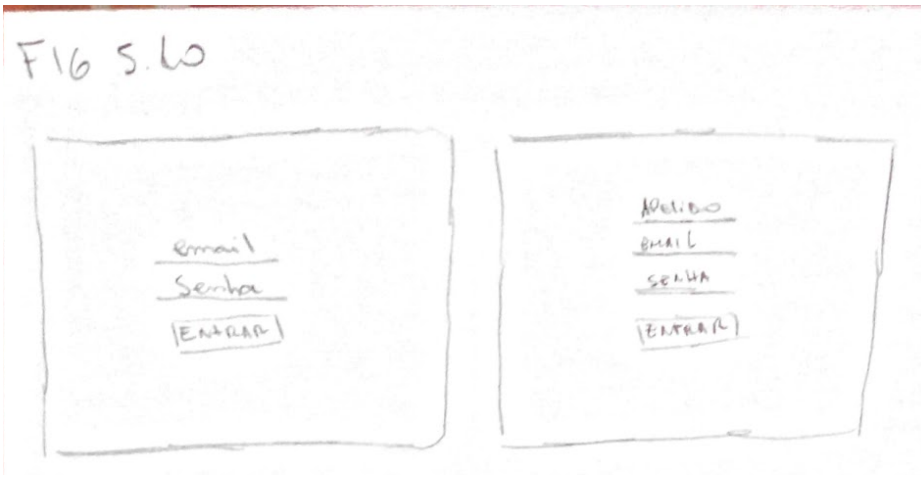


Figura 5.10 - *Wireframes* para o *Sprint* 2. *Login* à esquerda, cadastro à direita

Atividades

Para Autorização foi possível implementar todo o *Épico* de uma só vez, pois o sistema de Autenticação/Autorização vem pronto com o Laravel, bastando alguma configuração adicional no *backend* e, no *frontend,* o módulo Api e duas páginas simples de cadastro e login, como nos *Wireframes*. Antes de implementar seus Casos de Uso, porém, foi necessário instalar e configurar o plugin Vuex, como mencionado. As atividades para esta *sprint* foram listadas na Tabela 5.11.

#	Título	Descrição	Estimativa (hrs)
1	**Instalar *Global Store Vuex***	Download via *npm* e configuração	**0.5**
2	**API *Backend***	*Endpoints* automáticos	**3**
2.1	Banco de Dados	Criar tabela para Usuários e	1
2.2	Modelo	Criar classe modelo *User* para acesso ao banco de dados	1
2.3	Configuração	Configurar classe User para ser utilizada no sistema de autenticação	1
3	**API *Frontend***	Adicionar métodos login e signup à interface	**3**
3.1	Login	Deve verificar email e senha, guardando *token* de acesso, para requisições posteriores	2

3.2	Signup	Apenas envia dados ao *endpoint* correto	1
4	***Store auth***	Deve conter ações de autenticação, que invocarão a API *Frontend*, e armazenar dados retornados da API *Backend*	**2**
4.1	Login		1
4.2	Signup		1
5	**Páginas *Frontend***		**8**
5.1	Cadastro	Página com formulário simples, apenas nome de usuário, email e senha	4
5.2	*Login*	Formulário similar, porém sem nome de usuário	4
6	**Configuração**		**3.5**
6.1	Página inicial	Tornar *Login* a página inicial	0.5
6.2	Proteger páginas privadas	Páginas consideradas privadas apenas podem ser visualizadas quando usuário estiver logado, com dados de usuário armazenados na *Store auth*. Caso contrário redirecionar para *Login*.	3
6.3	Redirecionar páginas públicas	Páginas consideradas públicas devem redirecionar usuário já autenticado para a página de pacientes.	2

Tabela 5.11 - Atividades para *Sprint* 2

5.6.2 Desenvolvimento

O desenvolvimento do *backend* para esta *sprint* foi bastante simples e rápido, por vir quase inteiramente funcional com o *framework* utilizado. Pelo procedimento ser bastante simples, todo o *backend* foi criado em um único ramo no sistema de versionamento. Esta prática pode não ser adequada para equipes, mas para o caso de um desenvolvedor individual foi mais eficiente, por não interromper o fluxo de trabalho para a criação, aceitação e fusão de vários *Pull Requests*. Mais detalhes na seção Autenticação. A verificação da corretude do *backend* afetou pouco no tempo de desenvolvimento. Essa rapidez em verificar a corretude de uma parte do programa não pode ser sempre garantida e nem subestimada, pois algumas verificações exigem

vários testes e validações diferentes. Por esta razão, a verificação deve ser considerada uma atividade de desenvolvimento com estimativa de tempo própria. Nas próximas *sprints* deve ser criada atividades apenas para verificações.

A prática TDD foi novamente utilizada para incrementar funcionalidades no módulo de comunicação do *frontend*, mas, como explicado na seção Testes, não foi possível isolar outras partes do sistema para efetuar testes unitários seguindo a mesma prática. Toda a implementação efetuada sem a TDD foi validada manualmente, com sucesso.

Todo o resto do desenvolvimento se deu como esperado. As páginas foram compostas apenas por simples formulários, como indicado nos *wireframes*. Foi possível implementá-las em dois ramos de desenvolvimento para cada, um para configurar e outro para inserir o formulário. Mais detalhes na seção Páginas.

Store

O plugin Vuex pôde ser rapidamente adquirido através do gerenciador de pacotes NPM. Com ele, basta um comando no terminal do sistema operacional para que o plugin esteja disponível para utilização no código. Foi criado um ramo de versionamento para a sua instalação.

Autenticação

Foi necessário primeiro adquirir o plugin *Passport* para Laravel. Então foi necessário criar uma tabela *users* para armazenamento de usuários no banco de dados e uma classe *User*, com métodos de acesso à mesma. Todas estas atividades levaram quatro horas para serem concluídas. Este resultado ultrapassou a estimativa em uma hora. A diferença se deu em especial por ter sido a primeira vez que o desenvolvedor utilizou a versão corrente do Laravel, sendo então necessário estudar as diferenças em sua documentação.

Testes

O desenvolvimento da API de *frontend* seguiu a prática *TDD*. Para rodar os testes automaticamente, uma biblioteca de dados de formulários falsos chamada Faker foi utilizada. Um problema que surgiu foi que a cada execução, os testes criavam um novo usuário. Não há Caso de Uso para remoção de usuários e por esta razão, essa funcionalidade ainda não foi implementada. Como demonstrado nos Diagramas WAE das figuras 5.8 e 5.9, existe uma relação de associação entre a Store e o módulo de comunicação Api, que conecta o *frontend* com o *backend*. Não foi possível encontrar uma solução rápida para isolar ambos. Em testes unitários é necessário encontrar esse isolamento, para que apenas uma parte do sistema seja testado. Dessa forma, o seu desenvolvimento não seguiu a prática TDD nesta *sprint*. Uma solução para esta questão deve ser buscada a fim aumentar a confiabilidade no desenvolvimento da *Store* futuramente. A parte da *Store* referente à Autorização foi implementada com sucesso, com validação manual.

Páginas

A interação entre as páginas e a API de dados se deu entre o controlador da página e a *Store* intermediária, de forma a garantir a permanência dos dados relevantes em memória. Além das páginas, foi necessário alguma configuração de rotas e *scripts* de proteção e redirecionamento de páginas públicas e privadas, baseado nos dados da *Store*. Essa configuração exigiu um pouco mais de pesquisa que o esperado, totalizando 6 horas de desenvolvimento.

Não foi criado um ramo de desenvolvimento para cada atividade, como havia sido planejado. A integração dos ramos em forma de *Pull Request* (*PR*) é um tanto burocrática e tiram o foco do desenvolvimento, concentrando-o no sistema de versionamento. Ramos muito pequenos resultam em muitos *PRs* e por esta razão, um ramo foi criado para cada conjunto de atividades, categorizados por temas. Os ramos foram: *Backend*, Api de *Frontend*, *Store*, *Página de Login*, *Página de Cadastro* e Guardas de Acesso.

5.6.3 Entrega

Não houve maiores problemas na transferência de código para o servidor em produção. Ficou decidido que o método de entrega a cada *sprint* permaneceria ocorrendo via protocolo FTP, manualmente. Apesar de lento, o método FTP ainda é viável para pequenas entregas, como são as produzidas num intervalo de uma *sprint*.

5.6.4 Verificação

Os Diagramas escolhidos se mostraram úteis para a modelagem rápida do sistema. O Diagrama de Sequência auxiliou na descoberta de componentes importantes envolvidos em cada operação, enquanto o Diagrama WAE formalizou seus relacionamentos, interfaces e estado interno. Apesar de útil, facilitando a visão geral de componentes, a definição do Diagrama WAE não entregou maiores informações e não possui um custo-benefício adequado para ser utilizado por um indivíduo quando desenvolvendo funcionalidades simples. Na próxima *sprints* será elaborado um único Diagrama WAE para todas as funcionalidades, de modo a reduzir o tempo investido. Os *Wireframes* foram de grande importância no projeto das páginas, em especial para estimar a duração do seu desenvolvimento.

Como mencionado na seção de 5.6.2, não foi adequado criar diversos ramos de versionamento, um para cada pequena atividade, porque sua integração é um processo um pouco burocrático, que retira o foco sobre o desenvolvimento. Dessa forma, a prática Base de Código Unificada, que define pequenos ramos de curta duração, não é completamente adequada. Foi decidido criar grupos de atividades, definidos por proximidade, que seria implementado em um único ramo de versionamento. Os grupos seriam definidos por critério de cronologia. Atividades que fossem implementadas em sequência e tivessem algum sentido estarem agrupadas, estariam no mesmo grupo, contanto que não permanecessem ramificadas até o dia de trabalho seguinte.

Por falta de habilidade com testes automáticos, muitas partes do sistema foram verificadas manualmente. É necessário discernir onde é imprescindível utilizar a prática TDD e onde a validação manual é suficiente.

A Tabela 5.12 traz um resumo das observações para cada prática mencionada.

Prática	Está Adequado?
Modelagem 1. Diagrama de Sequência 2. Diagrama WAE *3. Wireframes*	Deve se atentar à burocratização das práticas e caso não se torne mais eficiente, talvez seja possível eliminar o Diagrama WAE da prática Modelagem. Para funcionalidades mais complexas pode ser necessário adicionar novas práticas.
Base de Código Unificada	Não foi interessante para manter ramos de versionamento muito pequenos, que contivessem implementações de funcionalidades muito pequenas. A integração de cada ramo exige um processo que reduz o foco no desenvolvimento.
Desenvolvimento Orientado a Testes (TDD)	Deve se encontrar formas de aplicar mais testes ao desenvolvimento sem atrasos. Se não for possível, identificar quais tipos de atividades são mais adequadas para testes automáticos e quais devem ser verificadas manualmente.

Tabela 5.12 - Avaliação das práticas no *sprint* 2

5.6.5 Adequação

Durante esta *sprint* a prática Base de Código Unificada se destacou como muito burocrática e foi decidido agrupar atividades menores e implementá-las em ramos que durariam até o limite de um dia. Dessa forma não se perderia tempo com o processo de efetuar um *Pull Request*, acessar o site *Github*, fazer o merge do ramo menor ao ramo de desenvolvimento principal, retornar ao ambiente de trabalho, atualizar o ramo principal e ramificá-lo para uma nova tarefa. Para uma equipe, pode ser necessário seguir tal processo, de forma a evitar conflitos entre ramos paralelos e manter toda a equipe atualizada constantemente. Para um único indivíduo, porém, executar tal procedimento para cada atividade faz pouco ou nenhum sentido. Será mais proveitoso utilizá-lo para criar pontos de versionamento para um conjunto de funcionalidades sendo implementadas.

A prática TDD não está sendo aplicada na maior parte do desenvolvimento. Nas próximas *sprints* a sua utilização deverá se concentrar no desenvolvimento da *Store* e

na busca por novas aplicações em outros tipos de atividades. É possível que esta prática exija uma especialização maior que um desenvolvedor individual comum pode oferecer, visto que não há tempo disponível para investir no aprimoramento desta técnica. Em uma equipe, um membro diferente fica responsável por elaborar tais testes e dessa forma não atrasa a entrega do produto.

A Tabela 5.13 demonstra as novas adequações para as práticas comentadas.

Prática	Critérios
Base de Código Unificada	Ramos para conjuntos de atividades pequenas podem ser integrados em no máximo um mesmo dia de trabalho.
Desenvolvimento Orientado a Testes	É possível validar automaticamente tanto os procedimentos, quanto o estado resultante da aplicação.

Tabela 5.13 - Adequação das práticas após *sprint* 2

5.7 Sprint 3 - Rotinas

5.7.1 Análise

Analisando o artefato "Diagrama de Épicos" e os UCs já implementados, foi possível identificar os próximos UCs a serem implementados. Como o maior objetivo no desenvolvimento do sistema para a Reato é a visualização da rotina de pacientes, representada nos casos de uso do Épico de *Visualização*, e dado que os UCs remanescentes do Épico *Cadastros* não são pré requisitos para os UCs do Épico *Visualização*, a PO decidiu fazer do Caso de Uso "Criar Rotina" a próxima prioridade, por entregar mais valor dentre as alternativas. A seção Funcionalidades explica em maiores detalhes tanto a decisão por este Caso de Uso, quanto o seu relacionamento com outros Casos de Uso.

Logo no início da modelagem dos Casos de Uso, foi notado um crescimento desnecessário de artefatos, com a criação de pelo menos dois diagramas para cada um. A prática *Código e Testes* recomenda que se evite o acúmulo de artefatos além dos especificamente voltados para entregar valor ao cliente. Para se adequar a essa prática, os diagramas da prática *Modelagem* foram comprimidos em um único diagrama para cada objetivo, um único Diagrama de Sequência para identificar todos os componentes e um Diagrama WAE para projetá-los. As Figuras 5.11 e 5.12, na seção Modelagem, representam respectivamente os Diagramas de Sequência e WAE unificados para todos os Casos de Uso desta *sprint*.

Tais diagramas esclareceram questões importantes quanto ao projeto do sistema. Nesse momento já era possível estabelecer quais componentes seriam necessários para o completo desenvolvimento do Caso de Uso para a *Sprint* atual. No diagrama WAE há um tipo de relacionamento de *Hiperlink*, com a legenda <*Redirect*>, que conectou o controlador de Visualizar Rotinas *RoutineListController* e os controladores de Criar Rotina e Atualizar Rotina, respectivamente NewRoutineController e EditRoutineController, indicando haver alguma forma de transitar entre as páginas atreladas a estes controladores (CONALLEN, 1999). Com essa alteração foi identificada

uma nova necessidade da fase de Análise da *sprint*: determinar as conexões entre as páginas, indicando qual caminho um usuário deve percorrer até encontrar um determinado conteúdo. A técnica usual utilizada com esse fim é a de *Site Maps*, que são estruturas de dados em forma de Árvore Enraizada Direcionada, onde cada folha representa uma página no *Website* e cada aresta representa uma conexão de *Hiperlink* entre duas páginas. Por ser direcionada, é possível que uma conexão tenha um único sentido, representando a situação de quando uma página dá acesso à outra, mas não há caminho de retorno direto. Tal técnica pode ser utilizada em qualquer aplicação navegável pelo usuário, inclusive sistemas Web (CARDELLO, 2017). Site Maps foram utilizados tanto no *frontend*, descrevendo percursos entre páginas, quanto no *backend* descrevendo os *endpoints* e suas operações. Um *Site Map* genérico foi criado para qualquer tipo de recurso no *backend*, incluindo possíveis relacionamentos com outros recursos, de forma a poder ser reaproveitado. Ambos estão descritos na seção Modelagem, com as Figuras 5.13 e 5.14, para *backend* e *frontend*, respectivamente.

Durante a listagem de Atividades para esta *sprint*, foi encontrada uma atividade cuja estimativa não possuía referências até o momento. Neste caso, uma boa estimativa, recomendada por alguns autores independentes na internet (ALLAN, 2014; "BEN", 2018) e inspirada na Lei de Hofstadter (HOFSTADTER, 1999), seria o dobro da expectativa. Essa "lei" reafirma a dificuldade em ser preciso em estimativas de tempo. A expectativa foi calculada através de referências para a atividade em questão, como as estimativas para outras atividades. A lista de atividades para esta *sprint*, bem como as estimativas de tempo para cada uma estão explicitadas na seção Atividades, Tabela 5.11.

Funcionalidades

Após a execução das *sprints* anteriores, onde foram implementados os Casos de Uso de *Autorização* e "Cadastrar Pacientes", poderiam ser implementados os UCs "Cadastrar Médicos" e "Cadastrar Cuidadores", do Épico *Cadastros* e "Criar Rotina" do

Épico *Visualização*. Como anteriormente informado, a PO decidiu fazer do Caso de Uso "Criar Rotina" a próxima prioridade.

Rotinas são um conjunto de Atividades realizadas num dia específico. Dado que uma Rotina contém múltiplas Atividades, inclusive possivelmente nenhuma, e que cada Atividade pertence sempre necessariamente a apenas uma Rotina, há uma relação um-para-muitos entre ambos. Por esta razão, a criação das Rotinas pode ser independente da criação das Atividades. O inverso não seria possível, porém. Um Diagrama de Casos de Uso para esta *sprint* foi criado com as funcionalidades: "Criar Rotina", "Atualizar Rotina", "Visualizar Dados de Rotina", "Arquivar Rotina" e "Visualizar Todas as Rotinas".

Modelagem

Os Diagramas unificados entregam uma visão mais geral do sistema, além de reduzir o número de artefatos. Por outro lado, eles são mais difíceis de ler e produzir. Por esta razão, foram produzidos digitalmente nas Figuras 5.11 e 5.12.

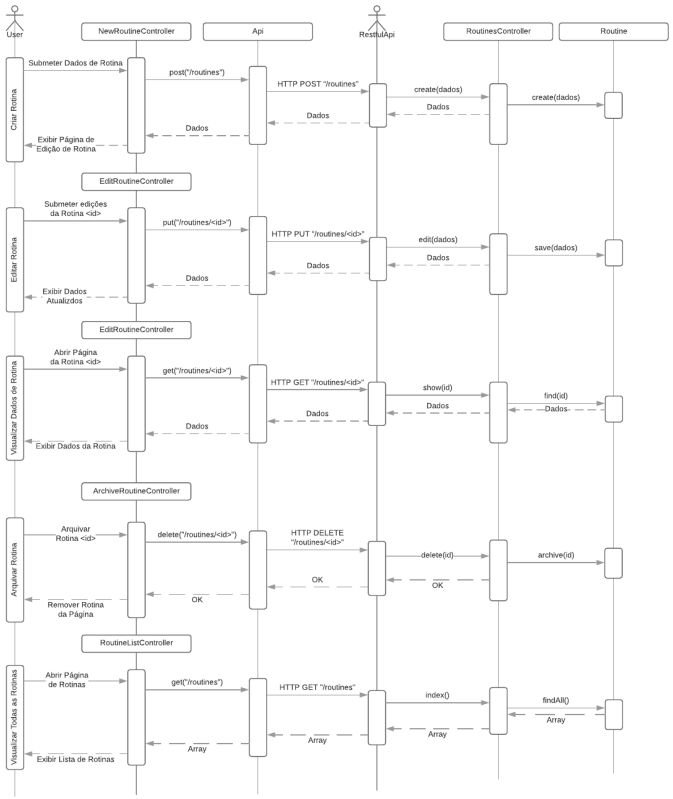


Figura 5.11 - Diagrama de Sequência Único para Casos de Uso do *Sprint* 3

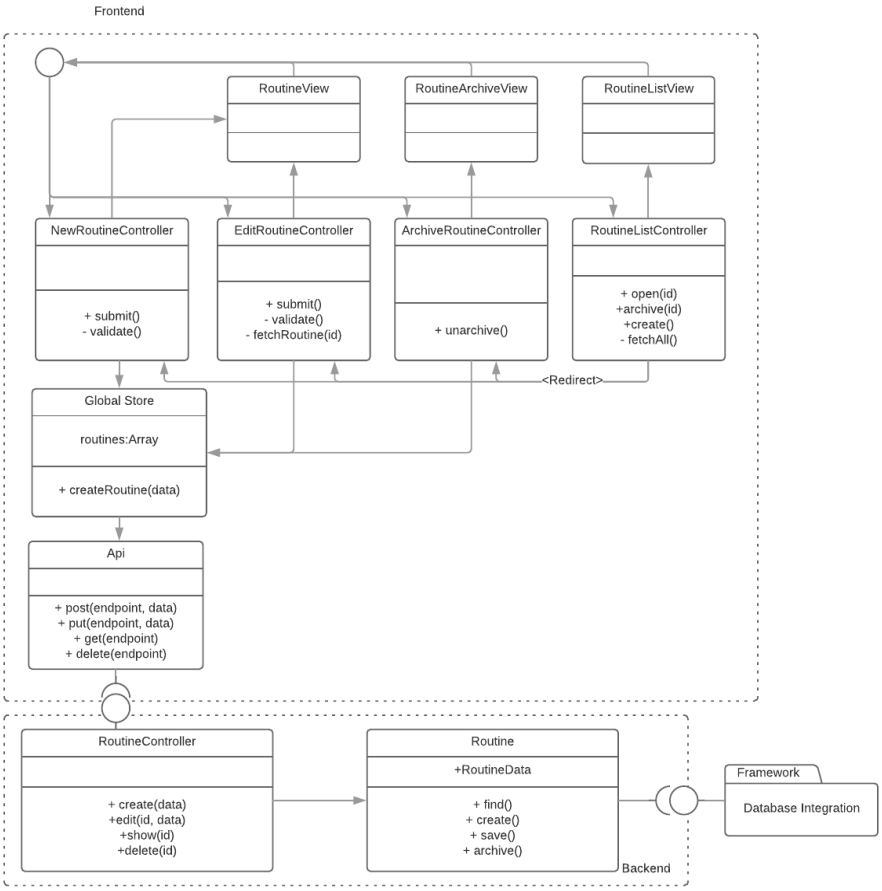


Figura 5.12 - Diagrama de Classes Único para Casos de Uso do *Sprint* 3

Foi desenvolvido o *Site Map* parcial para as páginas envolvidas na *sprint* atual, com raiz na página principal para o recurso Rotinas, "Listar Rotinas". Tal artefato está representado na Figura 5.13. A mesma técnica também foi utilizada para desenhar rotas na Api de *backend*. Como grande parte dos recursos possuem operações e *endpoints* similares, foi criado um *Site Map* genérico, com as operações mais básicas para qualquer recurso e seus relacionamentos com outros recursos. Cada nó raiz é um *endpoint* e cada folha significa uma operação disponível nele, através de um método http. Após concluído, dado que o *Site Map* não entrou em conflito com os diagramas produzidos anteriormente, foi considerado adequado como estava. O *Site Map* de *Endpoints* genérico e o específico para esta *sprint*, criado para exemplificar o uso do Mapa genérico, estão representados nas Figuras 5.14 e 5.15, respectivamente.

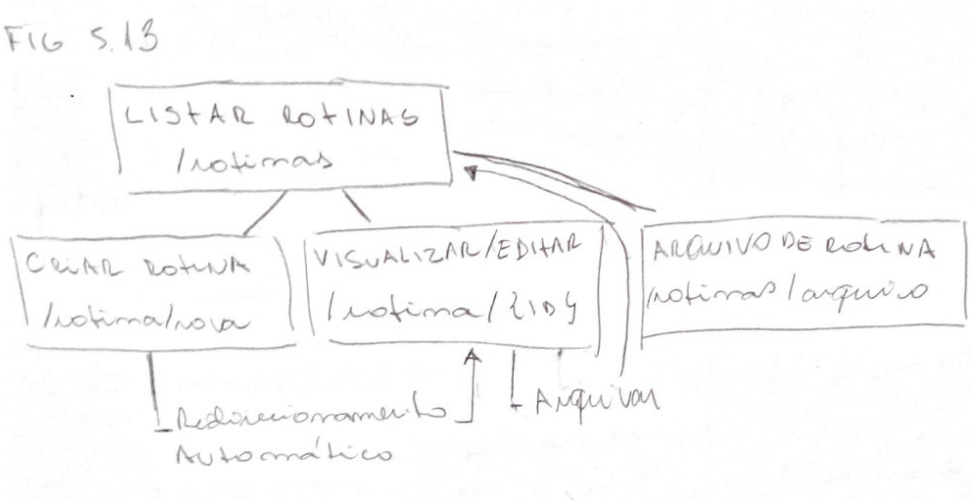


Figura 5.13 - *Site Map* do *Sprint* 3

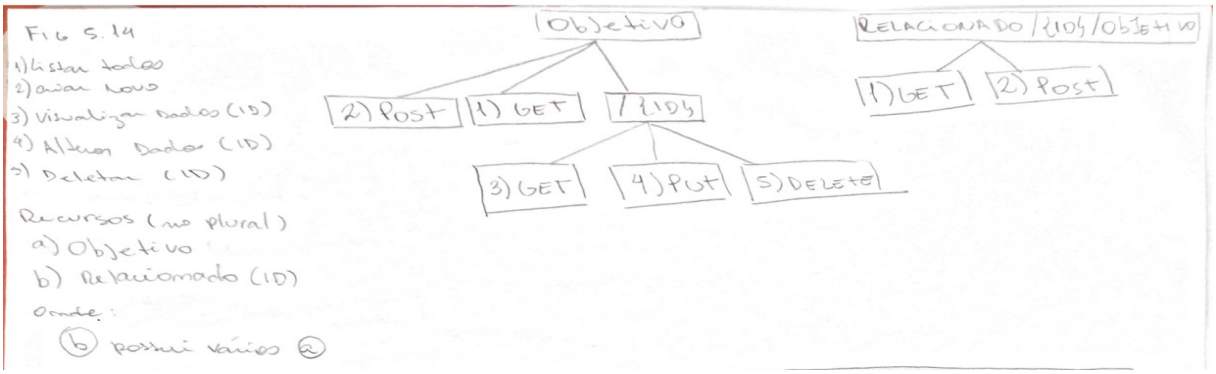


Figura 5.14 - *Site Map* de *Endpoints* genérico

Figura 5.15 - *Site Map* de *Endpoints* do *Sprint* 3

Atividades

Com o Projeto para esta *sprint* concluído, restava planejar as atividades e estimativas a serem executadas. O grande diferencial pras outras *sprints* foi a implementação do relacionamento entre Pacientes e Rotinas, o que provavelmente tomaria mais tempo que a mera implementação de *endpoints*, como vinha sido feito até então. Sem nenhuma referência adequada para estimar a duração desta atividade, pesquisou-se na documentação como ela deveria ser executada e se identificou que também bastava alguma configuração, com poucas linhas de código. Dado que a referência da expectativa de tempo seria o tempo registrado para implementar o *endpoint* da operação "*Read All"* para o recurso Paciente, a implementação do *endpoint* do relacionamento entre Paciente e Rotina foi estimada em duas horas, o dobro de tempo, utilizando o cálculo já mencionado. A Tabela 5.14 apresenta as atividades para esta *Sprint*.

#	Título	Descrição	Estimativa (hrs)
1	**API *Backend***	**CRUD *Restful* em PHP**	**9**
1.1	Criação do Banco de Dados		1
1.2	*CREATE*	Criação de Rotinas via *POST*	1
1.3	*READ all*	Visualização de todas as Rotinas	1
1.4	*READ one*	Visualização de uma Rotina	1
1.5	*UPDATE*	Alteração de Rotina	1
1.6	*DELETE*	Arquivamento de Rotina	1
1.7	GET rotinas de paciente	Listar todas as Rotinas de um Paciente específico, pela sua ID	2
1.8	Verificação Manual	Teste HTTP utilizando a ferramenta *Postman*	1
2	**Store *Routines***	**Store Vuex com métodos de acesso aos *Endpoints* criados anteriormente**	**2**
3	**Páginas *Frontend***		**12**
3.1	Listagem	Visualização de todos os Pacientes	5

3.2	Formulário	Criação, visualização e alteração de Pacientes	7
3.3	Verificação Manual		1

Tabela 5.14 - Atividades para *Sprint* 3

5.7.2 Desenvolvimento

O desenvolvimento do *backend* foi rápido e ocorreu abaixo do tempo estimado, em um único dia de trabalho, porém ainda na abordagem *bottom-up* "em largura", onde todo o *backend* é implementado de uma só vez, antes de iniciar o desenvolvimento do *frontend*. O seu versionamento ocorreu em uma única ramificação, evitando ramos muito curtos e ramos que ultrapassassem a duração de um dia de trabalho, assim como planejado na *sprint* anterior. O *backend* para acesso às Rotinas é muito similar ao de Pacientes, implementado na *Sprint* 1, apenas diferenciando por possuir uma rota adicional, relacionando ambos os recursos, como demonstrado no *Site Map* de *endpoints* (Figura 5.15). Tanto a construção da *Store*, quanto as páginas de Visualização e Edição de dados, são também bastante similares às equivalentes na *Sprint* 1. Por estas razões, as estimativas foram adequadas e todo o desenvolvimento foi completado no prazo e sem maiores dificuldades.

5.7.3 Verificação

Os diagramas utilizados na prática de Modelagem foram bem sucedidos na identificação de métodos, componentes e relacionamentos entre componentes. Os Diagramas de Sequência e WAE, porém, são definidos na UML focando em estabelecer uma comunicação clara e bem definida dentro de uma equipe e por esta razão exigem maiores cuidados em sua elaboração. Além disso, o Diagrama de Sequência unificado acabou sendo demasiadamente comprido, o que atrapalhou a sua produção e posterior leitura. Para ambos se experimentou diagramação digital, e apesar da produção de diagramas digitais facilitar a leitura, o que é ideal para uma comunicação eficaz, não são a forma mais eficiente de se projetar sozinho. Outros diagramas devem ser praticados na busca por uma modelagem mais adequada a um indivíduo.

O *Site Map* de *frontend* foi um guia eficaz quanto à organização de arquivos do projeto no formato *CBA* (*Component Based Architecture*), onde o código fonte é estruturado numa árvore de componentes. Tal estrutura faz bastante sentido e facilita a interpretação do código, por criar uma correspondência lógica entre código fonte e componentes do sistema. Erros no planejamento de tal arquitetura costumam causar grandes transtornos no momento de qualquer readequação que altere o caminho físico de algum arquivo, quando todas as referências a ele em outras partes do código serão quebradas. Esta dificuldade se dá pela possibilidade da árvore crescer e se aprofundar, criando referências a arquivos localizados em caminhos profundos. Numa arquitetura tradicional, como a MVC, os arquivos se encontram em níveis pouco profundos e é muito mais fácil se realizar alterações em seus caminhos. Este *Site Map* também permitiu identificar necessidades de navegação e planejar caminhos entre as páginas. Nesta *sprint*, por exemplo, foi identificada a possibilidade do redirecionamento automático da página de criar uma nova rotina para a página de edição da nova Rotina criada, de forma que assim que os campos obrigatórios fossem preenchidos, a página seria encaminhada discretamente para a página de edição, que é visualmente igual, mas ativa operações de edição e não de criação. Também foi tomada a decisão de ao arquivar uma Rotina, redirecionar o usuário para a lista de Rotinas não arquivadas.

O *Site Map* de *endpoints* também foi importante para projetar a api de dados, de forma que o acesso a seus recursos fosse simples e intuitivo. A medida que a api cresce, um bom planejamento se torna quase imprescindível. As URLs de acesso aos recursos devem ser consistentes para manter a qualidade no serviço de dados, em especial caso tais dados venham a ser utilizados em aplicações de terceiros (TEA, 2017). Outra razão para utilizar um *Site Map* de *endpoints* é documentar a interface utilizada entre ambos *tiers*, onde o *frontend* consome dados do *backend*. Sem esse registro, seria sempre necessário acessar o projeto de *backend* e observar no código fonte quais *endpoints* correspondem a que recursos e ações.

A Tabela 5.15 trás um resumo da avaliação da prática Modelagem nesta *sprint*.

Prática	Está Adequado?
Modelagem 1. Diagrama WAE 2. Diagrama de Sequência *3. Site Map*	Sim. Ambos os diagramas se mostraram bastante úteis no projeto de desenvolvimento. Com elas foi possível planejar a estrutura de módulos e o relacionamento entre eles. Deverão ser ainda mais eficazes em *Sprints* mais complexos. Apesar disso, foi identificada a necessidade de adicionar um diagrama que represente o percurso a ser tomado pelo usuário para atingir determinadas páginas. Sem ele, o relacionamento entre páginas ficaria indefinido, sendo decidido durante o desenvolvimento. Dessa forma, poderia resultar em páginas de difícil acesso ou de acesso improvisado, com decisões de interface gráfica de última hora. Além dos caminhos no *frontend*, surgiu também a necessidade de descrever e padronizar a configuração da interface da API de dados no *backend*. Essa padronização mantém o projeto de *backend* estável e contínuo, dado que o uso de *frameworks* reduz a necessidade de programação em favor da configuração.

Tabela 5.15 - Avaliação das práticas no *sprint* 3

5.7.4. Adequação

Como experimentado durante o desenvolvimento e avaliado na seção anterior, nesta *sprint* foi adicionada um novo diagrama à prática Modelagem, de *Site Map*. Com *Site Maps* deve ser possível identificar caminhos no *frontend* e determinar a interface de *endpoints* no *backend*. Para serem considerados adequados, *Site Maps* devem ser capazes de representar em alto nível a distribuição de páginas e a acessibilidade entre elas, além de documentar a interface do serviço de dados no *tier* de *backend*. A Tabela 5.16 adiciona *Site Map* à lista de diagramas da prática modelagem.

Prática	Critérios
Modelagem *1. Site Map*	É possível identificar páginas e determinar o melhor caminho entre elas, de forma a simplificar seu acesso. Também deve se representar o estado corrente da aplicação, caso ele afete o fluxo de páginas. Na modelagem do *backend* deve registrar a interface de métodos em cada *endpoint*, com

	possíveis parâmetros extra.

Tabela 5.16 - Adequação das práticas após *sprint* 3

5.8 Sprint 4 - Visualização

5.8.1 Análise

Como próxima funcionalidade a ser implementada, a PO decidiu finalmente por uma das mais importantes entregas do projeto, "Visualizar Atividades". A seção Funcionalidades trás esse processo de decisão, bem como a extração de Casos de Uso mais específicos para esse funcionalidade, que vão além das operações CRUD que haviam sido implementadas até então para os recursos de outras *sprints*.

A partir dos Casos de Uso extraídos da funcionalidade escolhida, tentou-se elaborar um Diagrama de Sequência, como feito em *sprints* anteriores. A maior complexidade das funcionalidades presentes na visualização da tela para esta *sprint* impossibilitou a identificação completa do fluxo de dados sem antes haver algum projeto destas funcionalidades. Anteriormente, *wireframes* haviam sido construídos após toda a modelagem do Diagrama de Sequência e WAE. Desta vez se fez necessário construir o *Site Map* primeiro, para identificar quais páginas seriam relevantes e em seguida *wireframes*, para identificar que componentes do sistema seriam necessários em cada página. Ambos estão presentes na seção Modelagem.

Com os *wireframes*, identificou-se a necessidade de um outro módulo para efetuar os cálculos necessários para a construção da visualização das atividades no formato desejado. O Diagrama de Sequência foi construído incluindo este módulo. Com tal diagrama, foi possível determinar para os três Casos de Uso desta *sprint* os componentes envolvidos, seus métodos e os dados retornados. Utilizando o fluxo de dados encontrado no Diagrama de Sequência, foi possível identificar os componentes envolvidos em cada operação e representá-los utilizando o diagrama WAE, além de extrair funcionalidades genéricas dos componentes para possíveis reutilizações futuras. Detalhes da modelagem, bem como os próprios diagramas, na seção Modelagem. Com o projeto da *sprint* pronto, foi possível quebrar as funcionalidades em atividades de desenvolvimento estimáveis. Detalhes na seção Atividades.

Funcionalidades

Com Pacientes e Rotinas já podendo ser cadastradas, os próximos UCs a serem implementados, segundo o "Diagrama de Épicos", eram "Compartilhar Formulário de Preenchimento de Atividades", "Cadastrar Médicos" e "Cadastrar Cuidadores". O cadastro de Médicos e Cuidadores não era urgente para entregar um produto de valor mínimo e após dois *sprints* focados em cadastros, a PO gostaria de um entendimento maior do quão próximo o produto estava do resultado final. Visando entregar valor mais rapidamente, foi decidido como próximo UC "Visualizar Atividades". Foi necessário alertá-la, porém, que como ainda não havia cadastro de atividades, seria necessário criar Atividades artificiais automaticamente, apenas para validação das funcionalidades desenvolvidas.

Os UCs para esta *sprint* são diferentes dos UCs para as *sprints* anteriores. Desta vez não mais bastou implementar operações CRUD para algum recurso. A visualização de Atividades deveria ser uma solução mais elaborada, voltada para experiência do usuário, onde a informação seria entregue graficamente e não em um formulário textual, diferente do que havia sido feito com os recursos até então. Deveria haveria também a possibilidade de selecionar uma Atividade no próprio gráfico de Atividades, para visualizar e editar dados específicos da mesma. Desta vez sim, um formulário, mas diretamente sobre o gráfico de todas as atividades, sem páginas externas. Os Casos de Uso especificados para esta *sprint* foram: "Visualizar Atividades", "Selecionar Atividade", "Editar Atividade" e "Visualizar Detalhes de Atividades".

Primeiramente, foi necessário um estudo mais aprofundado sobre Visualização de Dados, para identificar qual a melhor abordagem para visualizar Atividades. Um formato muito utilizado e considerado bastante adequado pela comunidade é chamado Gráfico Gantt (PROJECTMANAGER.COM, [2018]), usualmente utilizado para visualizar atividades de processos em CPUs de computadores eletrônicos. A biblioteca Javascript D3 fornece diversos módulos de cálculo para qualquer tipo de escalas utilizadas na construção do Gráfico Gantt.

Modelagem

O *Site Map* de *Frontend* para esta *sprint* é bastante simples, pois se compõe apenas de uma nova página, "Visualização de Atividades", ligada a duas páginas da *sprint* anterior. Todos os Casos de Uso do usuário do sistema referentes às Atividades se concentram em uma única página. Nela estará presente a lista de Atividades no formato descrito na seção anterior e também um formulário de edição de dados de Atividade. A criação de Atividades, porém, não faz parte dos Casos de Uso do usuário. Este Caso de Uso é parte do Caso de Uso "Preencher Formulário de Rotina" do ator *Cuidador,* parte do épico *Preenchimento*. O *Site Map* de *Endpoints* para Atividades segue o *Site Map* de *Endpoints* para recursos genéricos definido na *sprint* anterior, inclusive o seu relacionamento com o recurso Rotina.

O *wireframe* da página "Visualização de Atividades" (Figura 5.16) revelou que diversas informações importantes podem ser exibidas em um gráfico Gantt, sem poluir a visualização ou atrapalhar a compreensão do usuário. O gráfico Gantt é costumeiramente utilizado para representar informações sobre eventos ao longo de um eixo temporal. Nele, há retângulos dispostos sobre um quadrante cartesiano e o comprimento, altura, cor e posição destes retângulos representam a informação desejada. Neste projeto, o gráfico é capaz de exibir as seguintes informações a respeito das Atividades:

1. Duração, com o comprimento
2. Momento, com o posicionamento horizontal
3. Categoria, com o posicionamento vertical
4. Independência do paciente ao realizar a Atividade, com a cor

Lista 5.4 - Informações Visuais da Visualização de Atividades

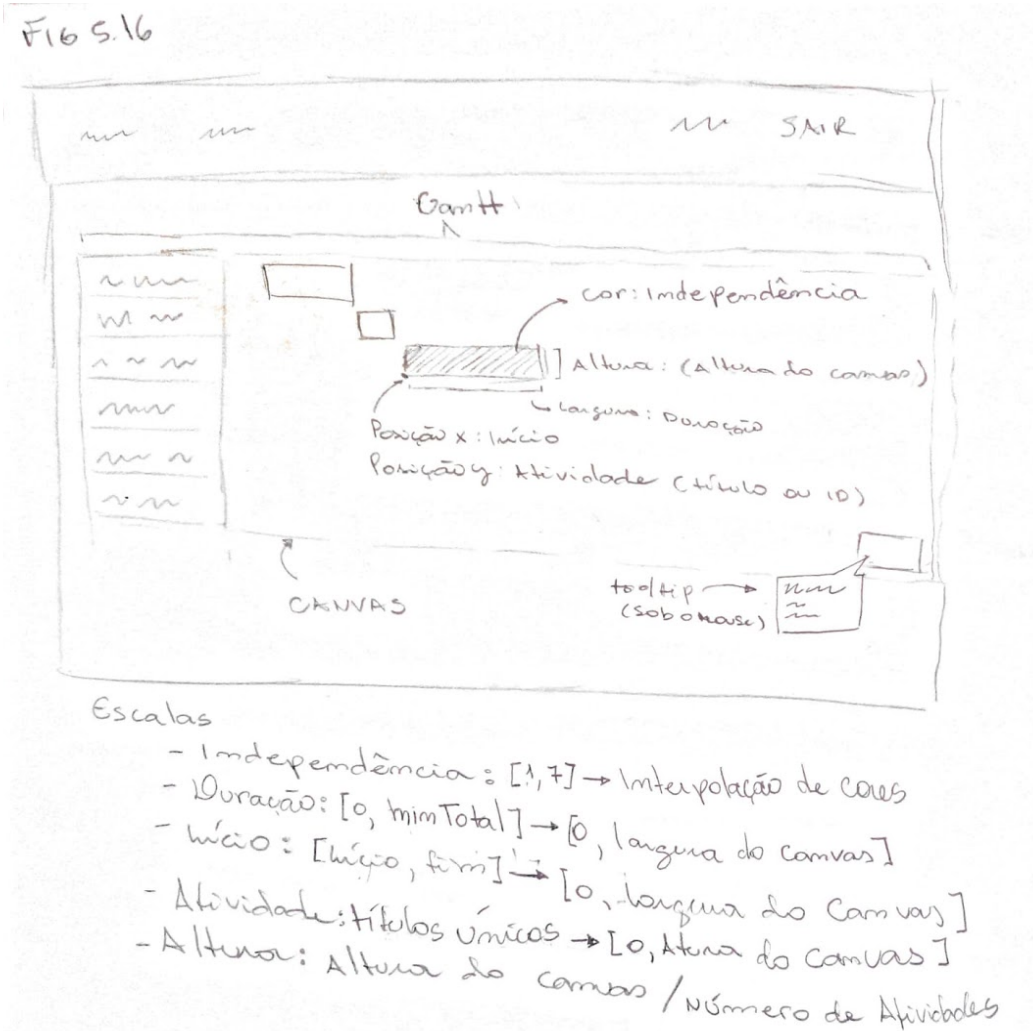


Figura 5.16 - Wireframe de "Visualização de Atividades"

O *wireframe* revelou que funcionalidades estariam disponíveis ao usuário através da interface gráfica. Um evento é disparado assim que o usuário acessa a página. A partir daí, uma sequência de carregamento de dados da Rotina e todas as Atividades cadastradas, acionado através da Store e, após o carregamento, a geração da interface gráfica de cada Atividade, que são os retângulos. Para cada um destes retângulos, haveria cálculos utilizando diferentes escalas, convertendo dados em informações visuais, conforme a Lista 5.4. A ação de "Selecionar Atividades" se inicia quando o usuário clica num retângulo, que por sua vez registra a seleção no estado interno da Atividade, além de alterar o conjunto de Atividades selecionadas no estado da página. Se o conjunto de Atividades selecionadas contiver apenas uma Atividade, passa a estar

disponível a ação do Caso de Uso "Editar Atividade", surgindo um formulário para possibilitar a sua edição. O Diagrama de Sequência da Figura 5.17 demonstra os métodos e eventos invocados nos três Casos de Uso.

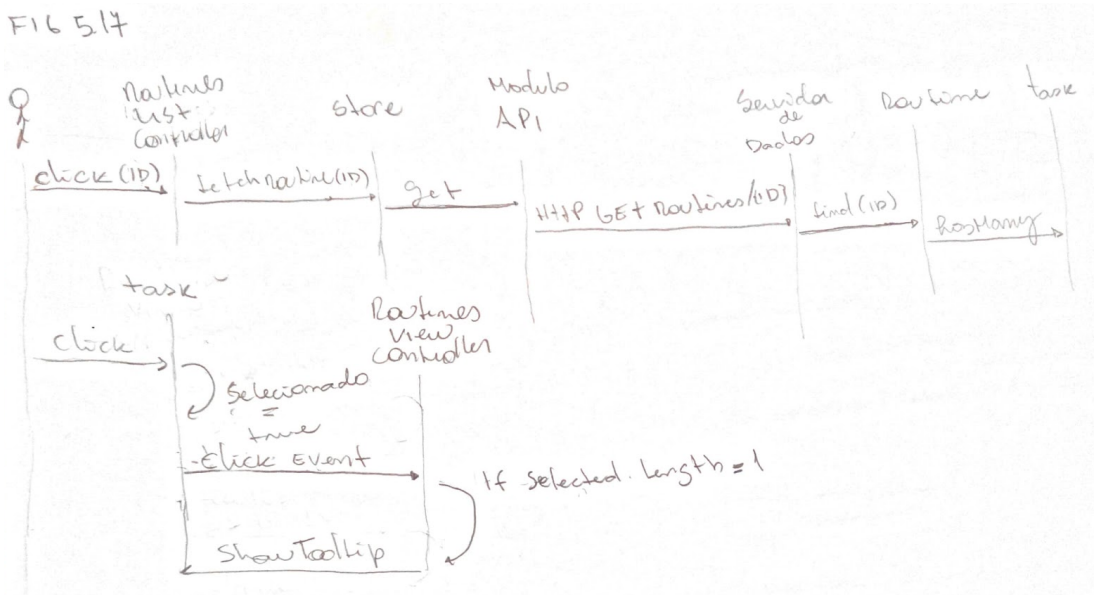


Figura 5.17 - Diagrama de Sequência para a *sprint* 4

O *framework* VueJS permite herança múltipla de componentes, também chamados de *mixins* no contexto de herança múltipla. Vários *mixins* podem ser utilizados por um único componente, que resultará numa mescla de todos eles. Fazendo uso desta possibilidade, o Diagrama WAE se inicia no componente "Página de Rotinas", que herda do *mixin RoutineModel*, com todos os dados e métodos referentes ao modelo de recurso Rotina, e do *mixin Stats*, que traz métodos e dados referentes a estatísticas de um conjunto de Atividades. *Stats*, por sua vez, herda do *mixin* mais genérico *Collection*, que representa uma coleção de objetos. Dessa forma, a página passa a ter a capacidade de agrupar Atividades, calcular estatísticas a seu respeito e armazenar dados da Rotina. O componente "Página de Rotinas" possui diversos componentes do tipo *Task* (Atividade), existindo então um relacionamento de agregação entre ambos. *Task* estende um *mixin TaskModel*, com todos os dados quanto à Atividade, e dois referentes à interatividade com o usuário, *TaskClickHandler* e

TaskHoverHandler, para cliques e deslizes do mouse. Existe também uma associação entre o componente da página e o módulo *Store*, que por sua vez está associado com o módulo API. A Figura 5.18 demonstra o Diagrama WAE para estes componentes. Para construí-lo, foi necessário estabelecer com a *PO* que informações seriam necessárias no recurso Atividade.

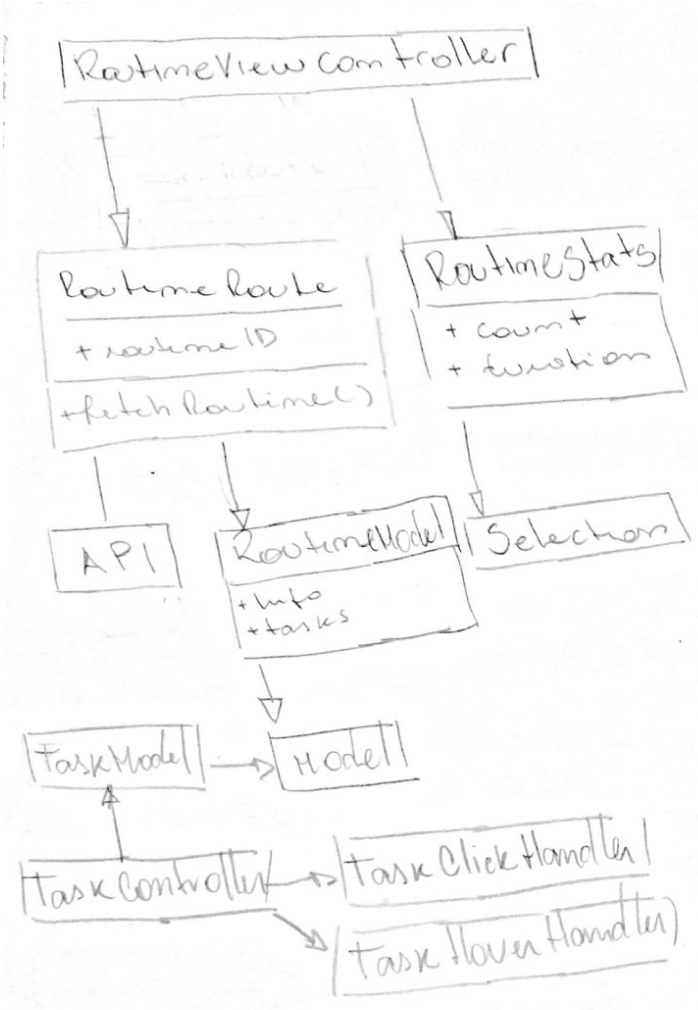


Figura 5.18 - Diagrama WAE para a *sprint* 4

Atividades

Nesta *sprint,* não houve necessidade das operações CRUD habituais no *backend* para o recurso Atividade. Desta vez, seria necessário apenas uma operação de "Visualização de Atividades de uma Rotina", fruto do relacionamento um-para-muitos

entre Rotinas e Atividades. Como decidido na *sprint* anterior, foi criada uma atividade específica para a validação manual, de modo a contabilizar a sua estimativa de tempo na estimativa total da *sprint*. A Store seria similar às outras já criadas e por isso, sua estimativa também foi igual. Para o novo módulo criado, *Scales*, não havia nenhuma referência. Baseado na documentação encontrada a respeito da biblioteca D3, reservou-se um dia inteiro para a construção do módulo que calcularia a conversão de dados de Atividades em dados visuais, descrita na Lista 5.4. A construção da página se deu por etapas, onde se criou uma atividade para cada componente da visualização (eixos, Atividades, *Tooltip*), ou interação com o usuário. A Tabela 5.17 trás as atividades selecionadas.

#	Título	Descrição	Estimativa (hrs)
1	**API *Backend***	**CRUD *Restful* em PHP**	**3**
1.1	Criação do Banco de Dados	Tabela "Tasks"	1
1.2	GET Atividades de Rotina	Listar todas as Atividades de uma Rotina específica, pela sua ID	1
1.3	Verificação Manual	Teste HTTP utilizando a ferramenta *Postman*	1
2	**Store *Routines***		**2**
3	**Módulo *Scales***	**Para cálculos de escala utilizados na visualização**	**6**
4	**Página de Visualização**		**15**
3.1	Gerar eixos	Eixos de Título e tempo	4
3.1	Gerar Atividades	Gerar Retângulos utilizando o módulo Scales criado anteriormente	4
3.2	Tooltip informativo	Caixa de texto informativa, quando mouse está sobre atividade	3
3.3	Seleção	Seleção múltipla de Atividades	4
3.4	Edição de Tooltip	Edição de dados de Atividade através de formulário no tooltip, quando apenas uma Atividade está selecionada	4

Tabela 5.17 - Atividades para *Sprint* 4

5.8.2 Desenvolvimento

O *backend* foi desenvolvido exatamente como previsto, sem qualquer novo obstáculo. A verificação manual foi rápida e continuou se mostrando uma boa opção, dado que neste experimento o sistema de dados, que constitui o *backend*, possui uma saída de dados bastante simples e previsível. Os métodos da *Store* foram poucos e bastante simples, desta vez sem o uso do TDD, no intuito de simplificar o desenvolvimento. Desta forma, o seu desenvolvimento foi mais rápido que o estimado.

Devido a sua natureza mais voltada para uma interface gráfica avançada, com elementos especiais, para além de simples formulários, nesta *sprint* o desenvolvimento do *frontend* se demonstrou complexo, com diversas reformulações na estrutura das páginas. No ciclo de vida de qualquer página, primeiramente ela é carregada sem conter todos os dados, o que impacta na sua visualização inicial. Em seguida, com os novos dados carregados, a visualização se reestrutura para acomodá-los. É necessário haver no projeto um processo que leve em consideração a dinâmica de dados e sua influência na interface gráfica. Além disso, os Wireframes das páginas em alto nível não detalham em nada os componentes utilizados na sua construção, o que resulta em muito improviso durante a fase de desenvolvimento. Somente a modelagem de alto nível como a desenvolvida durante a fase de Análise não bastou para alcançar tal entendimento. Desperdiçou-se bastante tempo reestruturando a página de forma a corrigir defeitos não previstos durante o projeto de interface gráfica.

5.8.3 Verificação

Nesta *sprint* houve uma demanda clara por maior especificidade da fase de Modelagem. O Diagrama de Sequência nada informou sobre o momento de entrega de dados à interface gráfica. Sem um planejamento prévio que entregasse esse conhecimento, se perdeu várias horas de desenvolvimento. As capacidades do Diagrama WAE com foco na Web não foram usadas até então, apenas recebendo atenção nas suas capacidades de representar classes e componentes estáticos. A falta

de um estudo geral, produzindo um Diagrama de Classes para o sistema como um todo, impede a identificação de reuso de funcionalidades. Nesta *sprint* se buscou identificar possibilidades de reuso em um escopo muito pequeno, produzindo componentes genéricos que podem nem ser utilizados. A prática "Design Incremental" diz que apenas devem ser implementadas as funcionalidades absolutamente necessárias, então um Diagrama WAE ou de Classes dentro de uma *sprint* na maioria dos casos não entrega todas as vantagens que devem ser atribuídas ao seu uso. O comportamento dinâmico está sendo representado em Diagramas de Sequência e *Site Maps*. Até esta *sprint* não se criou *wireframes* para projetar elementos de interface específicos, apenas para sugerir funcionalidades de páginas. Por esta razão a todo o momento estavam sendo tomadas decisões improvisadas, durante o desenvolvimento. Muitas dessas decisões precisaram ser desfeitas e repensadas, atrapalhando a criação de valor para o cliente, que deve ser a prioridade. A Tabela 5.18 trás as avaliações das práticas mencionadas.

Prática	Está Adequado?
Modelagem 1. *Wireframes* 2. Diagrama de Sequência	Não. Ambos os diagramas se mostraram insuficientes por serem pouco específicos. *Wireframes* foram utilizados primeiramente, logo após a definição de Casos de Uso, diferente das outras *sprints*.

Tabela 5.18 - Avaliação das práticas no *sprint* 3

5.8.4 Adequação

O Diagrama de Sequência deveria aprofundar mais nas funcionalidades encontradas no Wireframe, de modo a incluir os processos que calculam e entregam dados à interface gráfica. Dessa forma é possível compreender o estado da interface seguindo o fluxo de dados. Um uso relevante para um Diagrama de Sequência mais completo é a análise da dependência de partes da interface gráfica sobre outras partes. Nesta *sprint*, por exemplo, todos os dados relativos às dimensões dos retângulos no Gráfico Gantt dependia das dimensões que os títulos das atividades ocupariam na tela.

Apenas após preencher o espaço reservado a esses títulos seria possível calcular as dimensões disponíveis para a criação de retângulos. No caso de interfaces gráficas mais complexas, devem ser construídos *wireframes* específicos para alguns de seus componentes e para os diversos estados nos quais eles podem existir.

Com essa adequação, os Diagramas de Sequência e Wireframes irão aumentar em complexidade. Dado que já estão sendo utilizados diversos diagramas: Diagramas WAE, *Site Maps*, Diagramas de Sequência e *Wireframes*, um novo tipo de diagramação será utilizada na *sprint* seguinte, que engloba os objetivos de *Wireframes*, *Site Maps* e Diagramas WAE, simplificando o processo de Modelagem. Esta diagramação é pouco burocrática e é utilizada e recomendada por Maximilian Schwarzmüller, professor e criador da Academind (www.academind.com). Apesar de não haver um nome definido, neste trabalho o diagrama será chamado de "Mapa de *Wireframes*", com nome e metodologia inspirados no "Mapa de Telas" definido no Guia para Desenvolvedores Android (ANDROID OPEN SOURCE PROJECT, 2018). Detalhes na Tabela 5.19.

Assim como foi feito nesta *sprint* a modelagem do Mapa de *Wireframes* deve ser iniciada na produção de *wireframes* para as páginas, primeiramente construindo um *wireframe* para um *template* base, também chamado de *frame*, que poderia, por exemplo, conter um menu superior e talvez um rodapé, ou *sidebar*. Como *template* base, esse *wireframe* não representa nenhuma página, mas apenas um quadro em torno do conteúdo de cada página. Por esta razão, nele deve haver um espaço para o conteúdo específico de cada página. Em seguida, deve se criar os *wireframes* de cada página seguindo os *hiperlinks* a partir da página inicial. Caso uma página tenha mais de um estado relevante, deve se representar no mapa. Qualquer componente de GUI complexo, que exija maiores detalhamentos, deve ser extraído para um *wireframe* próprio. Todo *wireframe* que representar uma página deve explicitar uma URL. Deve existir setas entre elementos de interface que alterem o *wireframe* e o *wireframe* final. O mesmo vale para conexões com a api de dados disparadas por componentes da interface, representadas por setas tracejadas terminadas em um círculo pequeno, que representa o servidor de dados, e serem rotuladas com método HTTP e *endpoint*.

Conexões disparadas automaticamente pela página devem ser provenientes das arestas do *wireframe*. Eventos ocorrendo em sequência podem ser numerados.

Exemplificando o esquema descrito acima, a página "Edição de Paciente", com URL "/paciente/{id}", possui um *wireframe* com um formulário no centro. Uma conexão com o servidor de dados sai da aresta com legenda "GET /patients/{ID}". Do formulário sai uma conexão com o servidor de dados, com legenda "PUT /patients/{ID}" e de um botão na base do página uma seta até o *wireframe* da página "Lista de Pacientes". A Figura 5.19 ilustra o exemplo dado.

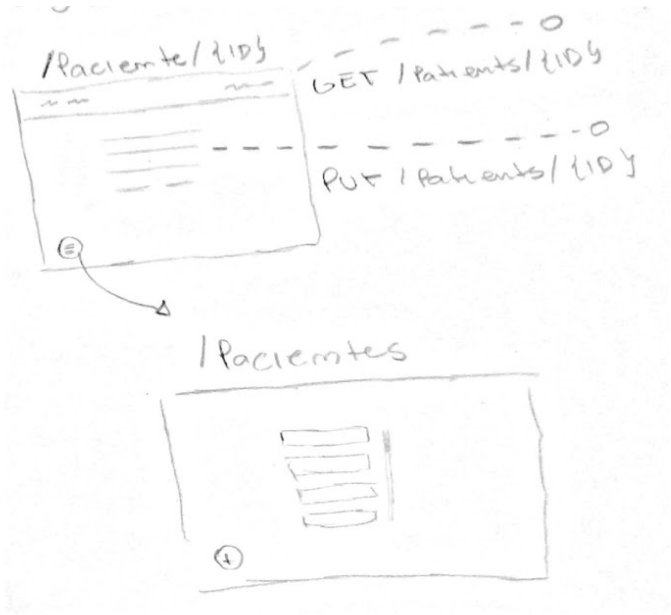


Figura 5.19 - Exemplo de Mapa de *Wireframes*

Prática	Critérios
Modelagem *1.* Mapa de *Wireframes* 2. Diagrama de Sequência	O Mapa de *Wireframes* deve substituir diversos outros diagramas. Diagramas de Sequência devem ser mais específicos quanto a entrega de dados à interface gráfica, visando determinar em que momento e com quais informações os componentes serão renderizados.

Tabela 5.19 - Adequações da *sprint* 4

5.9 Sprint 5 - Preenchimento

5.9.1 Análise

No projeto sendo desenvolvido existem dois Casos de Uso principais identificados pelas clientes. Um deles foi entregue funcional na *sprint* anterior. Nesta *sprint*, foram escolhidos os Casos de Uso remanescentes para entrega de um *MVP*, incluindo o segundo Caso de Uso principal e excluindo casos de uso suplementares, que não são vitais para o funcionamento básico da Aplicação.

A análise dos Casos de Uso se iniciou pelo seu detalhamento em Casos de Uso mais específicos, como já havia sido feito anteriormente e então um diagrama não utilizado até o momento, de Entidade-Relacionamento. Essa novidade surgiu da necessidade de representar novos relacionamentos entre recursos, que além de serem mais complexos, iriam desfazer relacionamentos já estabelecidos, por falta de um planejamento global, que deveria ter ocorrido inicialmente. O Diagrama de Entidade-Relacionamento identificou relacionamentos entre recursos, permitindo uma compreensão geral da estrutura de dados e das operações no *backend* e no *frontend*.

O processo de modelagem concentrou diversas utilidades diferentes no Mapa de *Wireframes*. Naturalmente, os *wireframes* identificam as telas e os elementos de interface gráfica em alto nível. Adicionalmente, o Mapa de *Wireframes* também incluiu os endereços localizadores de cada página (URL), e que informação é necessária que estes endereços contenham, como número de identificação do recurso sendo utilizado na página. Quanto ao *backend*, cada *wireframe* também trazia as conexões com os *endpoints* da api de dados que seriam efetuadas em cada página, com setas tracejadas partindo do elemento da interface que ativaria a conexão, terminando em um círculo, que representa o servidor de dados, e rotuladas com o endereço do *endpoint*. Por terem funcionalidades simples e já trabalhadas anteriormente, não foi necessário descrever a maioria dos componentes gráficos utilizados nos *wireframes* em um *wireframe* específico e detalhado. Apenas um *wireframe* extra de alta fidelidade foi desenvolvido.

Buscando atingir uma aplicação funcional mais rapidamente, nesta *sprint* se decidiu alterar o paradigma de determinação de atividades. No lugar de se percorrer a pilha de tecnologias *em largura*, foi decidido percorrê-la *em profundidade*, levando ao extremo a prática *Design Incremental*. Implementar a pilha *em largura* significa implementar todo o *backend* primeiro, criando recursos no banco de dados e todos os métodos da api de dados, para posteriormente criar toda a comunicação entre *frontend* e *backend* e apenas então criar todas as páginas. Implementá-la *em profundidade* significa implementar uma funcionalidade por vez em toda a pilha. Primeiramente deve se criar todos os recursos no banco de dados, por não ser prático quebrar tabelas relacionais em partes. Em bancos de dados não relacionais, pode ser possível quebrar a criação de recursos em partes menores, de forma que ela possa ser implementada incrementalmente com as funcionalidades relevantes. Após ter o banco de dados pronto, apenas serão implementados o método relevante da api de dados, sua interface com o *frontend* e os componentes necessários na página para este caso de uso. A seleção de atividades para esta *sprint* está descrita na seção Atividades.

Funcionalidades

Com a visualização de Atividades já funcional, os Casos de Uso de maior valor pertenciam ao Épico Preenchimento, "Compartilhar Formulário de Preenchimento" e "Preencher Formulário de Rotina" dos atores TO e Cuidador, respectivamente. Estes foram os Casos de Uso escolhidos para esta *sprint*. Foi decidido junto à *PO* que os demais Casos de Uso não faziam parte do *MVP* para o problema proposto e, dessa forma, esta seria a última *sprint* para a conclusão da primeira versão da aplicação.

Por não possuir cadastro no sistema e nem existir um sistema de autenticação para Cuidadores, os Casos de Uso do ator Cuidador não são protegidos por autenticação. Por essa razão exige-se maior cuidado com que informações são disponíveis para eles, publicamente. Para melhor entender os Casos de Uso, foram listados junto a *PO* características a respeito dos recursos tratados nesta *sprint* e suas relações:

1. Formulários possuem Preenchedor (Cuidador) e lista de Atividades
2. Formulários possuem inicialmente apenas o nome do Preenchedor
    a. Posteriormente poderão ser atrelados à Cuidadores cadastrados no sistema
3. É possível criar Formulários ilimitados para cada Rotina
4. Atividades são atreladas à Formulários de forma que possam ser tratadas como um conjunto.
5. É possível ativar e desativar Formulários e assim alternar a visualização de suas Atividades.
6. Formulários possuem uma URL compartilhável.
7. Formulários devem ser de preenchimento bastante simples.

Lista 5.5 - Características para *sprint* 5

Onde anteriormente uma Rotina possuía diversas Atividades, o novo recurso Formulário pertence a uma Rotina e possui diversas Atividades. As operações *CRUD* do recurso Formulário são do ator TO, autenticadas, sendo então privadas e por esta razão são iguais às de outros recursos anteriormente implementados. Apesar do Caso de Uso "Preencher Formulário de Rotina" pertencer ao ator Cuidador, o Caso de Uso "Cadastrar Cuidador" ainda não foi implementado. Dado que para a implementação do *MVP* decidiu-se que esse Caso de Uso não seria implementado, toda a funcionalidade foi criada para atender ao caso do preenchedor ser um Cuidador não cadastrado, apenas utilizando o seu nome. A simplicidade do preenchimento dos Formulários, do item 7 da Lista 5.5, será tratada na seção Modelagem. Os Casos de Uso para esta *sprint* são *"*Criar Formulário", "Compartilhar Formulário", "Listar Formulários", "Ativar/Desativar Formulário" e "Apagar Formulário", para o ator TO e "Visualizar Dados Públicos do Formulário", "Cadastrar Atividade*", "*Alterar Atividade" e "Apagar Atividade", para o ator Cuidador.

Com as informações obtidas, um Diagrama de Entidade-Relacionamento bastante resumido foi desenvolvido, apenas para estabelecer os novos recursos e suas relações entre recursos do sistema (Figura 5.20). Este diagrama se faz necessário para

estabelecer atributos e relacionamentos entre as entidades de dados e deveria ter sido elaborado na *sprint* 0, de modo a evitar a correção sendo efetuada nesta *sprint*. Aqui, esta etapa foi utilizada com a intenção de reduzir erros ao alterar o código já implementado para o novo modelo de dados e definir que atributos poderiam estar visíveis publicamente. A fim de proteger os dados de usuários mal intencionados, foi criado um novo atributo identificador para recursos disponíveis publicamente. No lugar de utilizar a id numérica tanto no endereço das páginas públicas, quanto nas operações do *backend*, foi utilizado uma longa sequência de caracteres única, chamada UID (*Unique Identifier)*, de modo que não bastasse digitar um número pequeno no endereço, para se ter acesso aberto às atividades de um Paciente. Seria necessário acertar dezenas de caracteres para encontrar um formulário específico.

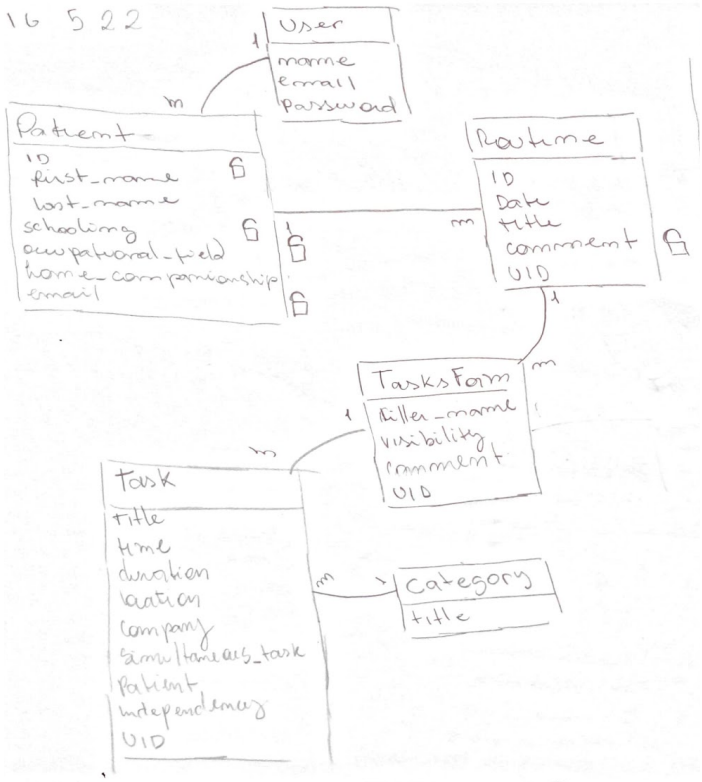


Figura 5.20 - Diagrama de Entidade-Relacionamento

Modelagem

Um Mapa de *Wireframes* foi construido incluindo as páginas:

1. "Visualizar Atividades",
2. "Novo Formulário",
3. "Editar/Compartilhar Formulário"
4. "Formulário de Atividades".

Lista 5.6 - Páginas do Mapa de *Wireframes* da *sprint* 5

A última foi dividida em dois *Wireframes*, que representam dois estados diferentes da página, Lista (a) e Formulário (b). Os *wireframes* estão conectados entre si quando há elementos na interface gráfica, ou eventos, que levem o usuário de uma página a outra. Em tracejado há conexões com o servidor de dados, que podem ser disparadas automaticamente pela página, quando a seta parte da aresta do *wireframe*, ou por alguma ação do usuário, quando a seta parte de algum elemento da interface gráfica. Todos os *wireframes* foram identificados pelas URLs das páginas que representam. O Mapa de *Wireframes* está representado na Figura 5.21.

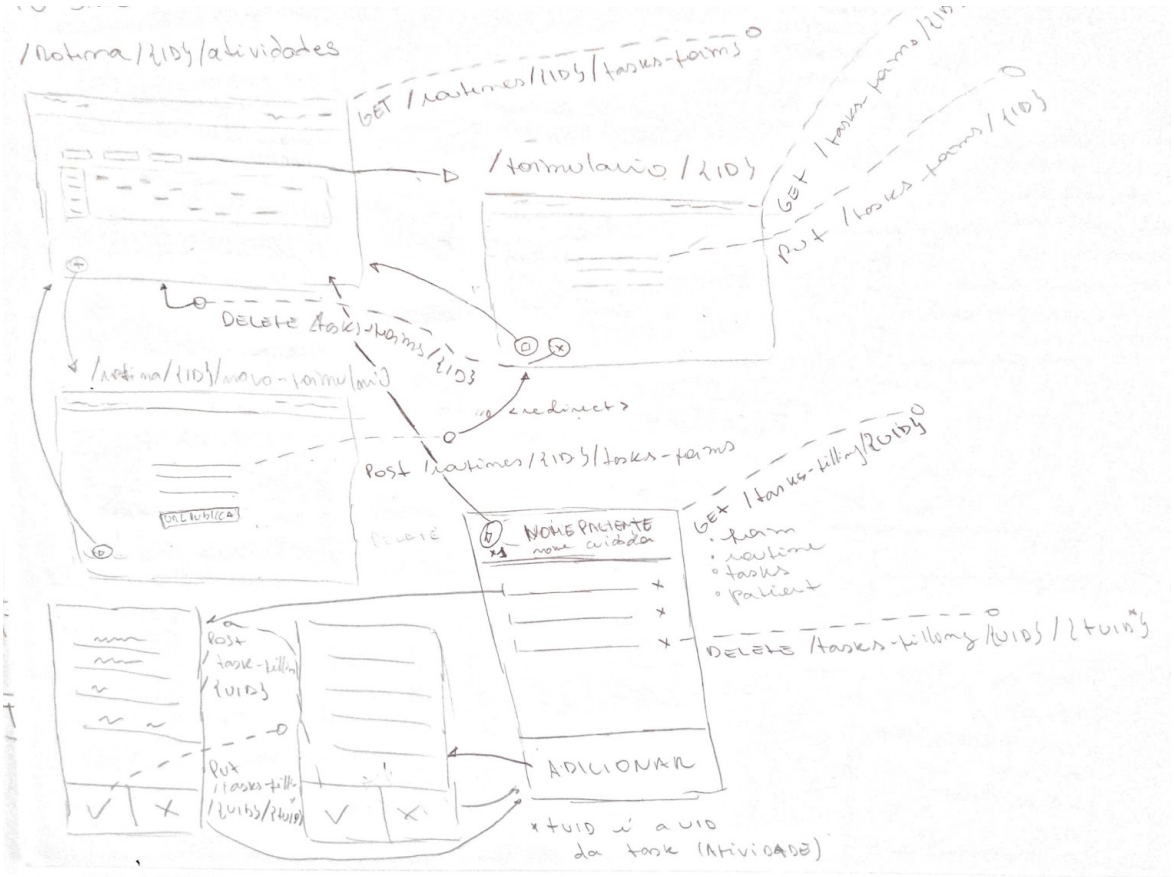


Figura 5.21 - Mapa de Wireframes para a *sprint* 5

Para os objetivos das clientes, é crucial que o preenchimento do formulário seja muito simples, evitando campos de preenchimento de texto. Também é crucial que ele possa ser preenchido no próprio celular, com compartilhamento via chat. Dois *wireframes* mais detalhados foram desenvolvidos para este formulário, página 4 da Lista 5.6, um para cada um de seus estados. Ambos os *wireframes* estão na Figura 5.22.

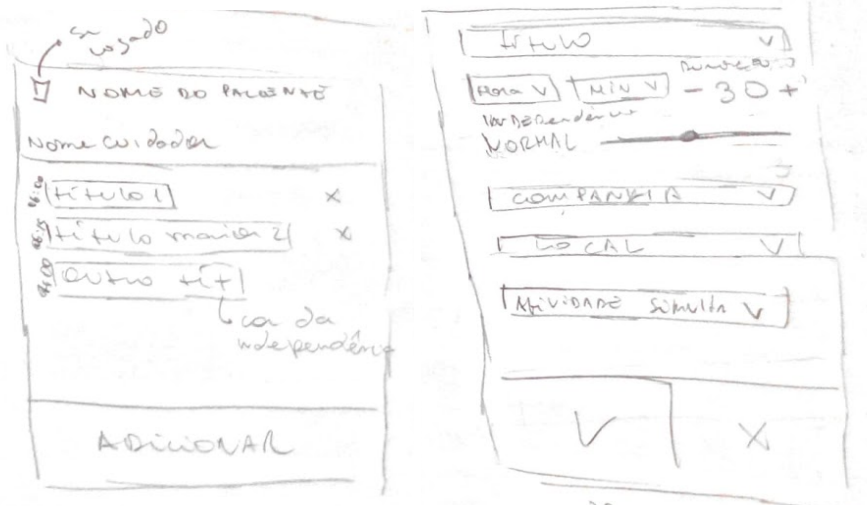


Figura 5.22 - Wireframes para a página "Formulário de Atividades"

Com o Mapa de *Wireframes* e os *wireframes* de alta fidelidade foi possível definir quais componentes seriam necessários desenvolver para a interface gráfica e quais a funcionalidade de outros componentes poderiam ser reaproveitados. Foi percebida uma ausência de uma visão geral do sistema. Uma prática de modelagem geral do sistema deve ser adicionada à *sprint* 0, com enfoque na identificação de generalizações e similaridades entre páginas e componentes.

Desta vez haveria duas novidades no desenvolvimento do *backend*: A utilização de *endpoints* públicos e de um atributo especial, UID, com funcionalidades comuns a pelo menos dois recursos, Atividade e Formulário. Dessa forma, houve uma pequena modelagem do cenário de uso destas novidades. Para acesso à dados públicos é

necessário um novo controlador, que filtra e modela os dados antes de entregá-los publicamente a página de "Formulário de Atividades", de forma que todos e apenas os dados necessários estariam disponíveis. Além do controlador, os recursos Atividade e Formulário possúem funcionalidades em comum de geração de UID. Para isso, o *Trait HasUid* foi criado. *Traits* são como classes para herança horizontal em PHP. Um diagrama de Classes foi criado, envolvendo as classes *Task*, *TasksForm*, *TasksFillingController* e *HasUid*, modelando as classes necessários (Figura 5.23).

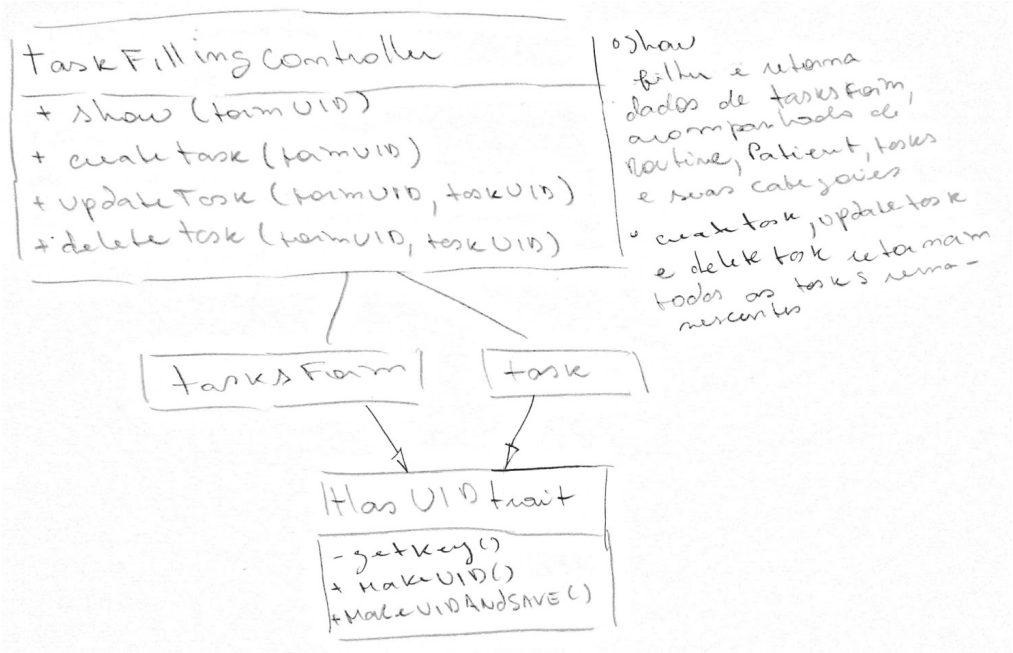


Figura 5.23 - Wireframes para a página "Formulário de Atividades"

Atividades

Nesta *sprint*, para seguir um paradigma mais incremental, onde cada funcionalidade é implementada por completo em todas as camadas da aplicação antes de se partir para outra funcionalidade, as atividades foram divididas por caso de uso e não mais por camadas da aplicação.

Primeiramente a criação de recursos no banco de dados. A tabela de Atividades *task* já estava criada, mas precisaria ser alterada para pertencer a um Formulário e não a uma Rotina. A tabela de Formulários *tasks_forms* deveria ser criada. Todos os Casos

de Uso da TO são procedimentos *CRUD* como os já mencionados ao longo das outras *sprints*. Cada caso de uso possui um método na api do *backend*, um método na api de comunicação e uma interface gráfica que possibilite o seu acionamento.

Apesar de alguns Casos de Uso do Cuidador já haverem sido implementados na outra *Sprint* 4 - Visualização (de Atividades), as operações sobre Atividades na *sprint* atual estariam disponíveis publicamente. Por esta razão, os *endpoints* criados não utilizaram a ID dos recursos, mas sim a UID, atributo já descrito na seção Funcionalidades. O controlador responsável no *backend* pela resposta às operações do Cuidador deve não apenas filtrar a informação entregue, mas também validar UIDs. A tabela 5.19 trás as atividades programadas para esta *sprint*.

#	Título	Descrição	Estimativa (hrs)
0	**Banco de Dados/*Backend***	**1. Alterar tabela Task** **a. Incluir UID** **b. Incluir form_id** **2. Criar tabela Form** **3. Criar classe/trait *HasUid* e aplicá-la à Task e TasksForm**	**2**
1	**Criar Formulário**		**7**
1.1	*Backend*	POST /routines/{id}/tasks-forms	1
1.2	TasksForm Store	*createRoutineTasksForm(id, data)*	0,5
1.3	Página	/rotina/{id}/novo-formulario	5
1.4	*Hiperlink*s	Botões na página de visualização de Atividades e na página de novo Formulário	0,5
2	**Listar Formulários**		**3,5**
2.1	*Backend*	GET /routines/{id}/tasks-forms	1
2.2	TasksForm Store	*fetchTasksForms(id)*	0,5
2.3	Lista de Formulários	Lista de botões de Formulários na página de visualização de Atividades	2
3	**Ativar/Desativar Formulário**		**2,5**
3.1	*Backend*	PUT /tasks-forms/{id}	1

3.2	TasksForm Store	*updateTasksForms(id, data)*	0,5
3.3	Botão de Formulário	Clique no Botão de Formulário transmite alteração no atributo *visible*	1
4	**Apagar Formulário**		**4,5**
4.1	*Backend*	DELETE /tasks-forms/{id}	1
4.2	TasksForm Store	*deleteTasksForms(id)*	0,5
4.3	Botão de Exclusão	Novo botão na página de novo Formulário	3
5	**Compartilhar Formulário**		**0,5**
5.1	Exibir URL	/preenchimento/{formUID}	0,5
6	**Visualizar Formulário**	**Dados públicos de Formulário, Rotina, Atividades e Paciente**	**6,5**
6.1	*Backend*	GET /tasks-filling/{formUID}	2
6.2	TaskFilling Store	*fetchFormData(uid)*	0,5
6.3	Pagina	/preenchimento/{formUID}	4
7	**Cadastrar Atividade**		**3**
6.1	*Backend*	POST /tasks-filling/{formUID}	2
6.2	TaskFilling Store	*createFormTask(uid, data)*	0,5
6.3	Atualizar Lista de Atividades	Atualizar lista com resposta do servidor	0,5
8	**Alterar Atividade**		**3**
6.1	*Backend*	PUT /tasks-filling/{formUID}/{taskUID}	2
6.2	TaskFilling Store	*updateFormTask(formUID, taskUID, data)*	0,5
6.3	Atualizar Lista de Atividades	Atualizar lista com resposta do servidor	0,5
9	**Apagar Atividade**		**3**
6.1	*Backend*	DELETE /tasks-filling/{formUID}/{taskUID}	2
6.2	TaskFilling Store	*deleteFormTask(formUID, taskUID, data)*	0,5
6.3	Atualizar Lista de Atividades	Atualizar lista com resposta do servidor	0,5

Tabela 5.18 - Adequações da *sprint* 4

2. Desenvolvimento

A fase de desenvolvimento para esta *sprint* ocorreu com o maior sucesso dentre todas as *sprints* no projeto de estudo. Nesta *sprint* houve implementações diferentes, ainda não praticadas anteriormente, como telas disponíveis publicamente, com dados filtrados; um atributo de identificação secundário; também a alteração de tabelas já existentes. Também houve uma tela voltada para reprodução no celular, em oposição a todas as outras telas, que foram construídas apenas para serem visualizadas em computadores de tela grande, a pedido da *PO*. Ainda havendo essas novidades, o desenvolvimento correu sem nenhum obstáculo, do início ao fim. Grande parte dos detalhes de interface gráfica foi definido durante o desenvolvimento, sem demasiado planejamento, além dos *wireframes* detalhados da página de Formulário.

3. Verificação

Todos os diagramas utilizados nesta *sprint* foram considerados adequados. O Diagrama WAE foi excluído e Diagramas de Sequência se mostraram desnecessários, sendo substituídos pelo Mapa de Wireframes, que representou toda a Modelagem do *frontend.* Devido a novidades nas funcionalidades do *backend*, houve a necessidade da criação de uma nova classe, que adiciona funcionalidades especiais aos modelos de dados. Um Diagrama de Classe bastante resumido foi utilizado para este fim.

Também foi desenvolvido um Diagrama Entidade-Relacionamento para representar as novas relações entre diferentes recursos de dados. Este Diagrama registra atributos dos recursos e também relacionamentos rotulados com cardinalidade, revelando como modelá-los no banco de dados. A ausência deste diagrama na primeira *sprint* causou um erro de planejamento que precisou ser corrigido nesta *sprint*. Um resumo das avaliações está na Tabela 5.19.

Prática	Está Adequado?
Modelagem 1. Diagrama WAE 2. Diagrama de Sequência 3. Diagrama de Classes	Não. Diagrama WAE e de Sequência se mostraram pouco úteis e foram descartados nesta *sprint*, substituído pelo Mapa de *Wireframes*. Este, por sua vez se demonstrou bastante eficaz.

4. Mapa de *Wireframes* 5. Diagrama Entidade-Relacionamento	Diagramas de Sequência devem ser deixados para funcionalidades mais complexas, talvez podendo ser substituídos por fluxogramas, que são de elaboração mais rápida. Para definições estáticas mais específicas foi utilizado um Diagrama de Classes resumido e um Diagrama Entidade-Relacionamento foi utilizado para modelar dados de forma simples e eficaz.

Tabela 5.19 - Avaliação das práticas no *sprint* 3

6. Adequação

O equívoco revelado ao final da seção anterior, da ausência de planejamento global, acima do planejamento de uma *sprint* individual, poderia causar a necessidade de uma remodelagem impraticável num sistema mais complexo. Por esta razão, e também pelo problema de planejamento de navegação entre páginas de diferentes *sprints*, a fase de análise da *sprint* 0 deve incluir mais procedimentos com enfoque no sistema total, antes de adentrar na elaboração de Casos de Uso isolados.

Alguns Diagramas se demonstraram pouco eficazes na Análise e na Modelagem da maioria das *sprints*, enquanto outros se demonstraram essenciais. Também a ausência de alguns gerou dificuldades. Os diagramas considerados essenciais foram agrupadas como "Primários", são eles: os Diagramas de Casos de Uso, de Entidade-Relacionamento e o Mapa de *Wireframes*. A criação de Diagramas de Classe, de Sequência/Comunicação/Atividades foram classificados como práticas "Secundárias", que podem vir a ser úteis, mas não são essenciais em todos os projetos.

Prática	Critérios
Análise 1. Casos de Uso 2. Entidade-Relacionamento Modelagem 1. Primárias a. Mapa de *Wireframes* 2. Secundárias a. Diagramas de Classe b. Diagramas de Sequência/Comunicação/Atividade	Se um diagrama não entrega valor ao desenvolvedor, não deve ser executado. Um desenvolvedor individual não necessita de registrar suas decisões para terceiros. Dessa forma, apenas se deve diagramar o que for estritamente útil para o desenvolvedor compreender melhor o sistema que está sendo construído e tomar decisões melhor informadas.

5.10 O Processo Final

Um desenvolvedor individual é responsável pela análise completa de requisitos e projeto do sistema, desde as primeiras fases. Por esta razão, não deve se considerar apenas o projeto de ciclos de desenvolvimento independentemente, mas também o projeto do sistema como um todo. Esta necessidade se tornou clara ao longo da investigação. A ausência de planejamento geral, para além do escopo de um único ciclo, gerou riscos desnecessários, como alteração de funcionalidades já implementadas. O processo final inclui uma fase inicial de Análise, onde há o entendimento de requisitos de alto nível, modelagem de dados e de componentes do sistema, com identificação de generalizações e possibilidades de reuso. Tal fase é executada de forma mais tradicional, pois um sistema bem estruturado é tão importante para um indivíduo, quanto para uma equipe. Um indivíduo, porém, não necessita gerar artefatos de documentação.

Durante a investigação optou-se por utilizar práticas de análise bastante flexíveis, que permitem ao desenvolvedor tomar diversas decisões durante o desenvolvimento. Quando trabalha sozinho, o desenvolvedor pode tomar decisões à parte do projeto inicial, sem que haja conflito com outros desenvolvedores. Esta prática por horas se demonstrou satisfatória, mas em certas situações, porém, se demonstrou falha e bastante arriscada. Um exemplo de situação de risco é na seleção de componentes de interface gráfica. Sem uma biblioteca de componentes pré estabelecida, o desenvolvedor pode passar a tomar decisões de UX enquanto desenvolve a aplicação, causando inconsistências que afetam negativamente o usuário. Como toda aplicação possui interface gráfica, a elaboração de tal biblioteca de componentes de GUI foi incluída na fase de projeto com o nome "Elaboração de biblioteca de Widgets". Nela, porém, apenas se lista os Widgets e alguns comentários relevantes, deixando a implementação para quando forem necessárias. Cada Widget deve ser elaborado como um projeto próprio de curtíssima duração, de modo a permitir reuso total, utilizando para tal os requisitos de uso que entreguem valor imediatamente no sistema.

A XP possui recomendações de constante integração de código com a equipe, de modo a evitar versões muito distintas do sistema em diferentes ambientes de trabalho. Para um indivíduo, além de esse não ser um problema, a constante integração de código é burocrática e atrapalha o desenvolvimento. Por outro lado, deve se isolar novas funcionalidades sendo trabalhadas de código já funcional, de modo a evitar conflitos e erros de difícil recuperação. Por esta razão se desenvolveu um método de integração que mantém a possibilidade de recuperação de código funcional, mas também não burocratiza o desenvolvimento.

O versionamento é concentrado em funcionalidades inteiras, que inclui todas as atividades de desenvolvimento que foram extraídas da mesma. Percebeu-se também ser mais vantajoso, e de entrega mais rápida de valor, que a ordem de execução dessas atividades também se dê por funcionalidades completas. Desta forma se entrega funcionalidades completas mais rapidamente e se garante que no apenas as funcionalidades necessárias estão sendo implementadas em cada camada do sistema. Apesar de neste experimento ter se utilizado a abordagem *bottom-up*, ou *backend-first*, alguns projetos podem ser melhor implementados da forma oposta. Usualmente se desenvolve primeiro o lado que funciona como guia do outro. Aqui, a modelagem de dados, utilizada no *backend*, guiou o projeto de diversas telas no *frontend*. Dado que cada funcionalidade é entregue por completo, essa questão não é de grande influência no desenvolvimento geral.

5.10.1 Modelagem do Processo

Foram utilizados diagramas BPMN para modelar o processo final. Seguindo o diagrama, há diversas descrições, sugestões e comentários. A Figura 5.24 demonstra em alto nível as etapas envolvidas no processo final. O processo inicia com o contato inicial do cliente buscando um serviço do desenvolvedor e se conclui com o projeto concluído e entregue.

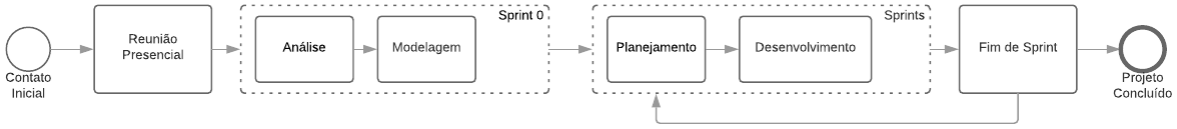


Figura 5.24 - Visão geral do Processo

Quando ocorre o contato inicial, há uma reunião presencial, preferencialmente no ambiente onde o problema é notado. Nesta reunião há uma conversa aberta sobre o cliente, sua área de atuação e seus objetivos no mercado. Também há a definição do responsável pela interlocução entre cliente e desenvolvedor, chamado *Product Owner (PO)*. Esta etapa é chamada "Contextualização" e produz um texto, um rascunho, um diagrama, ou qualquer forma de artefato que registre a conversa, o qual na Figura 5.25 é chamado de "Contexto". Em seguida há uma fase de idealização de proposta de solução, onde se identifica Usuários e funcionalidades de alto nível, registrando-os em um Mapa Mental e em rascunhos de *Wireframes*, o artefato "Visão Geral". Caso o cliente esteja satisfeito com o resultado, encerra-se a reunião e o desenvolvedor leva os artefatos produzidos para o seu ambiente de trabalho.

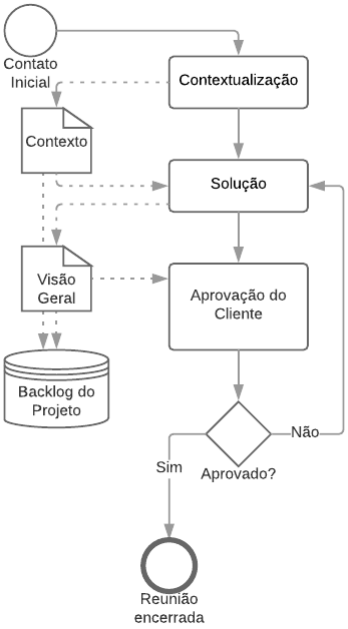


Figura 5.25 - Processo de Reunião Presencial

Já em seu ambiente de trabalho, inicia-se a *sprint* 0, ou fase de Planejamento do projeto. O desenvolvedor utiliza os artefatos "Contexto" e "Visão Geral" para identificar Atores que interagirão com a solução a ser desenvolvida e quais são os requisitos Funcionais e Não-Funcionais esperados pelo cliente. Com o artefato "Requisitos", que comumente se constitui em uma tabela de Requisitos, com descrição, tipo (funcional ou não-funcional) e um nome identificador, elabora-se um Diagrama de Casos de Uso de Alto Nível, chamado "Diagrama de Épicos". Inspirado na prática de elaboração de *Épicos* utilizado no *Scrum*, que utiliza *Estórias do Usuário* de alto nível, neste processo utiliza-se Casos de Uso, por serem mais rápidos. Como parte da Análise, elabora-se também um Diagrama de Entidade-Relacionamento (DER), para auxiliar na identificação e relacionamentos entre Entidades, ou Recursos de Dados. A Figura 5.26 trás a fase de Análise que se inicia com o processo de planejamento.

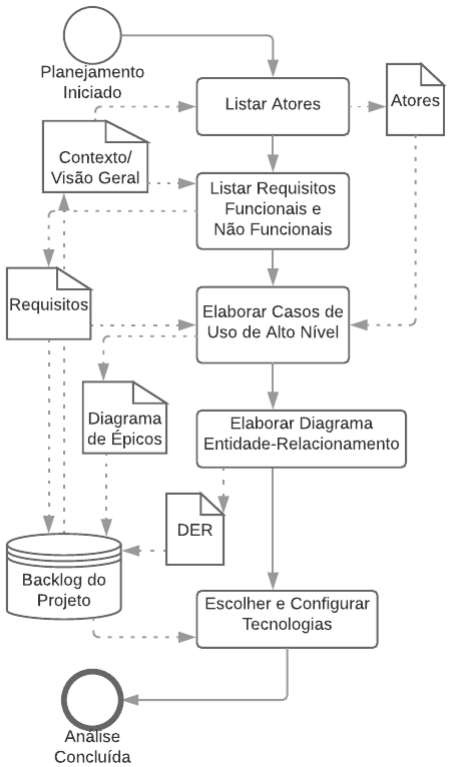


Figura 5.26 - Processo de Análise da Aplicação

Na Figura 5.27 está modelado o processo de Modelagem da Aplicação, de baixo para cima. Este processo se inicia na conclusão do processo de Análise e nele há

um subprocesso onde para cada Caso de Uso/Funcionalidade identificada no "Diagrama de Épicos" são elaborados os seguintes artefatos:

1. *Wireframes* que sugerem páginas para solucionar a funcionalidade buscada. Também deve se identificar páginas com funcionalidades similares, refinar projeto de *wireframes* caso haja possibilidade de aprimorar a experiência do usuário a nível global.
2. Listas de classes/componentes identificados em três tipos:
    a. Fronteira
        i. Atores se comunicam apenas com classes de fronteira
        ii. Cada relacionamento no Diagrama de Casos de Uso possui uma classe de fronteira
        iii. Classes de GUI, onde estão apenas as informações a serem exibidas. Utilizar Wireframes.
        iv. Interfaces com sistemas e dispositivos: Assinaturas de métodos utilizados em protocolos de comunicação.
    b. Controle
        i. Implementam funcionalidades dos casos de uso
        ii. Separam Entidade do comportamento do caso de uso
        iii. No geral existe um controle por Caso de Uso
    c. Entidade
        i. Recursos, substantivos do texto de escopo
        ii. Armazena dados
3. Lista de Recursos persistidos no *backend*. No geral são as classes de Entidade.
4. Diagrama de Comportamento. Um Fluxograma, ou Diagrama de Atividade, que relaciona o comportamento das diversas classes identificadas de modo a solucionar a funcionalidade.

Com estas informações é possível construir um Diagrama de Classes e identificar oportunidades de utilizar Padrões de Projeto/Reuso/Subsistemas. Ao longo

da investigação, subsistemas foram identificados como módulos, por ser o termo mais comumente utilizado no desenvolvimento Web. Ao fim, utilizando os artefatos gerados e em especial os *Wireframes*, elabora-se uma lista de componentes de GUI, ou Widgets, comuns a várias páginas, que no seu devido momento serão elaborados e utilizados ao longo do projeto. Esta etapa é importante para garantir consistência entre as páginas e melhor UX. Finalizada a Modelagem do projeto, é possível estimar prazos e custos para o desenvolvimento da aplicação e dessa forma entrar num acordo de produção com o cliente. Neste momento sugere-se a flexibilização do acordo, que pode sofrer alterações ao longo do desenvolvimento, com entregas em ciclos de curta duração. Neste momento conclui-se a fase de Planejamento. Toda esta etapa está registrada na Figura 5.27.

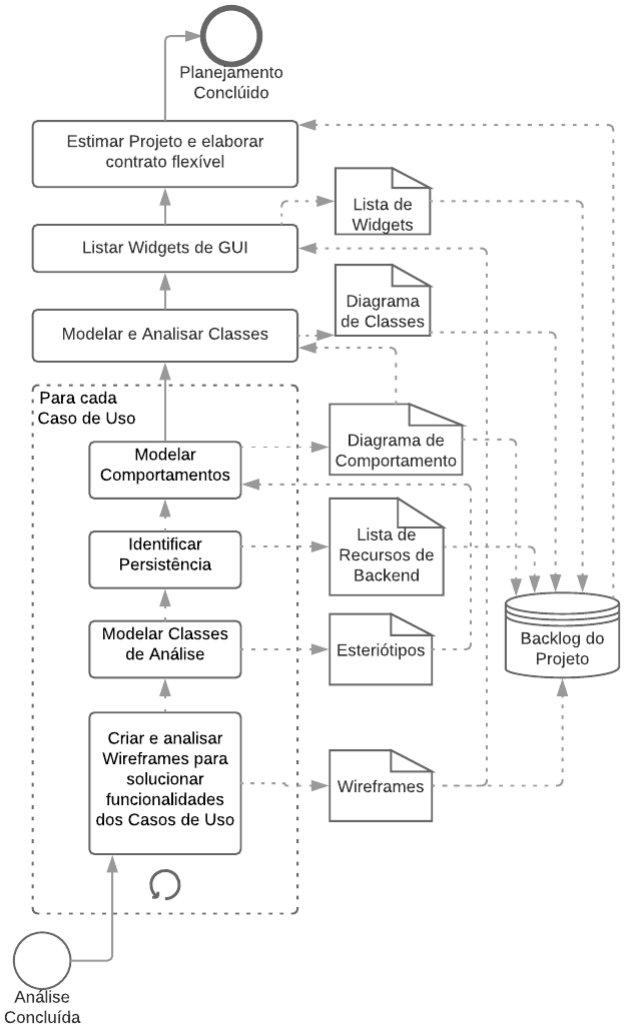


Figura 5.27 - Processo de Modelagem da Aplicação

Com uma visão global do sistema já projetado, iniciam-se os ciclos de desenvolvimento. Na XP, *sprints* são iniciadas na prática *Jogo de Planejamento* (Figura 5.28). O jogo do planejamento se inicia na seleção de Casos de Uso de alto nível mais interessantes para o *PO* e seu detalhamento em Casos de Uso menores. Com as funcionalidades selecionadas, elabora-se o "Mapa de *Wireframes*", incluindo em um único diagrama informações de interface gráfica, URLs, *endpoints*, navegação e comunicação com *backend*. Com isso é possível listar atividades de desenvolvimento para cada funcionalidade e estimá-las, de modo que caibam no período de uma *sprint*. As atividades devem estar bem categorizadas por funcionalidade em "Lista de Atividades".

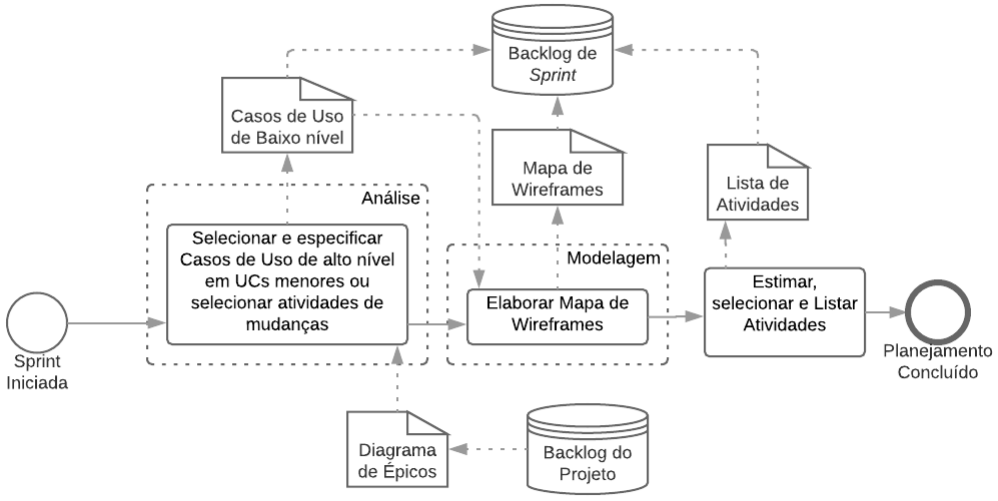


Figura 5.28 - Jogo do Planejamento da *Sprint*

A partir da elaboração da lista de atividades de desenvolvimento, concluída com a fase de planejamento da *sprint*, inicia-se um ciclo de desenvolvimento, que por sua vez é composto por diversos ciclos menores (Figura 5.29). O primeiro ciclo desenvolve funcionalidades completas. Se escolhe uma funcionalidade do artefato "Lista de Atividades". Caso haja uma funcionalidade para implementar, cria-se um ramo de versionamento e inicia-se outro ciclo, de implementação de Atividades. Nele, seleciona-se uma atividade

para ser implementada por vez, iniciando um ciclo como o *TDD*, mas na frequência e método que o desenvolvedor julgar necessário, inclusive apenas uma simples validação manual. Após implementar e testar até a atividade estar concluída, reinicia-se o ciclo de implementação de Atividades. Caso não haja mais atividades, integra-se o ramo de versionamento para a funcionalidade no repositório de versionamento remoto e inicia-se outro ciclo de implementação de Funcionalidades. Caso não haja mais funcionalidades a serem implementadas, publica-se as alterações, validando-as com o cliente. Caso estejam incorretas, adiciona-se as alterações no *Backlog* do Projeto, para serem implementadas em uma outra *sprint*, possivelmente até a próxima, caso o *PO* julgue prioritário.

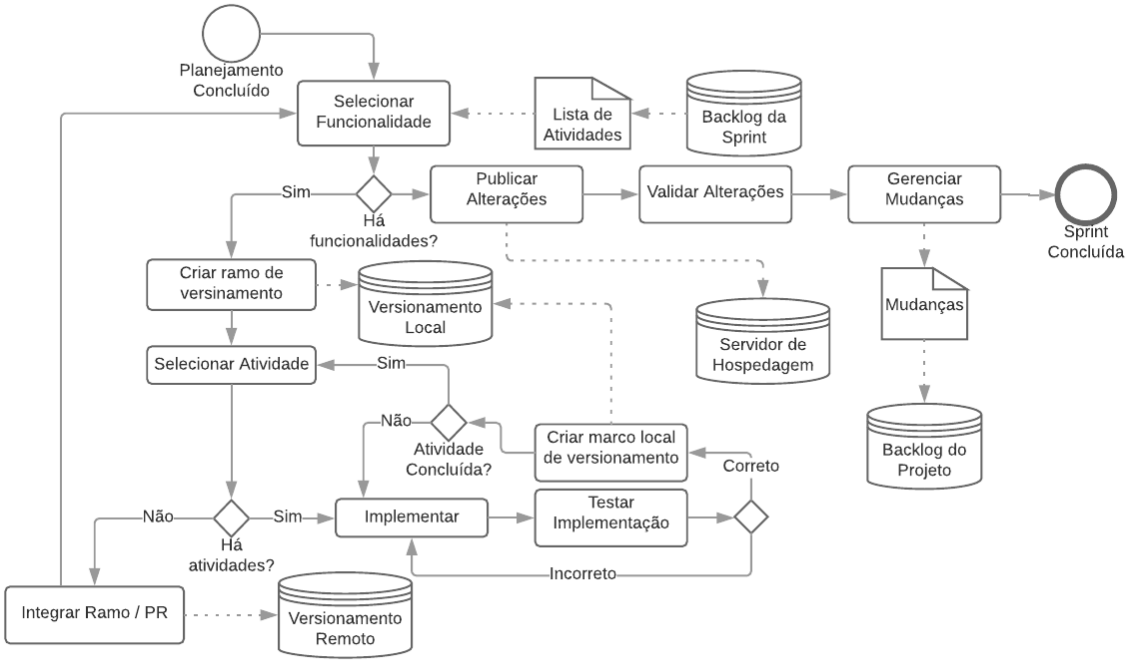


Figura 5.29 - Processo de Desenvolvimento da *Sprint*

Ao fim de cada *sprint* há um breve processo de avaliação se ainda existem funcionalidades a serem implementadas (Figura 5.30). Caso haja, uma nova *sprint* se inicia. Caso contrário, o processo de desenvolvimento está concluído.

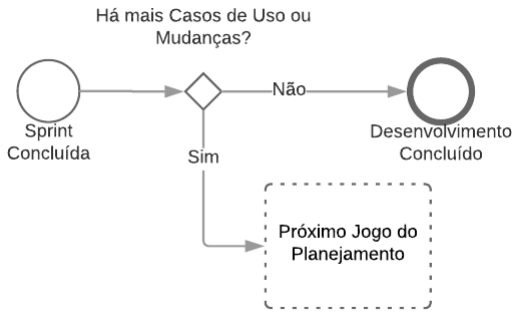


Figura 5.30 - Processo de conclusão de *Sprint*

6 Conclusão

No presente trabalho foi demonstrado a utilização de um método de Investigação-Ação na busca pela formalização de um processo de desenvolvimento Web para um único indivíduo. Através do método de Tentativa e Erro foram experimentadas diversas práticas e ferramentas de modelagem recomendadas pela metodologia Ágil, em especial a metodologia XP. A relevância deste trabalho está no grande número de desenvolvedores *freelancers* observados em fóruns e sites de trabalho *freelancer* que iniciaram no desenvolvimento Web pela alta demanda do mercado (WEBYDO, 2013) e oferta de cursos de baixo custo. Dentre as plataformas que suportam aplicações, a plataforma Web alcança usuários de todos os dispositivos Desktop e Mobile, possuindo para isso maior nível de abstração de código, resultando em grande interesse da comunidade não acadêmica em se profissionalizar na área, mesmo que informalmente.

O processo de desenvolvimento Ágil envolve a entrega rápida de valor ao cliente. Consequentemente, há ciclos curtos de seleção, implementação e entrega de requisitos. Alinhando este ciclo de desenvolvimento ao ciclo de melhoria de processos *PDCA*, foi estabelecido o processo cíclico de adequação utilizado neste trabalho. A cada ciclo houve a execução e avaliação de diversas práticas, adaptando-as ou excluindo-as do conjunto de práticas consideradas adequadas a um desenvolvedor individual, como já descrito. A cada ciclo observou-se uma constante aproximação dos princípios Ágeis, que são intrinsecamente flexíveis, e consequentemente um crescente distanciamento dos processos como comumente utilizados por equipes, necessariamente mais burocráticos, ainda que sigam a filosofia Ágil. Como resultado, foi obtido um processo bem definido, mas bastante leve e flexível.

Como definido por Flávia Neves no *site normaculta.com.br*, "A linguagem é um sistema de signos ou símbolos usados na transmissão de uma mensagem. É a capacidade de comunicação de ideias, pensamentos, opiniões, sentimentos, experiências, desejos, informações" (NEVES, 2018). Linguagens devem ser capazes de padronizar a transmissão de informação, de modo que todos os envolvidos a

compreendam. Por isso, linguagens de modelagem, como a *UML* (Linguagem de Modelagem Unificada, em tradução livre), estabelecem diversas normas e burocratizam o processo de elaboração. As primeiras descobertas ocorridas ao longo da investigação aqui relatada foram de flexibilização dos diagramas de *UML* ao ponto que foi considerado o seu abandono por completo. Porém em ciclos mais complexos, como a própria *sprint 0*, de análise global, a linguagem da *UML* se demonstrou útil, mesmo que ainda com normas flexibilizadas. Ao final, se desenvolveu e adicionou às práticas utilizadas, um outro diagrama, unificado, para atingir fins diversos, que uma única pessoa possa representar rapidamente a maior parte do seu planejamento.

Por dificuldades técnicas, neste experimento não foi possível executar o uso da prática *TDD*, *Test Driven Development*, onde o uso de testes automáticos é central no processo de desenvolvimento. Houveram tentativas de usá-la, porém foi considerada custosa e acabou sendo abandonada. Apesar de ser altamente incentivada entre desenvolvedores profissionais, desenvolvimento de testes é uma tarefa complexa e empresas usualmente possuem funcionários dedicados apenas a sua elaboração. No processo resultante deste trabalho a prática *TDD* foi deixada de lado, por ser muito custosa para um desenvolvedor individual, especialmente sem maiores formações específicas para testes. Foi utilizada uma prática de validação cíclica genérica, de forma que o desenvolvedor decida como melhor aplicá-la, se manualmente ou automaticamente.

A expectativa de se obter um processo muito mais leve que os utilizados por equipes influenciou na seleção de práticas iniciais. A metodologia final da investigação se demonstrou falha na escolha inicial de tais práticas, partindo de um conjunto insuficientemente elaborado. Com melhores práticas iniciais, sua relevância seria percebida mais cedo no experimento e as práticas selecionadas poderiam ser avaliadas por mais ciclos. A experiência provida por este trabalho contradiz essa expectativa que práticas de modelagem burocratizam demasiadamente um projeto de software, ao ponto de se tornarem descartáveis para um desenvolvedor trabalhando sozinho. Algumas adaptações tiveram que ser utilizadas, porém as práticas mais básicas para

equipes permaneceram no processo final para um indivíduo. Ainda se faz necessário avaliar o uso do processo individual em sistemas mais complexos, analisando o quão proveitoso é o seu uso em comparação com a sua ausência.

REFERÊNCIAS

1. MANYIKA, James et al. Independent work: Choice, necessity, and the gig economy. 2016. Disponível em: <https://www.mckinsey.com/featured-insights/employment-and-growth/independent-work-choice-necessity-and-the-gig-economy>. Acesso em: 25 nov. 2018.
2. NOGUCHI, Yuki. Freelanced: The Rise Of The Contract Workforce. 2018. Disponível em: <https://www.npr.org/2018/01/22/578825135/rise-of-the-contract-workers-work-is-different-now>. Acesso em: 25 nov. 2018.
3. UPWORK. Freelancing in America: 2018. 2018. Disponível em: <https://www.google.com/url?q=https://www.upwork.com/press/2018/10/31/freelancing-in-america-2018/&sa=D&ust=1543182316271000&usg=AFQjCNGmWzRUuTS4xnR6HzSFgm2YztyR6g>. Acesso em: 24 nov. 2018.
4. PRESSMAN, Roger S.. Software Engineering: A Practitioner's Approach. [s.l.]: Mcgraw-hill, 2005.
5. GALIN, Daniel. Software Quality: Concepts and Practice. [s.l.]: Ieee Press, 2018.
6. DZHUROV, Yani; KRASTEVA, Iva; ILIEVA, Sylvia. Personal Extreme Programming: An Agile Process for Autonomous Developers. 2018. 259 f. TCC (Graduação) - Curso de Mathematics And Informatics, Sofia University, [s.l.], 2009.
7. DATA USA. DATAUSA: Web Developers. 2016. Disponível em: <https://datausa.io/profile/soc/151134/#growth>. Acesso em: 25 nov. 2018.
8. AGRAWAL, Tushar. Why Progressive Web Apps Are The Future Of Web Development. 2017. Disponível em: <https://medium.freecodecamp.org/why-progressive-web-apps-are-the-future-of-web-development-13db7dd5f640>. Acesso em: 25 nov. 2018.

9. THE NPM BLOG. How many npm users are there? 2016. Disponível em: <https://blog.npmjs.org/post/143451680695/how-many-npm-users-are-there>. Acesso em: 25 nov. 2018.
10. WEB FOUNDATION. History of the Web. [20--] Disponível em: <https://webfoundation.org/about/vision/history-of-the-web/>. Acesso em: 19 ago. 2018.
11. BERNERS-LEE, Tim. What do HTTP URIs Identify? 2002. Disponível em: <https://www.w3.org/DesignIssues/HTTP-URI.html>. Acesso em: 25 nov. 2018.
12. NATIONS, Daniel. What Exactly Is a Web Application?: Improve your understanding of web-based application programs. Lifewire. 22, fev. 2018. Disponível em: <https://www.lifewire.com/what-is-a-web-application-3486637>. Acesso em: 20 ago. 2018.
13. DROIDHEAD. Understanding How the Chrome V8 Engine Translates JavaScript into Machine Code. Medium. 20 dez. 2017. Disponível em: <https://medium.freecodecamp.org/understanding-the-core-of-nodejs-the-powerful-chrome-v8-engine-79e7eb8af964>. Acesso em: 20 ago. 2018.
14. WEST, Matt. Exploring the JavaScript Device APIs. 2013. Disponível em: <http://blog.teamtreehouse.com/exploring-javascript-device-apis>. Acesso em: 20 ago. 2018.
15. STACKIFY. What is N-Tier Architecture? How It Works, Examples, Tutorials, and More. 2017. Disponível em: <https://stackify.com/n-tier-architecture/?utm_referrer=https%3A%2F%2Fwww.google.com.br%2F>. Acesso em: 19 maio 2017.
16. MICROSOFT DOCS. Deployment Patterns. 2014. Disponível em: <https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff646997(v%3dpandp.10)>. Acesso em: 22 ago. 2018.

17. CLEARBRIDGE MOBILE. A Guide to Mobile App Development: Web vs. Native vs. Hybrid. 2018. Disponível em: <https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/#WEB_VS_NATIVE_VS_HYBRID>. Acesso em: 22 ago. 2018.
18. ANDREW, Paul. The Next Step in the Evolution of Responsive Web Design: Responsivity Analysis. 2016. Disponível em: <https://speckyboy.com/the-evolution-of-responsive-web-design-responsivity-analysis/>. Acesso em: 22 ago. 2018.
19. TECHTARGET. Latency. 2016. Disponível em: <https://whatis.techtarget.com/definition/latency>. Acesso em: 22 ago. 2018.
20. TABLELESS. Fluxo de execução assíncrono em JavaScript: Generators e Async Functions. 2016. Disponível em: <https://tableless.com.br/fluxo-de-execucao-assincrono-em-javascript-generators-e-async-functions/>. Acesso em: 22 ago. 2018.
21. NETO, Waldemar,. Node.js: V8, Single thread e I/O não bloqueante. 2018. Disponível em: <https://imasters.com.br/front-end/node-js-v8-single-thread-e-io-nao-bloqueante>. Acesso em: 22 ago. 2018.
22. JACKSON, G.S. The Advantages of Modularization in Programming. [2018]. Disponível em: <https://www.techwalla.com/articles/the-advantages-of-modularization-in-programming>. Acesso em: 22 ago. 2018.
23. MODULECOUNTS. 2018. Disponível em: <http://www.modulecounts.com/>. Acesso em: 22 ago. 2018.
24. DIERX, Michael Wanyoike Peter. A Beginner's Guide to npm: The Node Package Manager. 2018. Disponível em: <https://www.sitepoint.com/beginners-guide-node-package-manager/>. Acesso em: 22 ago. 2018.

25. CEROS. Why Does User Experience Matter? 2017. Disponível em: <https://www.ceros.com/originals/why-does-user-experience-matter/>. Acesso em: 25 ago. 2018.
26. CEROS. UX Design & Digital Storytelling: Why Story Is Central to User Experience. 2016. Disponível em: <https://www.ceros.com/originals/reframing-ux-design-digital-storytelling/>. Acesso em: 25 ago. 2018.
27. THINKWIK. Do you know the Importance of UI/UX Development? 2010. Disponível em: <https://medium.com/@thinkwik/do-you-know-the-importance-of-ui-ux-development-773eae38436e>. Acesso em: 26 ago. 2018.
28. ZHANG, Ping; VON DRAN, Gisela M.. User Expectations and Rankings of Quality Factorsin Different Web Site Domains. International Journal Of Electronic Commerce. [s. L.], p. 9-33. dez. 2002. Disponível em: <https://www.researchgate.net/profile/Ping_Zhang24/publication/2560629_User_Expectations_and_Rankings_of_Quality_Factors_in_Different_Web_Site_Domains/links/0c960524316506f697000000/User-Expectations-and-Rankings-of-Quality-Factors-in-Different-Web-Site-Domains.pdf?origin=publication_detail>. Acesso em: 25 ago. 2018.
29. ANDERSON, Shaun. How Fast Should A Website Load in 2018? 2018. Disponível em: <https://www.hobo-web.co.uk/your-website-design-should-load-in-4-seconds/>. Acesso em: 26 ago. 2018.
30. WAGNER, Jeremy. Why Performance Matters. 2018. Disponível em: <https://developers.google.com/web/fundamentals/performance/why-performance-matters/>. Acesso em: 27 ago. 2018.
31. SERRADOR, Pedro; PINTO, Jeffrey K.. Does Agile work?: A quantitative analysis of agile project success. International Journal Of Project Management. [s.l.], p. 1040-1051. Julho, 2015.

32. NOFFKE, Susan; SOMEKH, Bridget. The SAGE Handbook of Educational Action Research. [s.l.]: Sage, 2009.
33. Kock, N. Action Research: Its Nature and Relationship to Human-Computer Interaction. 2011.
34. BALDISSERA, Adelina. Pesquisa-Ação: Uma Metodologia do "Conhecer" e do Agir Coletivo. Pelotas, 2001.
35. HAHN, Heidi A. Planning for Action Research: Looking at Practice through a Different Lens. Orlando, 2012.
36. HUNT, E; MCKAY, EA. A scoping review of time-use research in occupational therapy and occupational science. Scandinavian Journal Of Occupational Therapy. Cork, p. 1-12. ago. 2014.
37. XIMENES, Filipe; JUVENAL, Flávio. How to design a RESTful API architecture from a human-language spec. 2017. Disponível em: <https://www.oreilly.com/learning/how-to-design-a-restful-api-architecture-from-a-human-language-spec>. Acesso em: 26 nov. 2018.
38. FILGUEIRA, Israel E.. O que você precisa saber sobre State Manager, single source of truth (SSOT) e Vuex—part 1. [2018]. Disponível em: <https://medium.com/@IsraelEriston/o-que-voc%C3%AA-precisa-saber-sobre-state-manager-single-source-of-truth-ssot-e-vuex-part-1-db5442e4063>. Acesso em: 27 nov. 2018.
39. CONALLEN, Jim. Modeling Web Application Architectures with UML. Cupertino: Rational Software, 1999.
40. CARDELLO, Jeff. The modern web design process: creating sitemaps. 2017. Disponível em: <https://webflow.com/blog/the-modern-web-design-process-creating-sitemaps-and-wireframes>. Acesso em: 27 nov. 2018.
41. ALLAN, Patrick. Estimate How Much Time You Need for a Project, Then Double It. 2014. Disponível em:

<https://lifehacker.com/estimate-how-much-time-you-need-for-a-project-then-dou-1592791225>. Acesso em: 27 nov. 2018.

42. "BEN". How do you estimate time required for assigned task? 2018. Disponível em: <https://dev.to/imben1109/how-do-you-estimate-time-required-for-assigned-task-57md>. Acesso em: 27 nov. 2018.
43. HOFSTADTER, Douglas R. Gödel, Escher, Bach: An Eternal Golden Braid. [s.l.]: Basic Books, 1999.
44. TEA, Matthew. A Massive Guide to Building a RESTful API for Your Mobile App. 2017. Disponível em: <https://savvyapps.com/blog/how-to-build-restful-api-mobile-app>. Acesso em: 27 nov. 2018.
45. PROJECTMANAGER.COM. The Ultimate Guide to: Gantt Charts: Plan Your Projects with ProjectManager.com's Award-winning Gantt Chart Tools.. [2018]. Disponível em: <https://www.projectmanager.com/gantt-chart>. Acesso em: 27 nov. 2018.
46. ANDROID OPEN SOURCE PROJECT. Putting it all together: Wireframing the example app. 2018. Disponível em: <https://developer.android.com/training/design-navigation/wireframing>. Acesso em: 27 nov. 2018.
47. WEBYDO. Market Analysis of the Web Design Industry. 2013. Disponível em: <https://www.prnewswire.com/news-releases/market-analysis-of-the-web-design-industry-236426961.html>. Acesso em: 29 nov. 2018.
48. NEVES, Flávia. Linguagem, língua e fala: Qual a diferença?. Disponível em: <https://www.normaculta.com.br/linguagem-lingua-e-fala-qual-a-diferenca/>. Acesso em: 29 nov. 2018.
49. VISUAL PARADIGM. What is Unified Modeling Language (UML)? [20--] Disponível em:

<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#uml-overview>. Acesso em: 29 nov. 2018.