CENTRO UNVERSITÁRIO DE ANÁPOLIS - UniEVANGÉLICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

JOSUÉ HENRIQUE FERREIRA DA SILVA
NOEMI MENDES PIMENTEL

DESENVOLVIMENTO DE SOFTWARE: AUTOMAÇÃO PARA ESTABELECIMENTOS COMERCIAIS DE ALIMENTOS

ANÁPOLIS – GO
2016

JOSUÉ HENRIQUE FERREIRA DA SILVA

NOEMI MENDES PIMENTEL

DESENVOLVIMENTO DE SOFTWARE: AUTOMAÇÃO PARA ESTABELECIMENTOS COMERCIAIS DE ALIMENTOS

Trabalho de Conclusão de Curso II apresentado como requisito parcial à obtenção do grau de Bacharel em Engenharia de Computação do programa de graduação dos Cursos Superiores de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Orientadora: Profª Especialista Aline Dayany de Lemos.

ANÁPOLIS – GO
2016

JOSUÉ HENRIQUE FERREIRA DA SILVA

NOEMI MENDES PIMENTEL

DESENVOLVIMENTO DE SOFTWARE: AUTOMAÇÃO PARA ESTABELECIMENTOS COMERCIAIS DE ALIMENTOS

Trabalho de Conclusão de Curso II apresentado como requisito parcial à obtenção do grau de Bacharel em Engenharia de Computação do programa de graduação dos Cursos Superiores de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Orientadora: Profª Especialista Aline Dayany de Lemos.

Banca Examinadora

_____

Prof. Esp. Aline Dayany de Lemos
Orientadora / UniEVANGÉLICA

_____

Profª Ms. Luciana Nishi
Docente / UniEVANGÉLICA

_____

Prof. Ms. Marcelo de Castro Cardoso
Docente / UniEVANGÉLICA

Anápolis, 28 de novembro de 2016

RESUMO

O projeto refere-se ao desenvolvimento de software para restaurantes e lanchonetes que utilizam comandas ou cartões para controle de entrada, saída e consumo de seus clientes no estabelecimento. Ao se desenvolver um software é necessário definir os processos, métodos e ferramentas que auxiliarão sua produção. Neste trabalho os autores apresentam o processo de desenvolvimento definido, os métodos, ferramentas e incrementos que foram desenvolvidos.

PALAVRAS-CHAVE: desenvolvimento; restaurantes e lanchonetes; processo; métodos; ferramentas.

ABSTRACT

The project refers to the development of software for restaurants and snack bars that use commands or cards to control entrance, exit and consumption of their customers in the establishment. When developing software, it is necessary to define the processes, methods and tools that will aid in its production. In this work the authors present the defined development process, the methods, tools and increments that were developed.

KEY WORDS: development; restaurants and snack bars; process; methods; tools.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AM	*Agile Modeling* (Modelagem Ágil)
ASD	*Adaptative Software Development* (Desenvolvimento Adaptável de Software)
AUP	*Agile Unified Process* (Processo Unificado Ágil)
BDD	*Behaviour Driven Developmet* (Desenvolvimento Orientado por Comportamento)
CASE	*Computer Aided Software Engineering* (Engenharia de Software com o Auxílio do Computador)
DOM	*Document Object Model* (Modelo de Objeto de Documento)
DSDM	*Dynamic Systems Development Method* (Métodos de Desenvolvimento de Sistemas Dinâmicos)
DSL	*Domain Specific Language* (Linguagem Específica de Domínio)
FDD	*Feature Driven Development* (Desenvolvimento Guiado por Funcionalidades)
HTTP	*HyperText Transfer Protocol* (Protocolo de transferência de Hipertexto)
INVEST	*Independent; Negotiable; Valuable; Estimable; Small; Testable* (Independente; Negociável; Valiosa; Estimável; Pequena; Testável)
LSD	*Lean Software Development* (Desenvolvimento de Software Enxuto)
MVC	*Model-View-Controller* (Modelo-Visão-Controlador)
OTAN	Organização do Tratado do Atlântico Norte
SDK	*Software Development Kit* (Kit de Desenvolvimento de Software)
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language* (Linguagem de Consulta Estruturada)
UML	*Unified Modeling Language* (Linguagem de Modelagem Unificada)
XP	*Extreme Programming* (Programação Extrema)
YAML	*Yet Another Markup Language* (Ainda é Outra Linguagem de Marcação)

SUMÁRIO

INTRODUÇÃO

Estabelecimentos comerciais que vendem produtos alimentícios como, por exemplo, restaurantes e lanchonetes que utilizam comandas e/ou cartões enumerados para gerenciar a entrada, saída e consumo dos clientes, podem ter problemas referentes a realização de uma ação ludibriosa, conscientemente praticada pelo cliente dizendo que perdeu o cartão ou comanda; apropriação indébita do cartão ou comanda de outro cliente; consumo e entrega de cartão sem registro de uso, entre outros.

Devido a esses problemas os estabelecimentos treinam os funcionários para identificação de possíveis suspeitos que possam cometer tais atitudes. Quando ocorre a perca de cartão/comanda o cliente se sente constrangido pois cria um sentimento de auto culpa e de marginalização para com os funcionários do estabelecimento. Baseado nos cenários acima como controlar a entrada e saída de clientes dos estabelecimentos? Como minimizar a perca da comanda, ou cartão? Como manter digitalmente o rastreio do que foi consumido no estabelecimento?

Estes problemas foram identificados pelos pesquisadores após passarem pela situação constrangedora de perderem o cartão de uma lanchonete e terem que informar ao gerente o ocorrido.

Então surgiu a ideia de desenvolver um sistema para lanchonetes e restaurantes que utilizam comanda e/ou cartão para controlar a entrada e saída de seus clientes e que desejam utilizar as vantagens tecnológicas vigentes para agregar valor ao seu negócio.

O CloudGourmet é um software web que propõe a utilização de um identificador único para os clientes desses estabelecimentos, o que pode auxiliar o controle de entrada e saída minimizando erros advindos da utilização dos métodos atuais.

Ao contrário de sistemas que não utilizam identificadores únicos para controle de entradas e saídas, nosso produto vinculará o histórico de consumo ao identificador único, então o cliente poderá verificar o que foi consumido nos estabelecimentos que utilizarem esse serviço.

A metodologia cientifica que está sendo utilizada na elaboração desse trabalho é denominada pesquisa-ação que Vergara descreve como “método de pesquisa que visa à resolução de problemas por meio de ações definidas por pesquisadores e sujeitos envolvidos com a situação sob investigação” (2009, p.190) para conseguir abstrair as informações referentes ao desenvolvimento do software.

Uma das características do software que está sendo desenvolvido é a incerteza, pois seu intuito é apresentar uma solução referente a um problema presente em lanchonetes e restaurantes que utilizam a comanda ou cartão para controle de entrada e saída. Ele não possuiu um cliente específico, pois o objetivo da equipe é produzir o produto mínimo viável, apresentando alguma solução referente ao problema e ir incrementando novas funcionalidades que agreguem valor ao produto.

Baseado nesse cenário, os modelos de processos mais indicados são as metodologias ágeis, no processo de desenvolvimento adotado pode-se identificar principalmente as características do *Scrum* que Sommerville (2011) descreve como uma metodologia ágil com foco no gerenciamento do desenvolvimento de software iterativo.

A realização do levantamento dos itens do *Product Backlog* é através de observações presenciais em restaurantes e lanchonetes, levantando as possíveis necessidades e investigando de maneira informal. Outro meio de obtenção de informação é através do ciberespaço, onde os autores buscam respostas às questões levantadas.

No capítulo 1 é apresentado uma breve história referente a engenharia de software e suas características. São descritos os modelos de processos tradicionais e ágeis e apresentados as atividades genéricas que podem ser identificadas nos processos de desenvolvimento de software.

No capítulo 2 é apresentado o modelo do processo de desenvolvimento que está sendo utilizado no desenvolvimento do software. No capítulo 3 é apresentado a execução do projeto de acordo com o processo e métodos definidos. No capítulo 4 é apresentado as ferramentas que foram selecionadas para auxiliar no desenvolvimento e disponibilização do software.

1 ENGENHARIA DE SOFTWARE

"Ninguém poderia prever que o software seria incorporado em sistemas de todas as áreas: transportes, medicina, telecomunicações, militar, industrial, entretenimento, máquinas de escritório... A lista é quase infindável" (PRESSMAN, 2011, p. 30). Afinal o que é software?

Uma definição sucinta é apresentada por Sommerville que o descreve como programas de computador e suas documentações que podem ser desenvolvidos para um cliente ou para um mercado geral (2011, p. 04).

No século XXI, os softwares se tornaram presentes em praticamente todas as áreas de nossas vidas, portanto, deve-se esforçar para compreender o problema antes de desenvolver uma solução, com o aumento da complexidade dos softwares tornou-se necessário projetar e sua disponibilidade deve ser priorizada. O software deve ser passível de manutenção, logo, ele deve passar pelos processos de engenharia de software (PRESSMAN, 2011).

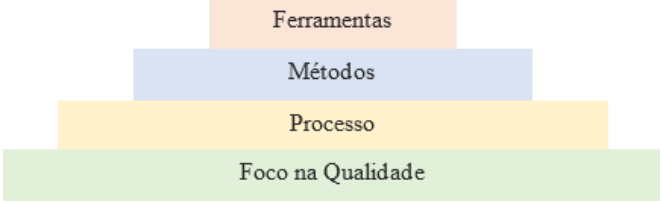
> A engenharia de software é uma disciplina cujo o objetivo é produzir software isento de falhas, entregue dentro de prazo e orçamento previstos, e que atenda às necessidades do cliente. Além disso, o software deve ser fácil de ser modificado quando as necessidades do usuário mudarem (SCHACH, 2009, p. 4).

Apesar do processo de engenharia de software ser similar aos processos de outras engenharias, ele tem características particulares que torna o seu desenvolvimento diferente e muitas vezes complexo. Algumas das características que devem ser levadas em consideração são: 1- Os requisitos sempre mudam; 2- Os softwares são desenvolvidos por seres humanos, logo, estão sujeitos a falhas (SCHACH, 2009). "Um produto de software é um modelo do mundo real, e o mundo real está em contínua mudança" (SCHACH, 2009, p. 40), por esse motivo ele é sujeito a tantas modificações.

1.1 Camadas da engenharia de software

Uma representação visual referente a engenharia de software é apresentada por Pressman (2011, p.39) consiste em um conjunto de quatro camadas, lembrando uma pirâmide (Figura 1). A primeira camada da engenharia de software tem foco na qualidade; a segunda nos processos; a terceira nos métodos; e a quarta nas ferramentas.

Figura 1- Camadas da engenharia de software.



Fonte: Adaptado de PRESSMAN (2011, p. 39).

1.1.1 Primeira camada: foco na qualidade

Definir o que é qualidade é uma difícil tarefa; assegurar a qualidade de um produto é ainda mais difícil, pois cada indivíduo tem uma visão diferente sobre o que é qualidade, em outras palavras: "Qualidade é algo que todos querem, porém é muito difícil defini-la" (HIRAMA, 2011, p. 140).

David Garvin (*apud* PRESSMAN, 2011, p. 359) sugere que a qualidade pode ser apresentada em cinco pontos de vistas diferentes e são eles:

- Visão transcendental: qualidade é reconhecível, mas não de maneira explícita;
- Visão do usuário: um produto atende seus objetivos e metas?
- Visão do fabricante: o produto atende as especificações?
- Visão do produto: a qualidade pode ser ligada as características peculiares e essenciais do produto?
- Visão baseada no valor: a qualidade pode ser identificada de acordo com o valor que o cliente estaria disposto a pagar;

A qualidade não está relacionada apenas aos pontos de vistas descritos acima, ela envolve também outros preceitos.

Uma especificação a respeito do objetivo da primeira camada da pirâmide apresentada por Pressman (Figura 1) foi apresentada por Hirama ao dizer "[...] qualidade é algo que deve ser conquistado e não simplesmente obtido" (2011, p. 140), pois todos os envolvidos devem estar dispostos a se esforçarem para obter resultados.

1.1.2 Segunda camada: processo

A camada referente ao processo proporciona a integração das camadas de métodos e ferramentas auxiliando o planejamento e controle de projetos de software (HIRAMA, 2011). O processo envolve a metodologia que será utilizada, as técnicas, ferramentas e indivíduos que participam do desenvolvimento do projeto (SCHACH, 2009, p.71).

O processo de software é definido como: "o conjunto de atividades, ações e tarefas realizadas na criação de algum produto de trabalho" (PRESSMAN, 2011, p.40). Uma atividade relata como atingir um determinado objetivo, independente do projeto, da complexidade, esforços ou nível de rigidez com que serão aplicadas. A tarefa é a quantidade de trabalho bem definida que deve produzir um resultado tangível e a ação corresponde ao conjunto de tarefas que serão realizadas para a criação do artefato (PRESSMAN, 2011).

Diferentemente do que se pensa, um processo de engenharia de software não é algo inflexível sobre como um software deve ser desenvolvido, pelo contrário, é algo flexível que

pode ser adaptado, no qual as pessoas podem selecionar e escolher um conjunto de tarefas e ações apropriadas (PRESSMAN, 2011).

1.1.3 Terceira camada: métodos

"A camada de métodos provê as abordagens e as atividades necessárias para a construção de um software" (HIRAMA, 2011, p. 9). "Os métodos de engenharia de software baseiam-se em um conjunto de princípios básicos que governam cada área da tecnologia e inclui atividades de modelagem e outras técnicas descritivas" (PRESSMAN, 2011, p. 40)

Para que ocorra uniformidade na comunicação dos envolvidos é necessário definir uma padronização referente a comunicação, ou seja, é necessário utilizar métodos. A utilização de métodos auxilia a padronização dos artefatos e consequentemente a manutenção e a inserção de novos integrantes na equipe (HIRAMA, 2011).

1.1.4 Quarta camada: ferramentas

As atividades de engenharia de software podem ser apoiadas por ferramentas de software, essas ferramentas são denominadas ferramentas CASE (*Computer Aided Software Engineering*). Atualmente existem diversas ferramentas para atender diferentes tipos de necessidades e auxiliar a execução das atividades de desenvolvimento, provendo qualidade e produtividade ao desenvolvimento de software (HIRAMA, 2011).

1.2 Modelos de processos de software

Os modelos de processos de software, também denominados como metodologia de processos, modelos de ciclo de vida ou *framework* de processo é uma representação do processo de desenvolvimento, seu objetivo é dar suporte ao processo de engenharia de software completo (PRESSMAN, 2011). Logo, os modelos de processos são utilizados para nortear o desenvolvimento de software.

Na literatura são apresentadas diferentes nomenclaturas para categorizar os processos de desenvolvimento de software, Sommerville (2011) classifica como processo dirigido a planos (abordagens pesadas) e processos ágeis, Koscianski e Soares (2007) classificam como metodologias tradicionais e metodologias ágeis e Pressman (2011) classifica as metodologias tradicionais como modelos de processo prescritivos e as metodologias ágeis como processo ágil.

Para auxiliar a compreensão dos modelos de desenvolvimento e não gerar dúvidas a respeito de suas características, será utilizado a nomenclatura metodologias tradicionais para descrever os processos dirigidos a planos e metodologias ágeis para descrever os processos ágeis.

1.2.1 Metodologias tradicionais

As metodologias tradicionais surgiram no princípio da engenharia de software e trouxeram grandes contribuições para a sua evolução. Suas atividades são planejadas antecipadamente e seu progresso é analisado de acordo com o planejamento inicial (SOMMERVILLE, 2011). "As metodologias tradicionais são também chamadas de pesadas ou orientadas a documentação" (ROYCE, 1970 *apud* KOSCIANSKI; SOARES, 2007, p. 191).

Acreditava-se que a melhor maneira de desenvolver softwares era através de um planejamento meticuloso do projeto, onde a qualidade da segurança era formalizada, o emprego de métodos de análise e projeto utilizando ferramentas e o emprego de um processo de desenvolvimento rigoroso e controlado. Esse pensamento surgiu entre pessoas envolvidas no desenvolvimento de softwares robustos e duradouros, tais como sistema aeroespaciais e de governos (SOMMERVILLE, 2011).

O modelo em cascata, também denominado como ciclo de vida clássico, é uma metodologia antiga aplicada no desenvolvimento de software e é utilizada até hoje, esse tipo de metodologia pode ser eficaz quando os requisitos são fixos e as atividades podem ser realizadas de forma linear (PRESSMAN, 2011).

Outras metodologias tradicionais são encontradas na literatura (PRESSMAN, 2011), tais como:

- Modelo V;
- Modelo de processo incremental;
- Modelo de processo evolucionário;
- Modelo espiral;
- Modelos concorrentes, entre outros.

As metodologias tradicionais são adequadas quando os sistemas são muito grandes; quando a equipe é dispersa geograficamente; em sistemas embutidos; e sistemas críticos que necessitam de uma análise minuciosa e um projeto bem detalhado; ou quando o sistema ou parte dele é terceirizado (SOMMERVILLE, 2011).

1.2.2 Metodologias ágeis

As metodologias ágeis são adequadas para desenvolvimento de software onde os requisitos se modificam com frequência. Por sua vez, uma metodologia só pode ser considerada ágil se aceita as mudanças ao invés de prevê-las (KOSCIANSKI; SOARES, 2007).

O desenvolvimento de software tem tido um ritmo acelerado e é propenso às mudanças intermináveis. Os métodos ágeis procuram resolver os problemas reais e perceptíveis das metodologias tradicionais (PRESSMAN, 2011).

O descontentamento referente a utilização de metodologias tradicionais provocou o surgimento da aliança ágil, que incentivam as equipes priorizarem a entrega de um incremento de software operacional e não sua concepção e documentação (SOMMERVILLE, 2011).

Em 2001, dezessete especialistas em processos de desenvolvimento de software criaram a Aliança Ágil (*Agile Alliance*), onde foram priorizados:

> Indivíduos e interações ao invés de processos e ferramentas; software operacional ao invés de documentação; colaboração do cliente ao invés de contratos; respostas a mudanças ao invés de planos (BECK, *et al..*, 2002, p. 2).

> A Aliança reconhece a importância dos processos e ferramentas, reconhecendo adicionalmente que a interação entre indivíduos capacitados tem ainda maior importância. Da mesma forma, documentação completa não é necessariamente ruim, mas o foco primário deve permanecer no produto final – a entrega de software operante. Entretanto, toda equipe de projeto precisa determinar por si só qual documentação é absolutamente essencial (BECK, *et al.*, 2002, p. 3).

O encontro desses especialistas resultou em um documento chamado de Manifesto do Desenvolvimento Ágil de Software (*Manifesto for Agile Software Development*) (BECK *et al.*, 2002). As metodologias conhecidas são (PRESSMAN, 2011, p. 91):

- *Extreme Programming* (XP)
- *Adaptative Software Development* (ASD)
- *Scrum*
- *Dynamic Systems Development Method* (DSDM)
- *Crystal*
- *Feature Driven Development* (FDD)
- *Lean Software Development* (LSD)
- *Agile Modeling* (AM)
- *Agile Unified Process* (AUP)

Os participantes do manifesto, definiram os doze princípios da aliança ágil que são:

1. Nossa prioridade mais alta é satisfazer o cliente através de entregas contínuas e antecipadas de software válido.
2. Mudanças nos requisitos são bem-vindas, mesmo as que chegam tarde no desenvolvimento. Processos ágeis asseguram a mudança como uma vantagem competitiva do cliente.
3. Entregar software produtivo frequentemente, de algumas semanas a alguns meses, de preferência os tempos mais curtos.
4. Pessoal de negócio e desenvolvedores trabalham juntos diariamente durante o projeto.
5. Criar projetos em torno de indivíduos motivados, proporcionar o ambiente e suporte que eles necessitam e confiar que eles farão o serviço.

> 6. O método mais eficiente e efetivo para transmitir informações entre e para a equipe de desenvolvimento é conversão cara a cara.
> 7. Software produtivo é a medida primária do progresso.
> 8. Processos ágeis promovem um desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
> 9. Atenção contínua à excelência técnica e boa solução melhoram a agilidade.
> 10. Simplicidade – a arte de maximizar a quantidade de trabalho não feita – é essencial.
> 11. As melhores arquiteturas, requisitos e projetos e*merge*m de equipes auto-organizadas.
> 12. Em intervalos regulares, a equipe reflete sobre como se tornar mais efetiva e então sintoniza e ajusta seu comportamento de forma apropriada. (BECK *et al.*, 2002, p. 8).

Assim, para uma metodologia ser considerada ágil ela deve estar em conformidade com o manifesto ágil. Algumas metodologias ágeis não aplicam os princípios ágeis citados acima com pesos iguais, mas esses princípios definem o que pode ser considerado o espírito ágil (PRESSMAN, 2011).

Uma metodologia ágil deve se adaptar incrementalmente. Para que isso ocorra, a equipe deve ter constante feedback do cliente. Uma boa prática para se obter esse feedback é através dos incrementos de software (PRESSMAN, 2011).

Os incrementos são "[...] partes do sistema real, ou seja, não existe a necessidade de reaprendizagem quando o sistema completo está disponível" (SOMMERVILLE, 2011, p. 32). Devem ser entregues em curtos períodos de tempo, para que as adaptações acompanhem o mesmo ritmo das mudanças (imprevisibilidade). A utilização dessa abordagem capacita o cliente a validar o que foi desenvolvido, fornecendo o feedback necessário, podendo influenciar nas adaptações necessárias (PRESSMAN, 2011).

O desenvolvimento de software utilizando metodologia ágil não é indicado para todos os projetos, produtos, pessoas e situações. O pensamento ágil evidencia a importância de equipes auto-organizáveis; aceitação de mudanças como oportunidades; foco na entrega de incrementos de softwares para satisfação do cliente (PRESSMAN, 2011).

A principal preocupação de quem irá utilizar um software é a entrega de um sistema de software executável, que atenda às necessidades e que realize coisas úteis para seus usuários, logo, caracterizar a metodologia como ágil ou tradicional não é relevante (SOMMERVILLE, 2011). Encontrar uma ordem em meio ao caos pode ser através da utilização das características dos processos ágeis e tradicionais.

Decidir entre um processo de desenvolvimento tradicional ou ágil depende do cenário envolvido, ou seja, depende do tipo de software que será desenvolvido, das habilidades e culturas da equipe e da organização, entre outros. Por esse motivo, eles desenvolvem o próprio

processo de desenvolvimento, desfrutando da capacidade dos envolvidos e das características dos sistemas desenvolvidos (SOMMERVILLE, 2011).

1.3 Atividades fundamentais presentes nos processos de desenvolvimento

O desenvolvimento de software independente do projeto, tamanho ou complexidade contêm algumas atividades fundamentais (SOMMERVILLE, 2011), também denominadas como atividades metodológicas genéricas (PRESSMAN, 2011), que podem ser identificadas nos processos de desenvolvimento de software.

Sommerville (2011) apresenta quatro atividades fundamentais presentes nos processos de desenvolvimento, que são: Especificação; Projeto e implementação; Validação; Evolução.

Hirama (2011) define essas atividades como: Engenharia de sistemas; Análise; Projeto; Codificação; Testes.

Schach (2009) define como: Levantamento de necessidades; Análise; Projeto; Implementação; Teste; Manutenção pós-entrega e; Retirada do produto.

Mesmo que os autores apresentem atividades com quantidades e nomenclaturas diferentes, elas contêm características similares. Nem todos os processos utilizam essas características de maneira igual, porém pode-se identificá-las.

Para que não haja confusão referente as atividades que serão apresentadas a seguir, as atividades metodológicas genéricas serão apresentadas de acordo com Pressman (2011), que são: Comunicação; Planejamento; Modelagem; Construção; Emprego.

Ele também adiciona às atividades metodológicas genéricas um conjunto de atividades de apoio, que tem o objetivo de auxiliar a equipe no gerenciamento, progresso, qualidade, mudanças e riscos do projeto, são elas: Controle e acompanhamento de projeto; Administração de risco; Garantia da qualidade de software; Revisões técnicas; Medições; Gerenciamento de configuração de software; Gerenciamento de reusabilidade; Preparo e produção de artefatos de software.

Para cada atividade metodológica genérica é apresentado um conjunto de princípios, que formam uma sustentação para o sucesso de cada atividade.

1.3.1 Comunicação

A comunicação é uma das atividades mais desafiadoras, pois é através dela que informações são compartilhadas, tornando-a essencial. “O processo de comunicação consiste na transmissão de informação entre o emissor e um receptor que descodifica (interpreta) uma determinada mensagem” (GUIMARÃES; CABRAL, 2016). Os princípios apresentados por

Pressman (2011 p. 112-113) podem parecer lógicos e racionais, porém nem sempre são utilizados.

As reuniões devem ser direcionadas por um facilitador; antes de encontrar os indivíduos é indicado que realize pesquisas a respeito do que será discutido. Interrompa somente quando o intuito da interrupção for obter mais informações e não apresente contestação ou relutância (PRESSMAN, 2011).

Reuniões face a face são mais produtivas, quando as palavras forem insuficientes, a utilização de um esboço ou representação gráfica pode facilitar o entendimento do que está sendo discutido. É interessante que um indivíduo faça anotações referentes aos pontos e decisões importantes tratadas durante a reunião (PRESSMAN, 2011).

As negociações realizadas durante as reuniões só podem ser menos onerosas se todas as partes envolvidas no projeto tiverem um único objetivo. Logo, quando houver necessidade de negociar será necessário o compromisso de todos (PRESSMAN, 2011).

A comunicação pode ser considerada como uma potente ferramenta em busca da satisfação – quando consegue-se atender à necessidade e superar as expectativas. Os responsáveis pelo levantamento de requisitos devem ser observadores, sagazes e conseguirem captar as informações que não foram ditas, pois as entre linhas carregam as reais necessidades dos clientes (LANA, 2009).

1.3.2 Planejamento

A comunicação auxilia a compressão de metas e objetivos que deseja alcançar, essas metas podem alterada no decorrer do tempo, planejar inclui técnicas práticas e gerenciais possibilitando a definição de um roteiro (PRESSMAN, 2011).

Conseguir planejar todo o projeto de software no início do processo e depois seguir esse plano até o produto ser entregue é a utopia de todos os envolvidos no processo de desenvolvimento de software, porém a realidade pode ser diferente disso (SCHACH, 2009).

Definir como deve ser o planejamento do projeto depende das pessoas envolvidas, alguns acreditam que um plano detalhado é a melhor abordagem, já que alterações podem ocorrer de maneira frequente e quanto mais detalhes forem abordados, menor será a probabilidade de se perder. Outros acreditam que um planejamento rápido abordando mais detalhes futuramente já é o suficiente (PRESSMAN, 2011).

“Como a maioria das coisas na vida, o planejamento deveria ser conduzido de forma moderada, o suficiente para servir de guia para a equipe – nem mais, nem menos”

(PRESSMAN, 2011). Planejar em demasia consome muito tempo, mas planejamento insuficiente pode levar ao caos.

Para definir o planejamento do projeto é necessário entender o objetivo do projeto. Ao estabelecer um plano seja realista, lembre-se que softwares são desenvolvidos por pessoas, podem ocorrer falhas na comunicação, omissões e ambiguidades, entre outros fatores (PRESSMAN, 2011).

Como o planejamento refere-se à definição do roteiro do projeto, e um projeto pode ser modificado, consequentemente seu planejamento também sofrerá alterações. Independente da metodologia de desenvolvimento de software utilizada, o planejamento pode sofrer alterações no decorrer do desenvolvimento, mas em processos de desenvolvimento iterativos e incrementais a modificação do planejamento pode ser mais visível a cada incremento entregue (PRESSMAN, 2011).

Inserir os interessados em tomadas de decisões auxiliando na definição das prioridades e restrições do projeto, faz com que essas decisões sejam mais assertivas. Para acomodar as prioridade e restrições definidas haverá negociações frequentes referentes ao projeto. Compreender um trabalho a ser realizado quando o trabalho é amplo pode ser muito difícil, dividi-lo em partes menores podem auxiliar a compreensão e determinar estimativas mais confiáveis (PRESSMAN, 2011).

Projetos de software podem atrasar, uma boa prática é checar seu progresso, de preferência diariamente para identificar os problemas referentes ao que foi proposto e o que está ocorrendo e caso necessário ajustar o plano. Inserir toda a equipe na atividade de planejamento garante máxima eficiência e os membros estarão envolvidos com o plano (PRESSMAN, 2011).

1.3.3 Modelagem

A modelagem refere-se à criação de modelos abstratos, cada modelo pode exibir uma visão ou perspectiva do sistema, o modelo mais utilizado atualmente é a UML (*Unified Modeling Language*) (SOMMERVILLE, 2011). A UML é um conjunto de notações gráficas – diagramações – que auxilia a análise e projeto de software, principalmente de desenvolvimento de softwares orientados a objetos (FOWLER, 2005).

O desenho ou leitura de UML tem o objetivo de tornar o que é proposto melhor compreendido através da capacidade cerebral de interpretar símbolos, unidades e relacionamentos. A ideia fundamental de qualquer diagramação é ajudar a explorar o panorama e relacionamentos de análise ou software, permitindo ignorar ou ocultar detalhes

desinteressantes, infelizmente essa ideia pode ser perdida dentre tantos detalhes e ferramentas UML (LARMAN, 2005).

Larman (2005) apresenta três modos de aplicar a UML:

- UML como rascunho: Diagramas parciais e informais utilizados para compreender partes difíceis do problema ou soluções;
- UML como planta de software: Diagramas mais detalhados a respeito do projeto, essas plantas são usadas em: engenharia reversa facilitando a compreensão do que já foi desenvolvido; geração de código fornecendo instruções para desenvolvimento.
- UML como linguagem de programação: o código executável será gerado através das especificações completas do software através da UML.

Métodos ágeis utilizam a modelagem como rascunho, mesmo as ferramentas sendo úteis Larman (2005), incentiva o uso de UML na abordagem ágil de modelagem. Ele descreve que a finalidade da modelagem não é criar vários diagramas para serem entregues para o programador e sim explorar a UML como alternativa para compreensão do problema ou resultado, apoiando o entendimento e a comunicação.

O objetivo do desenvolvimento de software é entregar software executável aos interessados, modelos que auxiliam são viáveis, já modelos que agregam pouca novidade devem ser evitados pois podem retardar o processo de desenvolvimento de software, pois modelos demandam tempo em sua elaboração e atualização tempo que poderia ser utilizado na construção do software (PRESSMAN, 2011).

Uma prática para identificar se um modelo é viável é questionar seu objetivo, se não encontrar um motivo plausível, que justifique o tempo que será despendido na sua elaboração ou atualização, não o faça (PRESSMAN, 2011).

Modelos têm um conjunto de notações e regras para sua elaboração, mas em determinados cenários não transmite a informação necessária, nesse caso, pode ser necessário adaptar as notações ou regras de acordo com o software que será construído. Não seja dogmático quanto à sintaxe do modelo. Se este transmite o conteúdo com sucesso, a representação é secundária (PRESSMAN, 2011).

O tempo despendido na construção de modelos completos e condizentes com o software construído é um esforço muito grande, considerando que o principal artefato que deve ser entregue aos interessados é o software funcional. Os modelos devem ser elaborados com o objetivo de nortear as próximas tarefas que devem ser realizadas. Apresenta-los aos interessados o mais rápido possível para se obter *feedbacks* auxiliará na correção da

modelagem, minimizando ambiguidades e acrescentando características ou funções que não haviam sido descritas (PRESSMAN, 2011).

1.3.3.1 Métodos de Modelagem de Requisitos

Pesquisadores "identificaram problemas de análise de requisitos e suas causas e desenvolveram uma série de notações de modelagem e uma série de "heurísticas" correspondentes para supera-los" (PRESSMAN, 2011, p. 117). Cada método de análise possui pontos de vista particulares, mas todos são inter-relacionados, Larman (2005 p.34, grifo do autor) descreve que a análise "[...] enfatiza a *investigação* do problema e dos requisitos, em vez de uma solução". Os princípios referentes aos métodos de requisitos são apresentados por Pressman (2011, p. 117 – 118):

- O universo de informações de um problema deve ser representado e compreendido;
- As funções que o software desempenha devem ser definidas;
- O comportamento do software (como consequência de eventos externos) devem ser representados;
- Os modelos que descrevem informações, funções e comportamentos devem ser divididos para que revelem detalhes por camadas (ou hierarquicamente);
- A análise deve partir da informação essencial para o detalhamento da implementação. A modelagem de requisitos se inicia;

1.3.3.2 Modelagem de Projetos

A modelagem de projetos proporciona vários focos do mesmo sistema e "tem por objetivo definir uma estrutura implementável para um produto de software que atenda aos requisitos especificados para ele" (PAULA FILHO, 2003).

Os métodos utilizados para identificar as características do projeto são: voltados a dados, onde a arquitetura e os componentes são determinados pela estrutura dos dados; voltados a padrões, usando as informações referentes a modelagem do projeto para definir os estilos arquitetônicos e os padrões de processamento; voltados a objeto, usando objetos do âmbito do problema para criar os métodos e as estruturas que os manipularão (PRESSMAN, 2011). Pressman (2011, p. 118 - 119) apresenta os princípios da modelagem de projeto independentemente do método utilizado, são eles:

- O projeto deve ser roteirizado para a modelagem de requisitos;
- Sempre considere a arquitetura do sistema a ser construído;

- O projeto de dados é tão importante quanto o projeto de funções de processamento;
- As interfaces (tanto internas quanto externas) devem ser projetadas com cuidado;
- O projeto de interface do usuário deve ser voltado às necessidades do usuário final. Entretanto, em todo caso, deve enfatizar a facilidade de uso;
- O projeto no nível de componentes deve ser funcionalmente independente;
- Os componentes devem ser relacionados livremente tanto entre componentes quanto com o ambiente externo;
- Representações de projetos (modelos) deve ser de fácil compreensão;
- O projeto deve ser desenvolvido interativamente. A cada iteração, o projetista deve se esforçar para obter maior grau de simplicidade.

1.3.4 Construção

Construir o software engloba as atividades de codificação e testes (PRESSMAN, 2011, p. 120 - 122). A codificação pode ser através de:

> (1) criação do código-fonte da linguagem de programação [...] (2) a geração automática de código-fonte usando uma representação intermediária semelhante a um projeto do componente a ser construído[...](3) a geração automática de código executável usando uma "linguagem de programação de quarta geração (PRESSMAN, 2011, p.120).

1.3.4.1 Codificação

A codificação está relacionada ao estilo de programação, linguagens e métodos de programação. Antes de iniciar o processo de codificação do software deve-se ter compreendido o problema que estará solucionando; saber os princípios e conceitos do projeto; escolher qual a linguagem de programação que será utilizada levando em consideração a necessidades do software e o ambiente em que ele irá operar; ter selecionado as ferramentas que auxiliarão seu trabalho, tornando-o mais fácil e determinando um conjunto de testes de unidades[1] que serão realizados após a codificação completa dos componentes (PREESMAN, 2011).

Estruture os algoritmos de maneira organizada; se possível, tente introduzir a programação em pares analisando a viabilidade de sua utilização; determine a estrutura de dados de acordo com a estrutura do projeto; compreenda a arquitetura de software e elabore interfaces consistentes; conserve as estruturas condicionais simples; ao implementar laços, deixe-os agrupados facilitando a aplicação de testes; Mantenha o nome de variáveis e métodos

[1] "[...] testa individualmente os módulos ou componentes de software desenvolvidos – testa-se o módulo ou componente que realmente executa a tarefa para a qual foi projetado" (HIRAMA, 2011).

consistentes e significativos obedecendo padrões de codificação; faça com que o artefato seja auto documentado; Idente e alinhe seu código hierarquicamente. Essas práticas facilitarão a inserção, alteração e compreensão do que está sendo codificado (PREESMAN, 2011).

Após a codificação do que foi proposto revise o código, ou melhor, faça a revisão do código sempre que for necessário; realize teste unitário e corrija os erros que forem identificados; sempre tenha em mente que o código é propriedade e responsabilidade sua, ou seja, refaça a codificação do que for necessário (PREESMAN, 2011).

1.3.4.2 Testes

Um software que está em desenvolvimento deve ser verificado, validado e testado[2]. A verificação investiga se o software está sendo desenvolvido de acordo com suas especificações; a validação tem como objetivo identificar se o software atende as expectativas do cliente; o teste implica na execução da codificação do software com dados de teste, averiguando a saída e o comportamento do software, o teste é uma estratégia dinâmica da verificação e validação (SOMMERVILLE, 2007).

> Você somente pode testar um sistema quando um protótipo ou uma versão executável do programa está disponível. Uma vantagem do desenvolvimento incremental é que uma versão passível de teste do sistema está disponível em estágio bem adiantado no processo de desenvolvimento. A funcionalidade pode ser testada conforme é adicionada ao sistema, dessa maneira você não precisa ter uma implementação completa antes de iniciar os testes (SOMMERVILLE, 2007, p. 342).

"O objetivo do teste é encontrar defeitos, revelando que o funcionamento do software em uma determinada situação não está de acordo com o esperado. Um teste bem-sucedido identifica defeitos que ainda não foram descobertos" (KOSCIANSKI; SOARES. 2007, p. 337). Para isso deve-se organiza-los de maneira ordenada, buscando abranger diferentes tipos de erros, utilizando o mínimo esforço e tempo.

A realização de testes pode envolver testes manuais e testes automatizados. Os testes manuais consistem na execução do software por um indivíduo, onde é inserido informações de entrada e comparados os resultados, as discrepâncias identificadas são informadas aos desenvolvedores. Os testes automatizados, são testes codificados e executados durante o desenvolvimento de software, verificando se o software realiza o que foi proposto. Eles são úteis quando é necessário realizar testes de regressão[3] (SOMMERVILLE, 2011).

---

[2] Koscianski, Soares (2007) e Hirama (2011) consideram a verificação e a validação como técnicas estáticas e teste como técnicas dinâmicas.

[3] Teste de regressão consiste na reexecução de testes quando ocorre uma correção ou inserção de uma nova funcionalidade, seu intuito é averiguar se a modificação não ocasionou um defeito (SOMMERVILLE, 2011).

Os pontos críticos na visão dos clientes referem-se as falhas relacionadas aos seus requisitos, portanto, os testes funcionais devem concentrar nos requisitos. Deve-se definir uma estratégia para testar o software que está sendo desenvolvido, antes mesmo que qualquer codificação tenha sido realizada. A definição de quais testes serão realizados pode ser estabelecido assim que o modelo de projeto esteja consolidado (PREESMAN, 2011).

No contexto de teste de software o princípio de Pareto refere-se que 80% dos erros possivelmente estão relacionados com 20% dos componentes. Para identificar os erros deve-se isolar os componentes duvidosos e testa-los por completo. Os testes devem ser realizados dos micros aos macros componentes, ou seja, deve-se planejar e realizar testes focando em componentes individuais; depois em grupos (componentes integrados); após realizar os testes em todos os componentes integrados deve-se testar o sistema como um todo (PREESMAN, 2011).

2 MODELO DE PROCESSO DE DESENVOLVIMENTO

O processo de desenvolvimento de software tem o objetivo de nortear o desenvolvimento. Um processo não é algo imutável, pelo contrário, pode ser modificado (PRESSMAN, 2011).

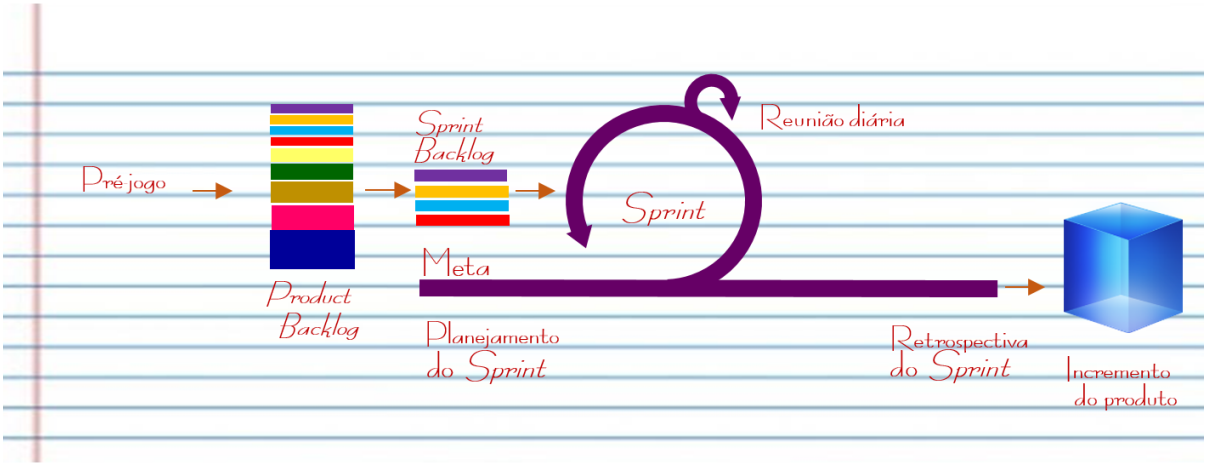
Sommerville (2011) acrescenta que não existe um processo ideal, logo, as empresas desenvolvem seu próprio processo de desenvolvimento de software para tirar melhor proveito dos indivíduos e das características dos sistemas que serão desenvolvidos.

Ao identificar o problema em questão, uma das características do software que está sendo desenvolvido é a incerteza, pois seu intuito é apresentar uma solução referente a um problema presente em lanchonetes e restaurantes que utilizam a comanda ou cartão para controle de entrada e saída. Ele não possuiu um cliente específico, pois o objetivo da equipe é produzir o produto mínimo viável, apresentando alguma solução referente ao problema e ir incrementando novas funcionalidades que agreguem valor ao produto.

Baseado nesse cenário, os modelos de processos mais indicados são as metodologias ágeis, no processo de desenvolvimento adotado pode-se identificar principalmente as características do *Scrum* que Sommerville (2011) descreve como uma metodologia ágil com foco no gerenciamento do desenvolvimento de software iterativo.

O processo de desenvolvimento foi adaptado a partir de um processo apresentado na obra “Scrum: Gestão ágil para projetos de sucesso” do autor Rafael Sabbagh. Onde são apresentados o *framework Scrum* juntamente com outros artefatos, marcos e métodos que podem ser utilizados para auxiliar o processo de desenvolvimento.

Figura 2 - Modelo do processo de desenvolvimento



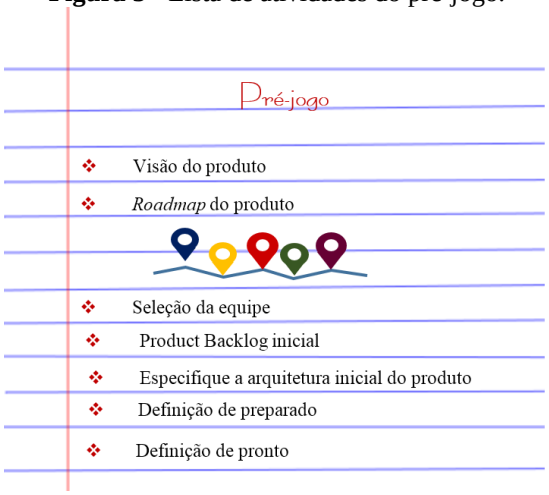
Fonte: Adaptado pelos autores (SCHWABER apud SABBAGH, 2014, p.41).

O modelo de processo apresentado (Figura 2) refere-se ao processo de desenvolvimento utilizado pela equipe até o momento. Os autores optaram pela utilização de um processo de desenvolvimento mais enxuto e ir incrementando atividades, ações e tarefas de acordo com o avanço do produto que está sendo desenvolvido.

2.1 Pré-jogo

O pré-jogo refere-se às atividades iniciais onde são realizados definições e preparativos básicos referente ao produto e ao processo, essa fase não possui tempo definido, pois depende do quanto é preciso definir ou preparar (SABBAGH, 2014). A Figura 3 apresenta o que deve ser realizado durante essa fase.

Figura 3 - Lista de atividades do pré-jogo.



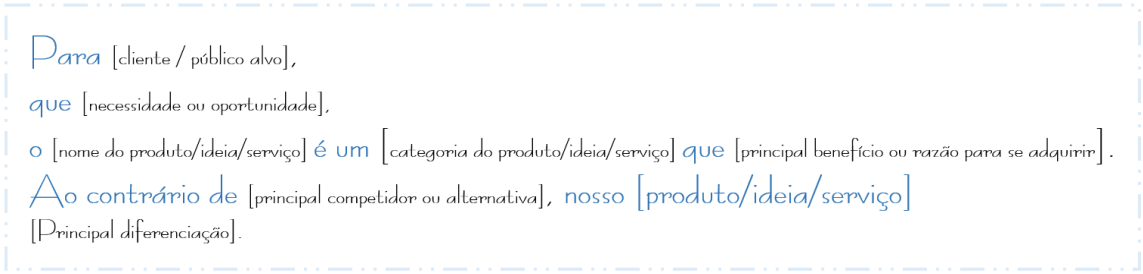
Fonte: Elaborado pelos autores (SABBAGH, 2014).

2.1.1 Visão do produto

A visão do produto deve ser definida antes de iniciar o desenvolvimento de software. Seu objetivo é prover um alinhamento entre os envolvidos no projeto e deve ser de fácil compreensão, conciso e claro, contendo apenas o necessário. O *Product Owner* é responsável pela sua criação, manutenção e divulgação, além de garantir que o *Product Backlog* esteja sempre alinhado com a visão do produto estabelecida (SABBAGH, 2014).

O modelo apresentado por Sabbagh (2014) para a elaboração da visão do produto é denominado como “Teste do elevador”, também conhecido como *Elevator Statement* ou *Elevator Pitch*. O teste do elevador (Figura 4) tem como objetivo descrever de maneira sucinta a solução que está sendo proposta.

Figura 4 - Teste do elevador



Fonte: Adaptado pelos autores do exemplo apresentado por Sabbagh (2014).

A criação da visão do produto faz com que todos os envolvidos tenham uma visão homogênea do sistema que está sendo desenvolvido.

2.1.2 Roadmap do produto

Considerado pelo site Endeavor Brasil (2015), como uma "bússola gerencial" o *roadmap* do produto apresenta de maneira descritiva e visual a evolução do projeto. A partir da visualização do *roadmap* as seguintes questões podem ser respondidas: Onde estamos? Onde queremos ir? Onde chegaremos?

É de responsabilidade do *Product Owner* sua criação, geralmente é apresentada por uma linha contínua, representando como será a evolução do produto contendo uma data estimada ou exata que pode ou não ser alcançada (SABBAGH, 2014).

2.1.3 Seleção da equipe de desenvolvimento

Durante o pré-jogo Sabbagh (2014) incentiva a definição do time *Scrum* que irá trabalhar no desenvolvimento do projeto, dependendo da organização pode ser que essa definição não seja necessária. O time *Scrum* é composto por um *Product Owner*, o time de desenvolvimento e um *Scrum Master*.

Como o projeto que está sendo desenvolvido só contém dois integrantes, definiu-se que um deles seria o *Product Owner* e o outro seria responsável pelo desenvolvimento, ou seja, seria o time de desenvolvimento.

2.1.4 *Product Backlog* inicial

A partir da visão do produto pode-se iniciar a criação do *Product Backlog* que irá evoluir com o decorrer do projeto, onde os itens podem ser retirados, adicionados e reordenados e modificados (SABBAGH 2014). "O *Product Backlog* inicial pode ser longo, contendo desde itens pequenos e bem detalhados, até itens grandes e vagos. Mas ele também pode conter apenas uma quantidade de itens necessária para se iniciar o desenvolvimento" (SABBAGH, 2014, p.41).

2.1.5 Especifique a arquitetura inicial do produto

Para reduzir riscos referentes a tomadas de decisões tardias que possam invalidar o que já foi produzido, uma boa prática é especificar uma arquitetura básica, onde essa especificação deve ser o suficiente para minimizar os riscos, entretanto deve-se tomar cuidado para não engessar o desenvolvimento do produto (SABBAGH, 2014).

2.1.6 Definição de preparado

A definição de preparado é uma lista de critérios ou condições acordadas entre o *Product Owner* e o Time de desenvolvimento, com o objetivo de reduzir os riscos do *Sprint* mal planejada, logo, para um item do *Product Backlog* ser discutido no planejamento do *Sprint* é necessário que este item esteja de acordo com a definição de preparado (SABBAGH, 2014).

A equipe definiu que para um item do *Product Backlog* ser discutido na Reunião do *Sprint* é necessário criar histórias de usuário tendo em mente o acrônimo *INVEST* (investir) apresentado por Wake (2003, p.1):

- Independente (*Independent*): As histórias de usuário devem ter o mínimo de dependência;
- Negociável e negociada (*Negotiable*): Uma história de usuário descreve a essência, a comunicação incrementará às histórias de usuário com detalhes necessários;
- Valiosa (*Valuable*): As histórias de usuário devem ser valiosas aos clientes, os desenvolvedores devem enquadrar suas preocupações e responsabilidades de forma que o cliente perceba a importância;
- Estimável (*Estimable*): Uma história de usuário deve ser estimável, para isso ela deve ser negociável e pequena o suficiente de forma que atribua descrições suficientes para auxiliar o desenvolvimento;
- Pequena (*Small*): Histórias de usuário pequenas facilitam a compreensão e incentivam a comunicação dos interessados. Podendo também auxiliar o time de desenvolvimento a estimar seu tempo;
- Testável (*Testable*): Uma história de usuário pode ser verificada; é incentivada a prática de que clientes (no nosso contexto o *Product Owner*), escrevam testes antes que o desenvolvimento seja iniciado.

Além disso, devem ser inseridos critérios e teste de aceitação. Os critérios para aceitação inicial devem ser descritos pelo *Product Owner* e apresentados na reunião de planejamento do *Sprint.* Após examinar a história de usuário e os critérios de aceitação e realizar

as alterações necessárias, o *Product Owner* descreve alguns cenários de testes de aceitação, utilizando BDD e responde as dúvidas que aparecerem.

2.1.7 Definição de pronto

A definição de pronto depende do projeto e da equipe de desenvolvimento de software consiste em uma lista de atividades definida de acordo com as necessidades do produto, juntamente com as atividades que serão observadas (SABBAGH, 2014). Essa lista é mantida e compartilhada nesse contexto pelo *Product Owner* e o Time de desenvolvimento.

Foi elaborado um fluxo geral apresentando as atividades, ações e tarefas referentes a definição de pronto do sistema CloudGourmet (APÊNDICE A). As ferramentas selecionadas pelo time de desenvolvimento são apresentadas no capítulo 4.

O *Product Owner*:

- Elabora as histórias de usuários;
- Acrescenta os critérios de aceitação e alguns testes de aceitação, de acordo com as negociações realizadas;

O Time de desenvolvimento para cada história de usuário (APÊNDICE B):

- Utiliza o gerenciador de configuração GIT é criado um novo ramo (*Branch*) a partir do repositório principal (*Master*) para que a tarefa seja desenvolvida;
- Analisa a história de usuário, critérios e testes de aceitação para só então desenvolver;
- Cria o código na linguagem de programação Ruby e os testes automatizados, tendo em vista o que foi descrito nos critérios de aceitação;
- E envia o código criado para o repositório remoto que automaticamente inicia um processo de construção (build) e este cria o ambiente para que sejam executados os testes automatizados;
- Notifica o *Product Owner* que os testes automatizados foram executados com sucesso e que ele já pode iniciar testes de caixa preta;
- Por fim o desenvolvedor pode realizar outra tarefa enquanto atividade anterior não é testada.

Product Owner:

- Ao receber do desenvolvedor a notificação do sucesso dos testes automatizados o *Product Owner*, através da plataforma Wercker de integração contínua (*Continuous Integration*), irá iniciar o processo de *deploy* para o ambiente de testes (APÊNDICE C);

- Para testar a atividade do desenvolvedor o *Product Owner* deverá:
  - Analisar a história de usuário, critérios e testes de aceitação vinculados as tarefas do desenvolvedor;
  - Executar cada critério de aceitação;
  - Avaliar se a experiência de usuário está satisfatória. Caso não esteja ele deve propor melhorias;
  - Por fim o *Product Owner* deve notificar o time de desenvolvimento as não conformidades, caso estas não existam ele deve informar que os critérios de aceitação foram satisfeitos.

Time de desenvolvimento

- Ao receber a notificação de os critérios de aceitação foram satisfeitos, o desenvolvedor irá criar uma solicitação para mesclar o código da atividade com o código principal (*Pull Request*), na *Branch Master*.
- Antes de ser mesclado, o desenvolvedor deverá analisar o código criado durante a atividade buscando possibilidades de otimização e erros de padronização, tendo em vista a convenção da comunidade Ruby on Rails.
- Caso não seja encontrada nenhuma inconformidade o código deverá ser mesclado (*Merge*) com a *Branch Master* e isso irá automaticamente iniciar um processo de construção no Wercker (Apêndice D).

E finalmente caso o processo de construção seja realizado com sucesso o Wercker irá iniciar o *deploy* automaticamente para o ambiente de produção, pois foi configuradora para realizá-lo automaticamente.

2.2 Product Backlog

O *Product Backlog* é uma lista que contém as necessidades ou objetivos do projeto, podendo conter melhorias, correções de problemas identificados nos incrementos, questões técnicas, pesquisas necessárias, entre outros itens. É ordenado de acordo com o grau de importância, gradualmente detalhado, ou seja, os itens no topo devem conter mais detalhes, os itens na parte inferior são de menor granularidade, ou seja, com menos detalhes (SABBAGH, 2014).

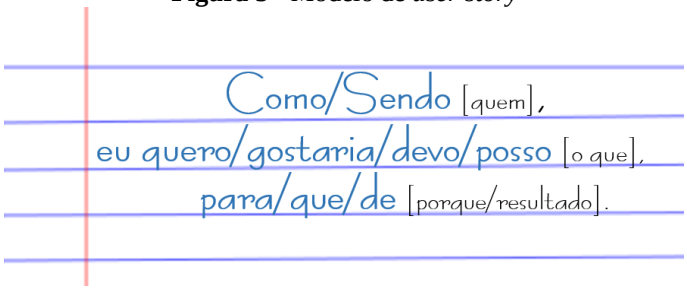
Os seguintes itens podem influenciar na priorização do *Product Backlog* são eles (BERNARDO, 2015, p. 1):

- Prioridade do cliente;
- Urgência em receber feedback;

- Dificuldade de implementação relativa;
- Relações entre itens de trabalho.

Os itens do *Product Backlog* serão descritos através de *user story* (história de usuário) (Figura 5). Uma história de usuário deve explicar para quem, o que e por que está sendo criada. Sabbagh (2014) indica que as questões técnicas, correções de problemas e pesquisas necessárias sejam embutidas nas histórias de usuários.

Figura 5 - Modelo de *user story*



Fonte: Adaptado pelos autores do exemplo apresentado por Bernardo (2014).

Os detalhes referentes ao negócio do cliente podem ser documentados de diversas maneiras, dependendo da complexidade do projeto, da equipe e da comunicação entre os indivíduos. Até o momento os documentos que a equipe decidiu utilizar são os Critérios e Testes de aceitação.

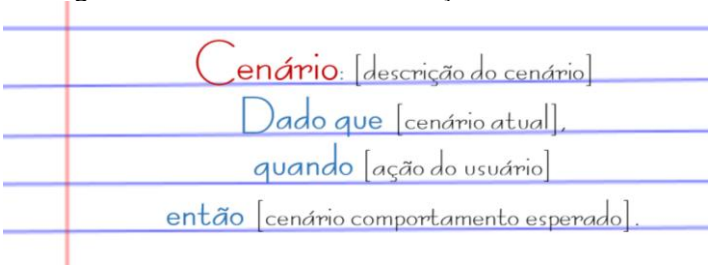
2.2.1 Critérios de aceitação

Os critérios de aceitação são uma lista de itens que serve para identificar quando uma funcionalidade está completa, essa lista é criada pelo *Product Owner*, negociada e corrigida quando os critérios são apresentados ao time de desenvolvimento.

2.2.2 Teste de aceitação

A partir dos critérios de aceitação o *Product Owner* define alguns testes de aceitação utilizando a técnica BDD (*Behaviour Driven Developmet*) (Figura 6). Ao apresentar os testes de aceitação ao time de desenvolvimento, eles podem ser melhorados ou retirados. Os testes de aceitação utilizando BDD descrevem cenários que indicam o comportamento esperado.

Figura 6 - Modelo de teste de aceitação utilizando BDD.



Fonte: Adaptado pelos autores do exemplo apresentado por Helme e Wildt (2016).

"Para cada *user story* existe um número infinitamente grande de testes de aceitação. A seleção do mais simples é uma arte que pode ser dominada em alguns minutos" (ASTELS; MILLER; NOVAK, 2002, p. grifo do autor). Escrever todos os testes de aceitação possíveis não será viável, o objetivo da elaboração dos testes anexados a estórias de usuários é auxiliar a compreensão do comportamento do sistema.

2.3 Sprint

O *Sprint* é o ciclo de desenvolvimento, onde o incremento do produto segundo a definição de pronto é gerado pelo Time de Desenvolvimento a partir dos itens mais importantes do *Product Backlog* (SABBAGH, 2014).

No projeto em questão, além do trabalho de desenvolvimento dentro do *Sprint* serão realizadas as reuniões de planejamento, reuniões diárias e retrospectiva do *Sprint.* Os *Sprints* geralmente têm um tempo de duração fixa, essa regularidade busca auxiliar o ritmo do time de desenvolvimento e pode facilitar a agenda do *Product Owner* com as partes interessadas (SABBAGH, 2014).

No processo de desenvolvimento adotado até o momento, não serão realizadas as revisões dos *Sprints,* pois o produto não tem um cliente e o *Product Owner* realiza testes referentes as funcionalidades criadas durante a execução do *Sprint* para verificar e validar o que foi desenvolvido juntamente com a equipe.

Outros marcos deverão ser inseridos no processo de desenvolvimento de software, quando houver mudança de cenário, como por exemplo a inserção de novos integrantes no time de desenvolvimento; isso acarretará a inserção de novos artefatos, métodos e meios de comunicação.

2.3.1 Planejamento do *Sprint*

O planejamento do *Sprint* é o primeiro marco de um *Sprint* e durante essa reunião ocorre o planejamento do trabalho que será realizado dentro do *Sprint*. O time de desenvolvimento juntamente como *Product Owner* determinam quais itens no topo do *Product Backlog* que serão desenvolvidos durante o *Sprint,* esses itens devem estar de acordo com a definição de preparado, assim a reunião de planejamento pode se tornar produtiva e eficiente (SABBAGH, 2014).

Após negociar os itens que serão desenvolvidos no *Sprint,* o *Product Owner* e o time de desenvolvimento definem a meta do *Sprint*, essa meta não pode ser alterada, a meta do *Sprint* "guia, dá propósito, motiva e mantém o foco do trabalho do Time de desenvolvimento no *Sprint"* (SABBAGH, 2014, p.162, grifo nosso).

Os itens negociados serão inseridos no *Sprint Backlog*, como o time de desenvolvimento contém apenas um integrante optou-se em não quebrar os itens em um conjunto de tarefas.

2.3.2 Reunião diária

A reunião diária, trata-se de uma reunião que ocorre entre os integrantes do time de desenvolvimento e o *Scrum Master*. O *Product Owner* não participa dessa reunião, para não interferir no andamento do *Sprint* (SABBAGH, 2014), podendo apenas observa-la.

A equipe decidiu manter esse nome, porém no cenário atual o time de desenvolvimento informa ao *Product Owner* como está o processo de desenvolvimento do *Sprint*, assim todos os envolvidos ficam informados.

2.3.3 Retrospectiva do *Sprint*

Durante a retrospectiva do *Sprint* a equipe se reúne para identificar pontos que podem ser melhorados e planejam como essas melhorias podem ser executadas. São levantados os pontos positivos e os pontos negativos identificados durante o *Sprint*, não é correto realizar acusações ou discussões improdutivas. A busca por melhorias antes mesmo da iniciação do próximo *Sprint* se torna mais produtiva do que avaliar lições aprendidas ao final de um projeto (SABBAGH, 2014).

3 SISTEMA CLOUDGOURMET

3.1 Pré-jogo

A fase inicial, denominada como pré-jogo teve a duração de um mês. Os autores tiveram dificuldades de implantar as práticas de um desenvolvimento iterativo e incremental, mesmo tendo o conhecimento básico a respeito de sua utilização e sabendo que o melhor meio de desenvolver o software seria utilizando métodos ágeis.

Essa dificuldade ocorreu, pois, queriam absorver o máximo de informações possíveis antes de desenvolverem o produto. Essa prática só foi alterada no momento em que começaram a se perguntar: “Isso realmente é necessário? ” E perceberam que essa é uma pergunta que só poderia ser respondida pelos clientes e usuários do produto.

A partir dessa concepção, foi iniciada a definição do processo de desenvolvimento, onde seria realizado a criação de um produto com escopo simplificado, para ser lançado no mercado e a partir do feedback relatados pelos clientes, vão sendo descritas novas histórias de usuários para serem incrementadas, agregando valor ao produto.

3.1.1 Visão do produto

Para lanchonetes e restaurantes que utilizam comanda e/ou cartão para controlar a entrada e saída de seus clientes e desejam utilizar as vantagens tecnológicas vigentes para agregar valor ao seu negócio.

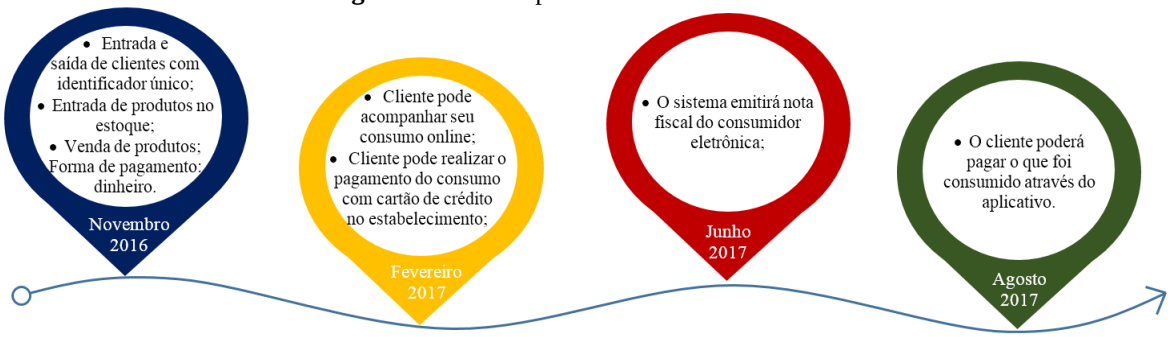
O CloudGourmet é um software web que propõe a utilização de um identificador único para os clientes desses estabelecimentos, o que pode auxiliar o controle de entrada e saída minimizando erros advindos da utilização dos métodos atuais.

Ao contrário de sistemas que não utilizam identificadores únicos para controle de entradas e saídas, nosso produto vinculará o histórico de consumo ao identificador único, então o cliente poderá verificar o que foi consumido nos estabelecimentos que utilizarem o serviço.

3.1.2 Roadmap do produto

O *Roadmap* (Figura 7) do produto foi definido pela equipe após levantarem questionamentos de como desejavam que o sistema CloudGourmet evoluísse nos meses seguintes.

Figura 7 - Roadmap do sistema CloudGourmet



Fonte: Elaborado pelos autores.

3.1.3 Equipe de desenvolvimento

A equipe de desenvolvimento é composta somente por dois integrantes, são eles: ***Product Owner:*** Noemi Mendes Pimentel; **Time de desenvolvimento:** Josué Henrique Ferreira da Silva. As atividades que cada um irá realizar individualmente ou em conjunto são apresentadas na Figura 8.

Figura 8 - Responsabilidades da equipe

Product Owner	***Product Owner*** + **Time de desenvolvimento**	**Time de desenvolvimento**
• Mantem a visão do produto • Mantem *roadmap* de produto • Mantem *Product Backlog* do produto • Define o produto que será desenvolvido • Fica disponível para fornecer feedback para o time • Acompanha o progresso do projeto • Define como será realizada as entregas • Determina se um *Sprint* atingiu a meta	• Definição de preparado • Definição de pronto • Planejamento do *Sprint* • Realizam o refinamento do *Product Backlog* • Definem os critérios de aceitação • Definem os testes de aceitação • Realiza os testes de aceitação	• Se auto-organiza • Se auto-gerencia • Define a arquitetura do projeto • Estima as histórias de usuário • Transformam histórias de usuário em incrementos • Informa o progresso da Sprint

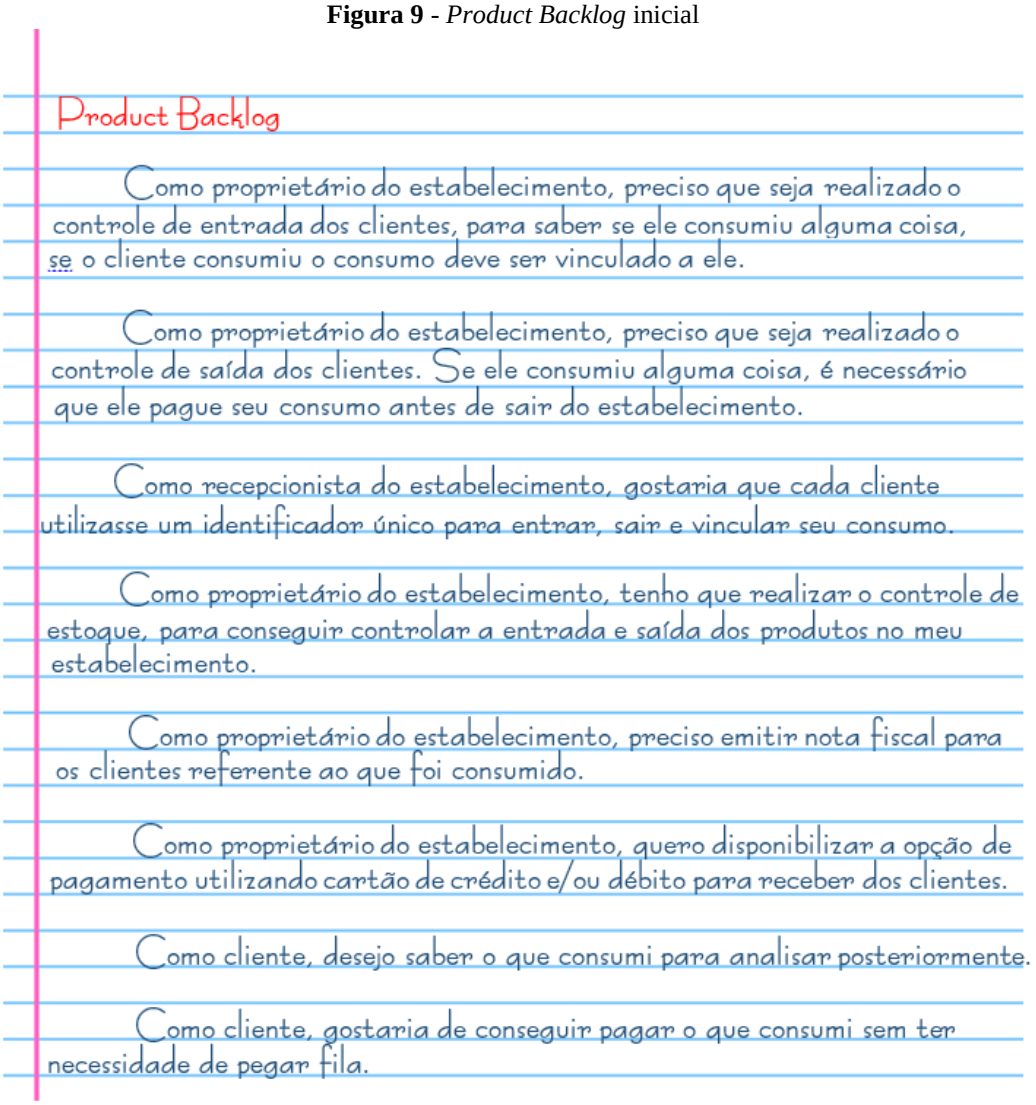
Fonte: Elaborado pelos autores.

3.1.4 *Product Backlog* inicial

O sistema CloudGourmet é uma proposta para solucionar um problema que foi identificado pela equipe e não possui um contratante. A realização do levantamento dos itens do *Product Backlog* é através de observações presenciais em restaurantes e lanchonetes, levantando as possíveis necessidades e investigando de maneira informal. Outro meio de obtenção de informação é através do ciberespaço, onde os autores buscam respostas as questões levantadas.

Ao analisar os cenários, o *Product Owner* identifica quem serão os usuários finais que se beneficiarão da funcionalidade e escreve as histórias de usuário. A ordenação dos itens do *Product Backlog* é realizada pela equipe. O *Product Backlog* inicial (Figura 9) apresenta um conjunto de itens considerados épicos, ou seja, histórias de usuário grandes, difíceis de mensurar que devem ser quebradas em histórias de usuário menores.

Figura 9 - *Product Backlog* inicial



Fonte: Elaborado pelos autores.

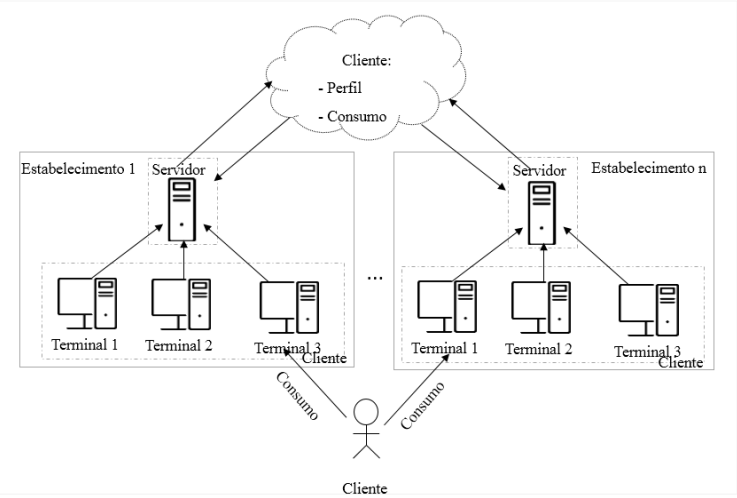
Como pode ser analisado, o *Product Backlog* inicial (Figura 9) não foi realizado com o nível de detalhamento necessário para inicial o desenvolvimento, nem descreve todas as funcionalidades que podem ser necessárias no sistema, pois o objetivo inicial é auxiliar a compreensão do que deve ser desenvolvido.

3.1.5 Arquitetura inicial

A partir da visão, *Roadmap e Product Backlog* do sistema. O time de desenvolvimento se reuniu juntamente com o *Product Owner* para determinar a arquitetura (Figura 10) referente ao sistema que seria desenvolvido.

Durante essa etapa, foi analisado qual seria um modelo de arquitetura viável para o desenvolvimento do projeto, onde o time de desenvolvimento decidiu que seria desenvolvido um sistema web pois os clientes que decidissem utilizar o sistema poderiam utilizar um servidor local ou remoto.

Figura 10 - Arquitetura inicial CloudGourmet



Fonte: Elaborado pelos autores.

O cliente vai aos estabelecimentos que utilizam o CloudGourmet e consome um produto. O recebimento é realizado nos terminais de atendimento em cada estabelecimento e esse registro é enviado para o servidor (local ou remoto) do estabelecimento. E o cliente pode, posteriormente, realizar o acesso dos dados de consumo que realizou em cada estabelecimento através do Aplicativo Mobile que será futuramente desenvolvido.

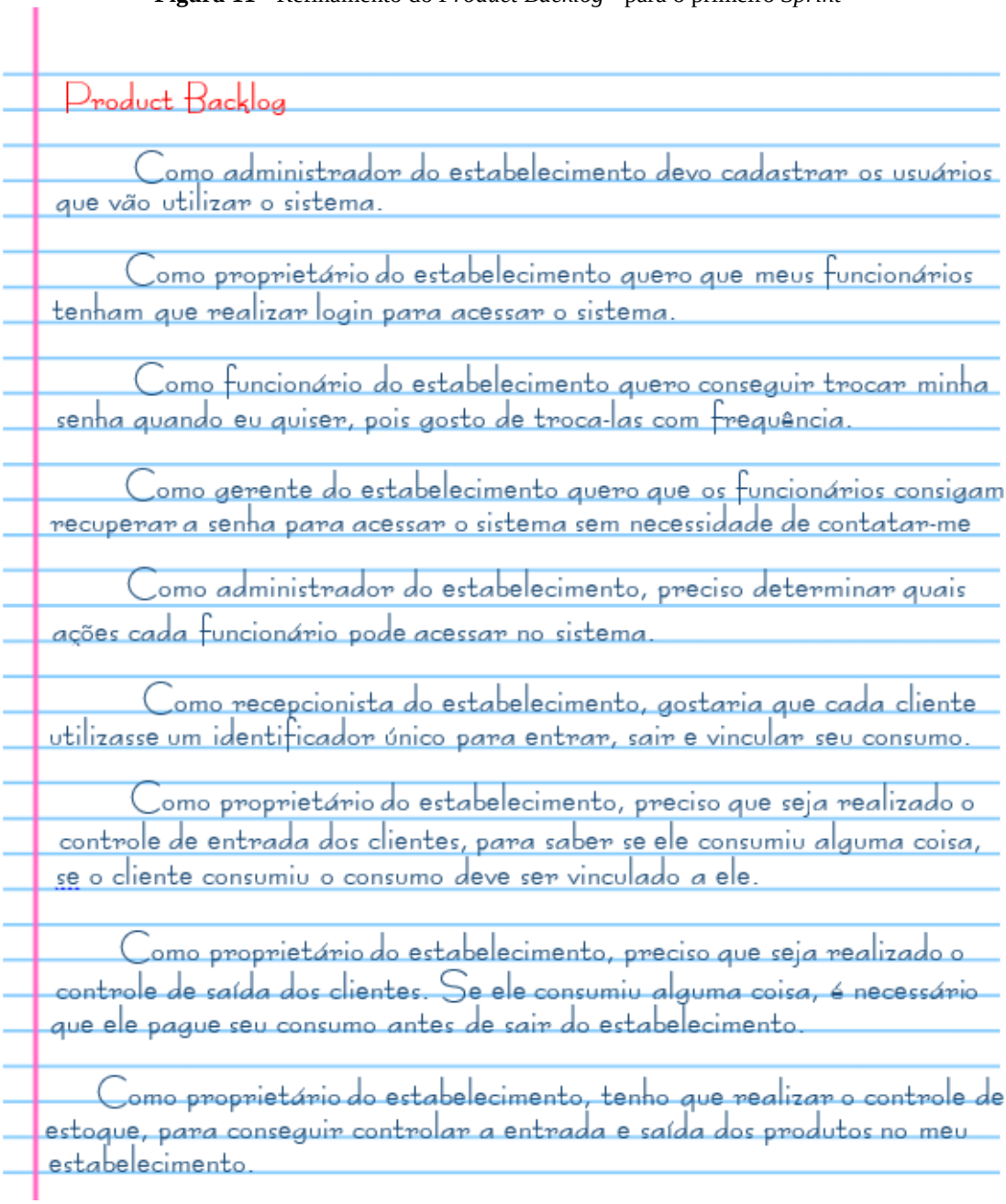
Baseado no primeiro marco do *Roadmap* do produto, para desenvolver as funcionalidades necessárias o time de desenvolvimento optou por utilizar a linguagem de programação Ruby, *Framework* Rails e o Sistema Gerenciador de Banco de Dados PostgreSQL. Outras ferramentas foram inseridas após iniciar o desenvolvimento e são descritas no capítulo 4.

3.2 Refinamento do *Product Backlog*

Após realizar o pré-jogo, a equipe se reuniu para realizar o refinamento do *Product Backlog* (Figura 11), onde foram analisadas as histórias de usuários épicas (Figura 9).

Ao analisá-las a equipe identificou que a história de usuário que agregaria valor ao produto inicialmente seria relacionada ao controle de entrada, saída e consumo vinculados a um identificador único, porém para implementar essa funcionalidade seria necessário adquirir um leitor biométrico. Por esse motivo não poderia ser inserida nos *Sprints* histórias de usuários relacionadas ao leitor biométrico enquanto ele não fosse adquirido.

Figura 11 - Refinamento do *Product Backlog*[4] para o primeiro *Sprint*



Fonte: Elaborado pelos autores

.

Durante essa reunião foi inserido ao *Product Backlog* histórias de usuários relacionadas ao *login*, controle de permissão, após as negociações as atividades referentes a essas histórias foram colocadas no topo do *Product Backlog.*

O time de desenvolvimento informou ao *Product Owner* que a preparação do ambiente seria realizada durante os primeiros *Sprints* e solicitou que fosse escolhido um *template front-end* antes do primeiro *Sprint*, pois havia um déficit referente a elaboração de *front-end.*

[4] O *Product Backlog* apresentado contém apenas os itens relacionados ao primeiro marco do *Roadmap.*

Após a finalização da reunião de refinamento o *Product Owner* selecionou alguns *templates* e apresentou ao time, que selecionou o *template* Complete Admin[5].

3.3 Sprints

Antes de realizar o planejamento do *Sprint* a equipe se reúne para realizar o refinamento do *Product Backlog.*

Após o refinamento do *Product Backlog*, a equipe realiza o planejamento do *Sprint*. Os encontros são realizados na biblioteca da universidade, no período noturno, pois trabalham e residem em cidades diferentes. Quando o time tem uma dúvida referente ao que será desenvolvido ou deseja que o *Product Owner* teste, eles utilizam o *whatsapp, hangouts* ou *facebook,* com o intuito de facilitar a comunicação.

Para facilitar a interpretação das histórias e seus critérios de aceitação, a equipe utiliza papel para especificá-las. A medida que as informações são absorvidas o *Product Owner* apresenta outros critérios e o time de desenvolvimento acrescenta, questiona e solicita a remoção. Após validar os critérios são descritos alguns testes de aceitação. Esse ciclo é realizado até ser apresentada as histórias de usuários selecionadas para o *Sprint Backlog.*

As funcionalidades desenvolvidas até o momento podem ser acessadas através do link: “http://cloudgourmet.herokuapp.com” com o CPF: “83874256219” e senha: “102030”. O repositório do ambiente de produção encontra-se disponível no link: https://bitbucket.org/tcc_n_j/cloud_gourmet_public/src.

3.3.1 *Sprint* #1 - Possibilitaremos que os usuários tenham acesso ao sistema

3.3.1.1 Reunião de planejamento do *Sprint* #1

Na reunião de planejamento desse *Sprint* o *Product Owner* apresentou os itens do *Product Backlog*, após seu refinamento. O time selecionou 4 histórias presentes no alto do *Product Backlog*, negociou a respeito das histórias e descreveu seus critérios e testes (APÊNDICE E). O time de desenvolvimento estimou durante a reunião de planejamento que o *Sprint* teria duração de 2 semanas.

3.3.1.2 Desenvolvimento do *Sprint #1*

No início do projeto era necessário que o time estabelecesse o básico necessário para começar a desenvolver de modo que, à medida que fosse necessário, outras ferramentas fossem adicionadas. A arquitetura deveria dar suporte à novos componentes. Sendo assim,

[5]*Template* Complete Admin – Bootstrap. <Disponível em: http://wrapbootstrap.com/preview/WB0K88500>. Acesso em: 19 ago. 2016.

inicialmente o projeto foi desenvolvido utilizando a arquitetura base que o Rails oferece para uma aplicação simples que é o modelo MVC (*Model, View e Controller*).

Optando por trabalhar com o Rails, o time de desenvolvimento tinha que integrar a aplicação com o *template* escolhido de forma que houvesse uma reutilização de código e padronização das telas exibidas dentro do que se tinha de ações básicas em cada recurso acessado (criar, editar, inativar e ativar).

O processo de integração envolveu estudo do *template*. Foi um momento que necessitou, em primeiro momento, que o time extraísse os *assets (javascripts e stylesheets)* comuns a qualquer página acessada para dentro do layout principal do Rails. Fazendo isso, para qualquer recurso acessado esses arquivos seriam carregados.

Logo em seguida, para facilitar o desenvolvimento, alguns dos principais componentes do *template* foram segmentados em funções de modo a facilitar a manutenção e implementações futuras. Isso foi de extrema importância pois, a partir desse momento, o time parava de trabalhar com *frontend* para se dedicar ao *backend*.

Toda a parte de experiência de usuário, harmonização das cores, layout dos menus, da tela de *login*, da tela de recuperar senha, dos formulários, das listas, dos filtros, das buscas, das mensagens de erro, sucesso e informação além de ícones a serem utilizados com suas respectivas cores, foram escolhidos pelo *Product Owner* a medida que se fazia necessário cada um destes.

Durante o desenvolvimento desse primeiro *Sprint* foi muito comum cenários aonde funcionalidades já testadas fossem impactadas por alterações ou novas implementações. Isso se repetiu algumas vezes até que o time, juntamente com o *Product Owner*, optaram por ter uma suíte de testes automatizados que garantisse as funcionalidades e/ou cenários já testados pelos desenvolvedores (no processo de desenvolvimento) e pelo *Product Owner* (nos critérios de aceitação e nos relatos de não conformidades).

Foi acordado que, para qualquer funcionalidade nova ou alteração em funcionalidades antigas, o desenvolvedor deveria fazer os testes unitários correspondentes ao que este tinha desenvolvido e, logo em seguida, montar os casos de teste correspondentes aos critérios de aceitação estabelecidos no BDD e, se necessário, deveria ainda implementar mais algum teste específico caso o ele ou o time julgasse necessário.

Para a realização dos testes foi utilizado o RSpec, ferramenta esta que supria as necessidades que o time tinha de simular um usuário interagindo com o sistema, realizar testes unitários e escrever os testes de uma forma mais clara e simplificada. Entretanto, durante a integração do RSpec com o Rails o time teve problemas com a configuração e tratamento das

mensagens advindas da conclusão de cada caso de teste. Nesse momento viu-se a necessidade de acoplar outras ferramentas para complementar os testes do tipo Capybara, Poltergeist, Webkit, entre outros. Apesar de, aparentemente o time ter perdido tempo, na verdade foi um grande ganho pois, a partir desse feito, o time de desenvolvimento configurou o processo de integração contínua no Wercker para que, a cada implementação ou manutenção, fossem executados todos os testes já implementados no sistema, garantindo assim que o sistema não tivesse alterações em outras funcionalidades não correlatas com a atual.

Todo este *Sprint* demorou mais do que o estimado pois, durante todo o desenvolvimento, era desenvolvido concomitantemente toda a arquitetura necessária para a aplicação. Além disso alguns alinhamentos foram realizados para melhorar a comunicação do time. Alguns desencontros entre o proposto e o desenvolvido aconteceram, porém, pela proximidade do time com o *Product Owner* esses desencontros foram facilmente ajustados.

3.3.1.3 Resultados obtidos e considerações referentes ao *Sprint #1*

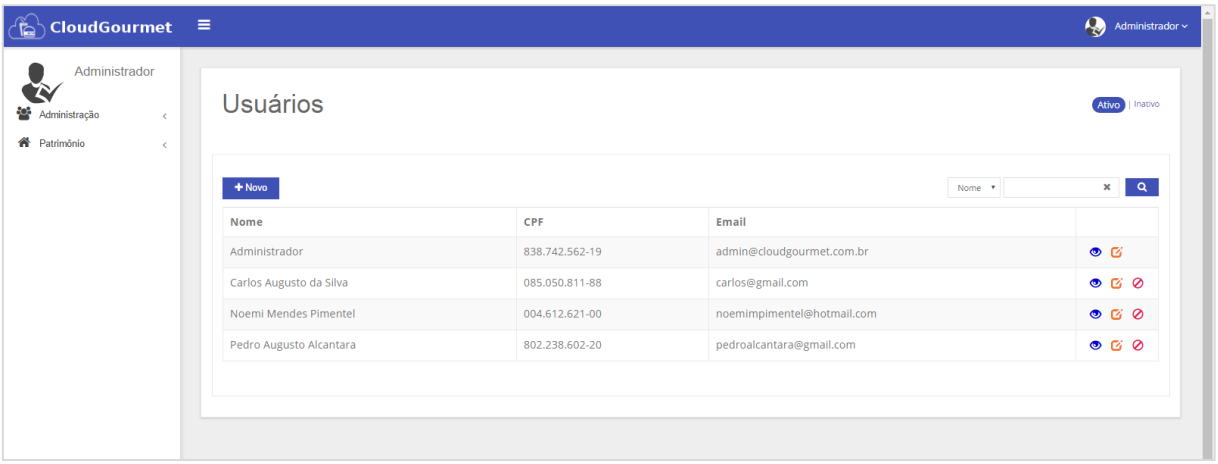
O primeiro *Sprint* teve duração de 6 semanas, a justificativa referente ao tempo gasto refere-se ao trabalho que seria realizado nesse *Sprint*, como por exemplo a preparação do ambiente de desenvolvimento, definição dos ícones, mensagens, entre outros, juntamente com o *Product Owner*. Outro ponto a ser considerado é que o time estimou que trabalharia 8 horas diárias 5 dias por semana, porém o máximo que o time de desenvolvimento teria para realizar o desenvolvimento era 4 horas diárias, 7 dias por semana.

Ao final do *Sprint* foi foram foi finalizado as seguintes histórias de usuários:

3.3.1.3.1 Como administrador do estabelecimento devo cadastrar os usuários que vão utilizar o sistema

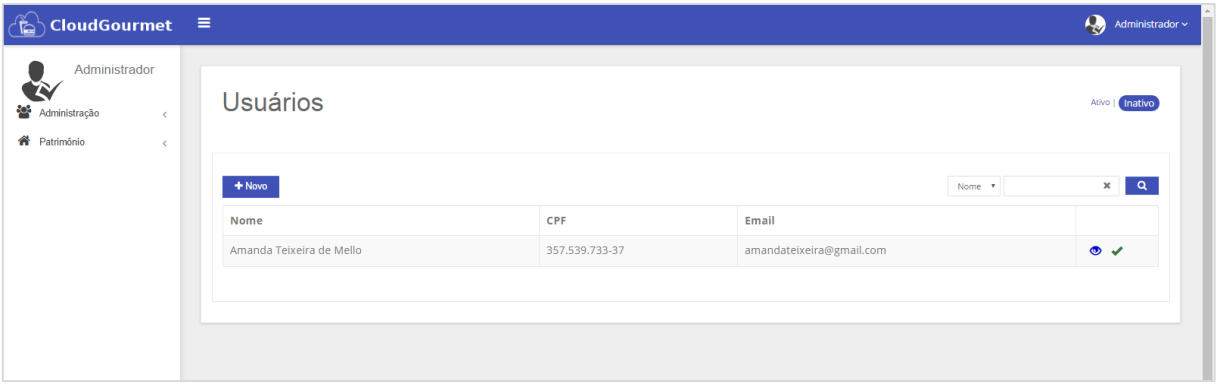
Na Figura 12 o usuário poderá visualizar a listagem de todos os usuários ativos do sistema com as informações de nome, cpf e email e, de acordo com a padronização adotada no sistema ele ainda poderá editar (Figura 16), visualizar informações (Figura 14), visualizar listagem de usuários inativos (Figura 13), criar novo usuário (Figura 15) e inativar, com a ressalva de que o usuário logado não poderá se inativar.

Figura 12 - CloudGourmet - Listagem de usuários ativos



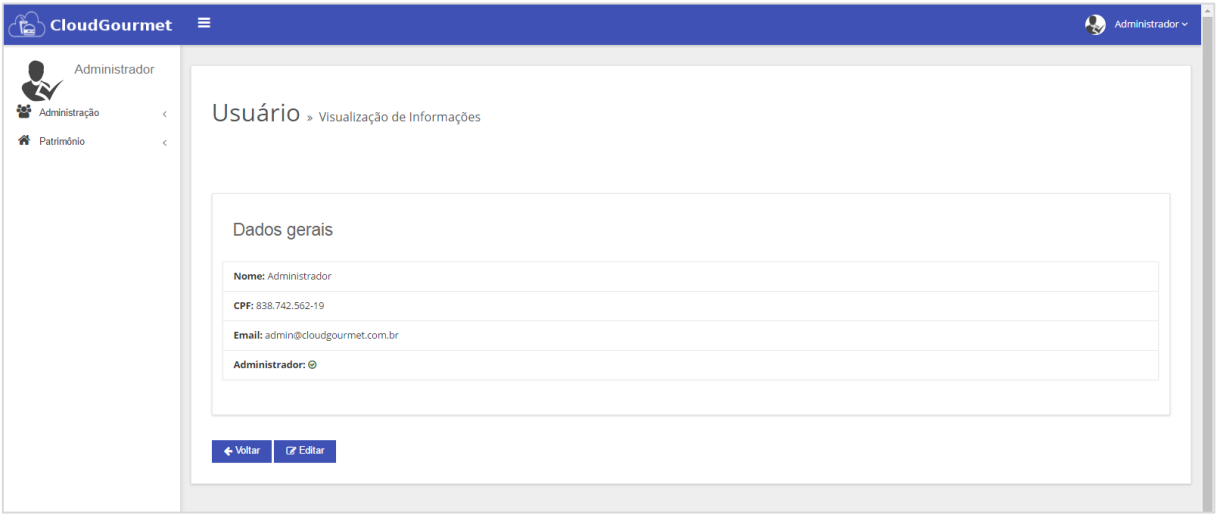
Fonte: <http://cloudgourmet.herokuapp.com/administration/users>

Figura 13 - CloudGourmet - Lista de usuário inativos



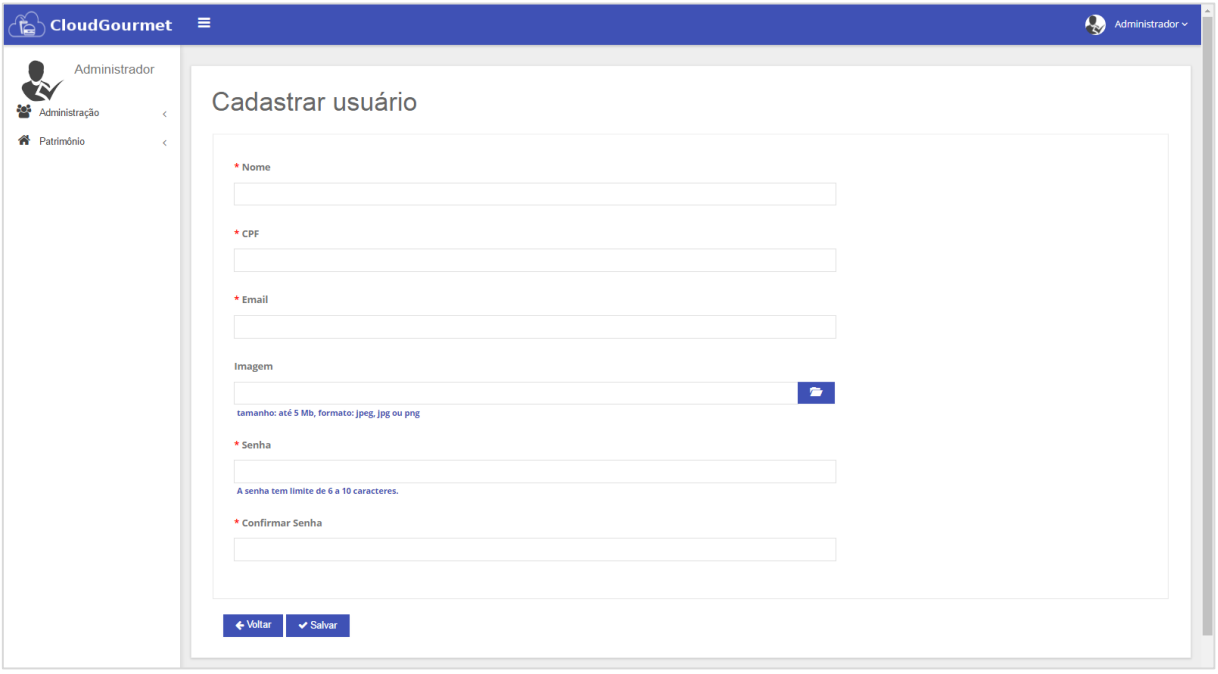
Fonte: <http://cloudgourmet.herokuapp.com/administration/users?active=false>

Figura 14 - CloudGourmet - Visualizar informações do usuário



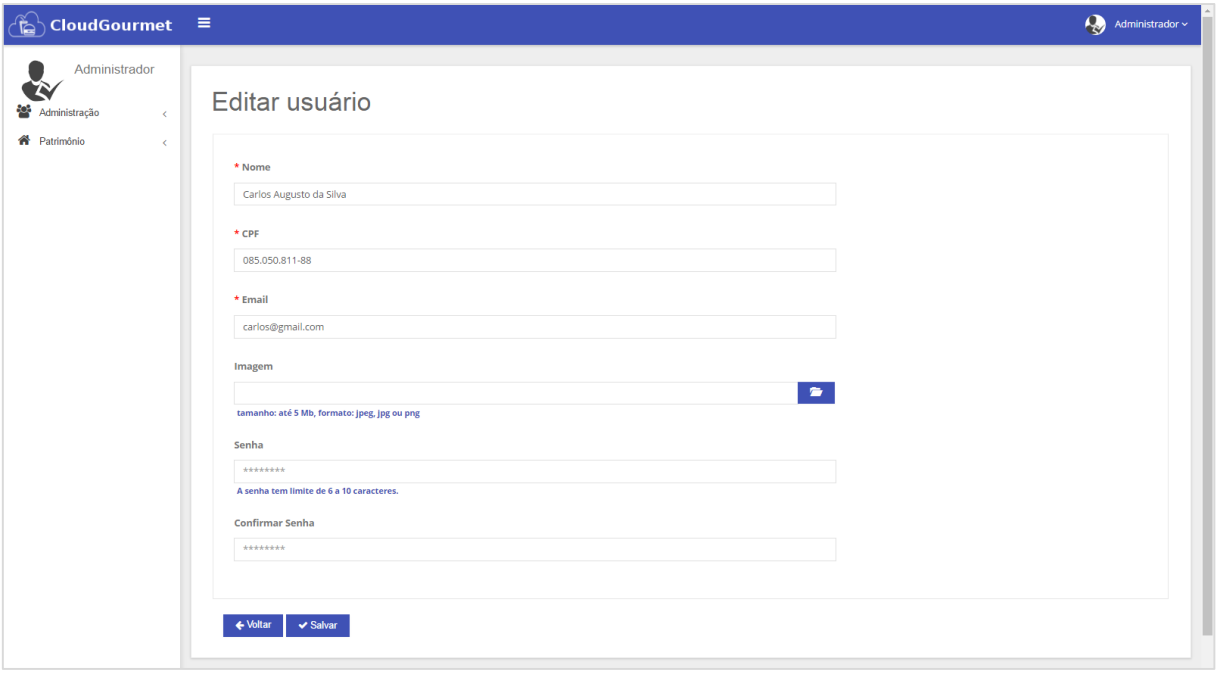
Fonte: <http://cloudgourmet.herokuapp.com/administration/users/1?>

Figura 15 – CloudGourmet - Cadastrar usuário



Fonte: <http://cloudgourmet.herokuapp.com/administration/users/new?>

Figura 16 - CloudGourmet - Editar cadastro de usuário



Fonte: <http://cloudgourmet.herokuapp.com/administration/users/4/edit?>

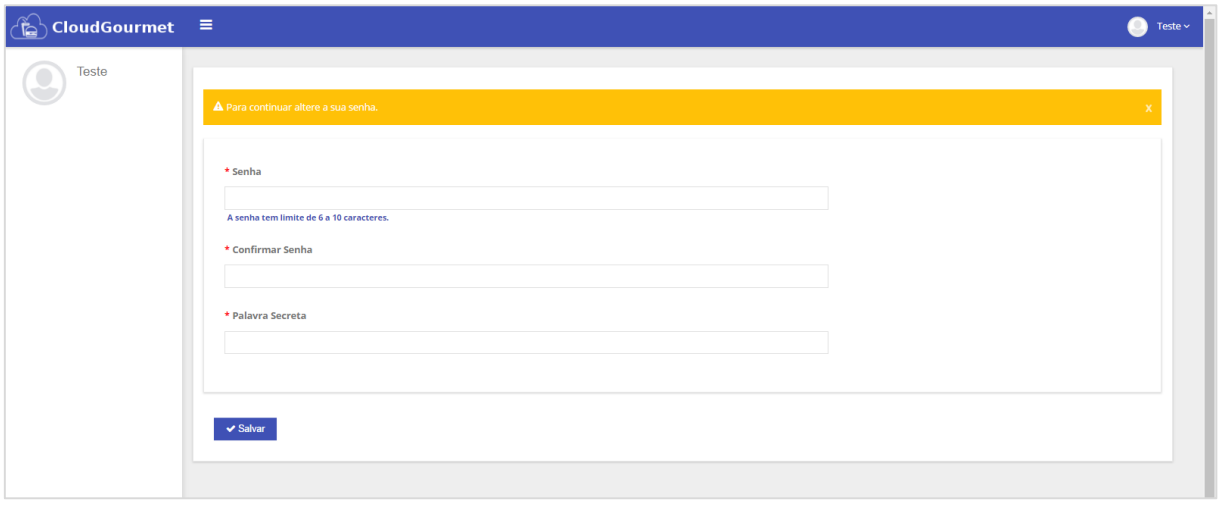
3.3.1.3.2 Como proprietário do estabelecimento quero que meus funcionários tenham que realizar login para acessar o sistema.

Nessa história foram desenvolvidas as seguintes funcionalidades: Acesso ao sistema (Figura 17), alteração da senha após o primeiro acesso (Figura 18), sair do sistema (Figura 19).

Figura 17 – CloudGourmet –Acesso ao sistema

Fonte: <http://cloudgourmet.herokuapp.com/users/sign_in>

Figura 18 – CloudGourmet – Alteração de senha, após primeiro acesso.



Fonte: <http://cloudgourmet.herokuapp.com/administration/user_profiles/6/unlock>

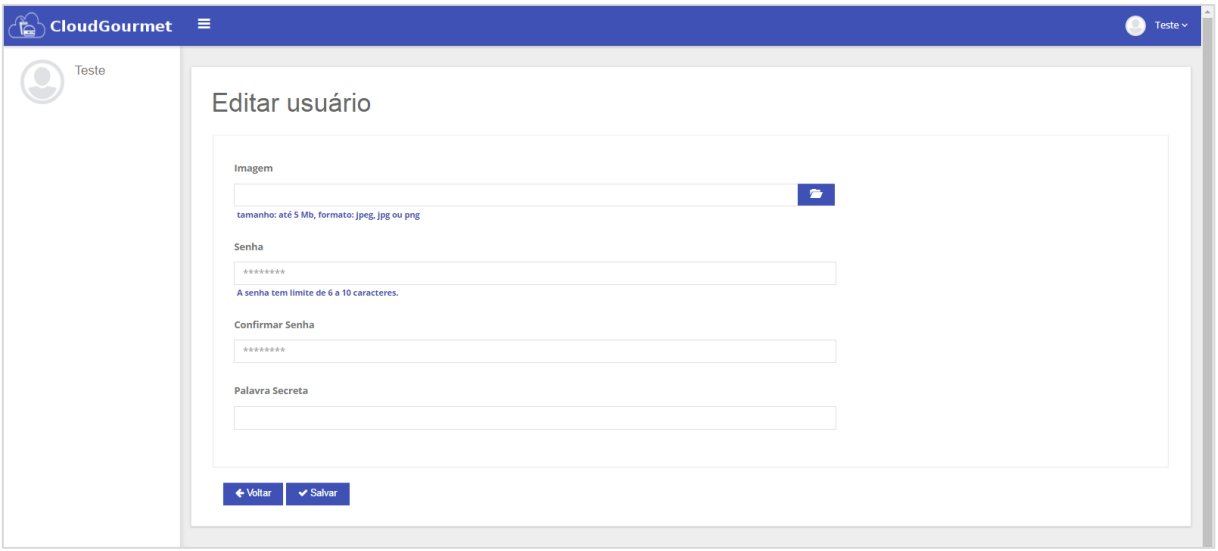
Figura 19 – CloudGourmet – Sair do sistema

Fonte: <http://cloudgourmet.herokuapp.com/>

3.3.1.3.3 Como funcionário do estabelecimento quero conseguir trocar minha senha quando eu quiser, pois gosto de trocá-las com frequência.

O usuário poderá alterar sua imagem, senha e palavra secreta a qualquer momento (Figura 20).

Figura 20 – CloudGourmet – Editar perfil de usuário

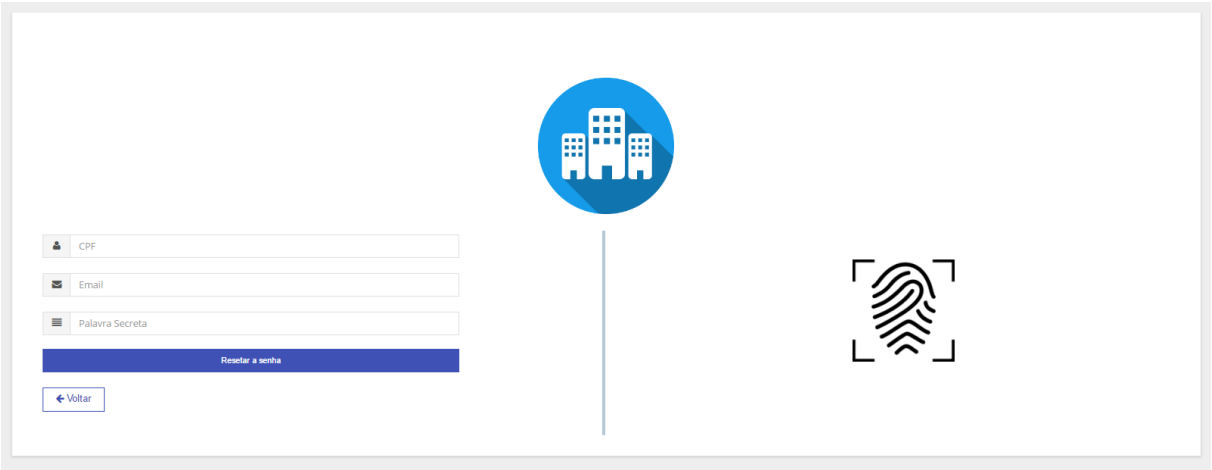


Fonte: <http://cloudgourmet.herokuapp.com/administration/user_profiles/6/edit>

3.3.1.3.4 Como gerente do estabelecimento quero que os funcionários consigam recuperar a senha para acessar o sistema sem necessidade de contatar-me.

O usuário poderá recuperar seu acesso ao sistema sem necessidade de solicitar ao administrador do sistema, basta inserir seu cpf, email e palavra secreta (Figura 21).

Figura 21 - CloudGourmet - Recuperar acesso ao sistema, quando o usuário esquece sua senha



Fonte:< http://cloudgourmet.herokuapp.com/users/password/new>

3.3.2 *Sprint* #2 - Possibilitaremos que o administrador possa definir o que cada funcionário pode acessar no sistema

3.3.2.1 Reunião de planejamento do *Sprint* #2

No refinamento do *Product Backlog* o time de desenvolvimento negociou defendendo que desenvolver o controle de permissão inicialmente seria mais viável, pois a cada nova funcionalidade desenvolvida seria testado as regras referentes ao controle de permissão.

O time de desenvolvimento levou em consideração o tempo gasto no desenvolvimento do *Sprint#1* e decidiu inserir na *Sprint*#2 apenas uma história de usuário (APÊNDICE F) onde estimou que seria gasto duas semanas para desenvolver essa funcionalidade de acordo com a definição de pronto.

3.3.2.2 Desenvolvimento do *Sprint#2*

Para o controle de permissão foi necessário utilizar um recurso do *template* até então não implementado que é a "modal" (tela menor projetada por cima da tela atual). Isso demandou algum tempo da equipe pois essa implementação seguiu o mesmo conceito das integrações anteriores do *template* ou seja, deveria ser reutilizável.

Algumas definições e alinhamentos foram necessários para o desenvolvimento do controle de permissões. Inicialmente foi simplificado o modelo utilizado adotando a filosofia de permissões por ações, controladores e módulos. Então para cada ação nova em um controlador, ou seja, uma nova rota, era necessário que essa permissão fosse cadastrada no banco de dados para que, posteriormente esta fosse atribuída a um usuário. Este processo de cadastramento no banco de dados era muito penoso para a equipe pois envolvia análise humana das ações que foram desenvolvidas e isso poderia acarretar em algum esquecimento das partes

de ações criadas. Para tanto o time de desenvolvimento mais uma vez optou pela automatização. Foi desenvolvido um script em Ruby que identificaria cada ação que existe no projeto e não foi cadastrada ainda no banco de dados como uma permissão e, automaticamente o script iria inserir essa permissão. Posteriormente esse script foi inserido como uma etapa no processo de *deploy* para que houvesse integridade na criação das permissões.

Foi alinhado com o *Product Owner* que, quando o usuário não tivesse a permissão para acessar o recurso, este não deveria nem ter a possibilidade de tentar acessá-la, ou seja, o ícone correspondente a ação não deveria ser exibido e se houvesse uma tentativa de acesso por URL o sistema deveria validar a permissão e retornar a página adequada, no caso a página de erro 403 (Baseado em erros HTTP) caso o usuário não tivesse permissão. Além desse alinhamento o *Product Owner* também definiu como os dados seriam exibidos na tela para a atribuição de cada permissão.

Um outro ponto importante a se destacar da atividade é que o botão direito do mouse e os botões F12 e Ctrl + I foram desabilitados por motivo de segurança dificultando o acesso ao modo de desenvolvedor do browser. Com acesso ao modo de desenvolvedor do browser é possível executar scripts e injetar tags HTML, além de poder forçar valores em formulário.

3.3.2.3 Resultados obtidos e considerações referentes ao *Sprint #2*

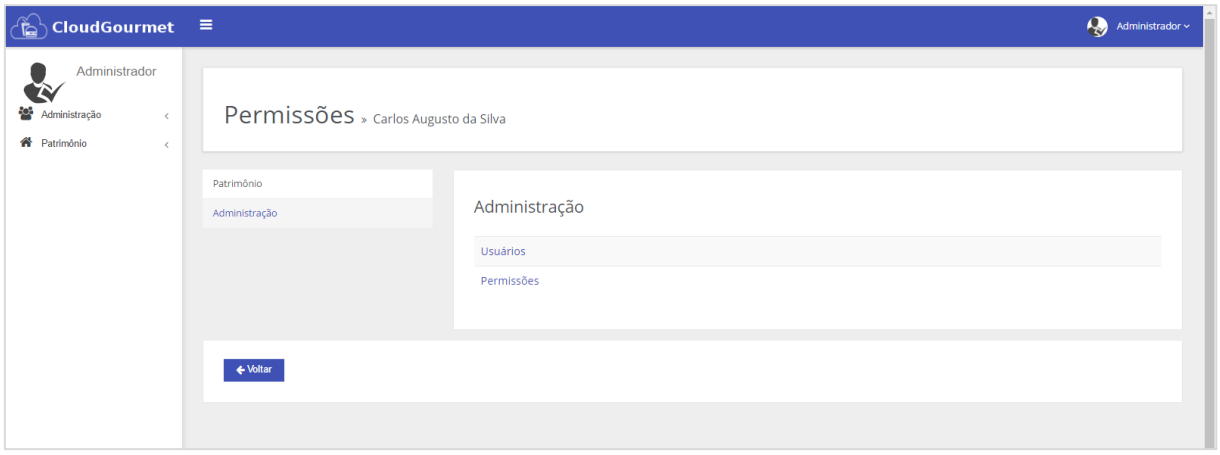
O *Sprint#2* teve duração de duas semanas, para validar se o controle de permissão, foi realizado testes nas funcionalidades desenvolvidas no produto entregue no incremento do *Sprint#1*.

As únicas funcionalidades disponíveis até o momento que não é controlado através do controle de permissão é:

- Acesso ao sistema (Figura 17);
- Alteração de senha, após o primeiro acesso (Figura 18);
- Sair do sistema (Figura 19);
- Editar perfil de usuário (Figura 20);
- Recuperar acesso ao sistema, quando o usuário esquece sua senha (Figura 21).

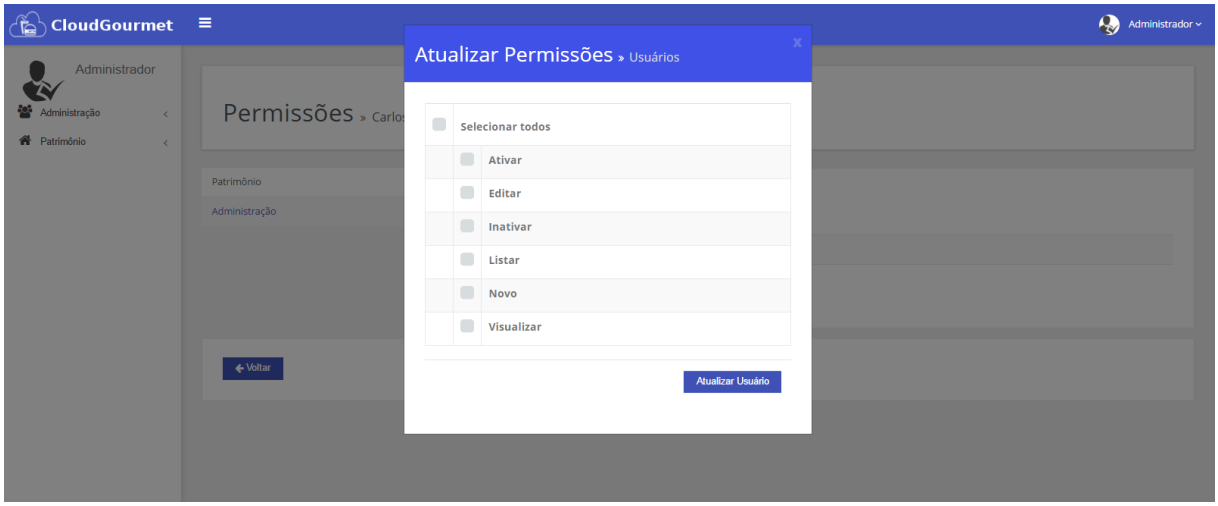
Para controlar o acesso das funcionalidades do sistema o administrador deve selecionar o módulo, o controle (Figura 22) e cada funcionalidade (Figura 23) que cada usuário pode utilizar no sistema.

Figura 22 – CloudGourmet - Módulos e Controles



Fonte: <http://cloudgourmet.herokuapp.com/administration/users/4/permits?module_name=administration>

Figura 23 – CloudGourmet – Funcionalidades referente ao controle: Usuários



Fonte: <http://cloudgourmet.herokuapp.com/administration/users/4/permits?module_name=administration>

3.3.3 *Sprint* #3 - Possibilitaremos que o controle de entrada e saída dos clientes sejam realizados através do CPF ou leitura biométrica

Para realizar o controle através da utilização de leitura biométrica era necessário adquirir o leitor biométrico. Após a aquisição foi necessário entrar em contato com o fabricante para que fosse disponibilizado a SDK (*Software Development Kit*) essa negociação foi realizada pelo time de desenvolvimento e teve início durante a terceira semana do primeiro *Sprint.*

Após a obtenção da SDK a equipe de desenvolvimento decidiu iniciar o *Sprint* referente ao controle de entrada e saída dos clientes nos estabelecimentos (APÊNDICE G), estimando que seria gasto 3 semanas para seu desenvolvimento, porém, mesmo com a SDK em

mãos o time de desenvolvimento passou por alguns impedimentos, após 5 dias a equipe decidiu interromper o *Sprint.*

Inicialmente a proposta era comprar o leitor biométrico FS80BR por conta da qualidade e do custo, além da SDK gratuita para desenvolvedores. O problema foi que a SDK só era liberada para quem fornecia o serial único do produto. Ao comprar o leitor o time de desenvolvimento entrou em contato com a Futronic que é a fabricante, depois de alguns emails a Futronic repassou o contato da SIS, representante da Futronic no Brasil, que encaminhou para a TechMeg, uma das distribuidoras da SIS no Brasil. O encaminhamento para a TechMeg se deu, pois, o serial do produto era na verdade de um lote que foi revendido pela TechMeg.

Depois de alguns dias a TechMeg repassou o driver e SDK para Windows 32 bits. Para validar a solução o time de desenvolvimento implementou um pequeno procedimento em Java que apenas conseguia ler a impressão digital, extrair o *template* e armazenar em um arquivo. Porém, por motivos de segurança, a implementação da autenticação do usuário deveria ocorrer no servidor da aplicação que opera com o S.O. base Linux e não no cliente base Windows. Com a SDK em Windows 32 bits seria possível apenas a implementação da autenticação nos terminais Windows.

Foi necessário então entrar em contato novamente com a TechMeg que ofereceu grande resistência no atendimento dificultando a aquisição da SDK Linux 64 bits. O time então entrou em contato com a SIS que, repassou novamente para a TechMeg. Em um último email o time foi informado que a SDK Linux 64 bits ainda estava em desenvolvimento e que dentro de alguns dias a Engenheira de software iria disponibilizar para download a nova versão.

Como o time de desenvolvimento estava em *Sprint* foi necessário interromper o *Sprint* e planejar uma nova para que não houvesse mais perca de tempo.

3.3.4 *Sprint* #4 - Possibilitaremos que funcionários realizem a entrada de produtos nos estoques do estabelecimento

Após a interrupção do *Sprint#3* a equipe iniciou o *Sprint* relacionado a entrada de produtos nos estoques (APÊNDICE H). O time de desenvolvimento estimou que gastaria quatro semanas no desenvolvimento desse *Sprint*.

3.3.4.1 Desenvolvimento do *Sprint #4*

Devido a arquitetura e todo o planejamento realizado nos *Sprints* anteriores, a primeira atividade desse *Sprint* foi realizado mais rápido em comparação com as outras histórias já implementadas. A única ponderação foi realmente a respeito de quais informações deveriam

ser exibidas na tela, informações estas que foram ditas posteriormente pelo *Product Owner* e como deveria ser a regra para o "Estoque Principal' que futuramente seria o único estoque que teria saída para venda. Não houveram demais complicações para este *Sprint*.

3.3.4.2 Resultados obtidos e considerações referentes ao *Sprint #4*

Do *Sprint*#4 apenas a primeira história de usuário foi desenvolvida, pois o time precisou pausar o desenvolvimento para concluir o trabalho de conclusão de curso, ela foi disponibilizada no ambiente de produção, porém as outras histórias de usuário ainda estão sendo desenvolvidas ou aguardando a realização de testes.

Na Figura 24 foi disponibilizado a listagem dos estoques ativos informando se o estoque é "Principal" ou não (informação necessária para determinar de qual estoque irá sair os itens para uma futura venda). Esta tela de listagem tem ainda os componentes padrão adotados para listagens do sistema. São eles: visualizar (Figura 26), editar (Figura 28), inativar, listar inativos (Figura 25), cadastro de novo registro (Figura 27), busca e filtro.

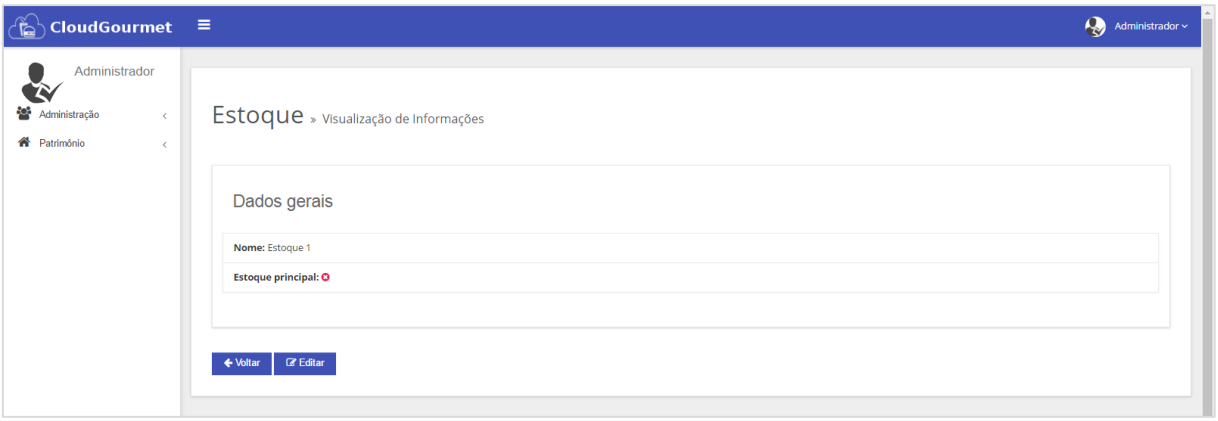
Figura 24 – CloudGourmet - Listagem de estoques ativos

Fonte: <http://cloudgourmet.herokuapp.com/patrimony/stocks>

Figura 25 – CloudGourmet - Listagem de estoques inativos

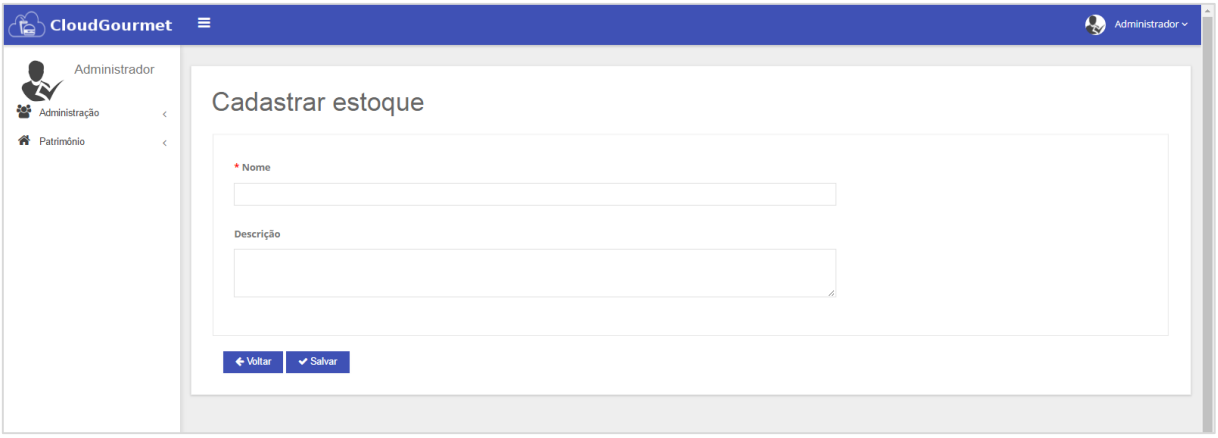
Fonte:< http://cloudgourmet.herokuapp.com/patrimony/stocks?active=false>

Figura 26 – CloudGourmet – Visualizar estoque



Fonte: <http://cloudgourmet.herokuapp.com/patrimony/stocks/2?>

Figura 27 – CloudGourmet – Cadastrar estoque



Fonte: <http://cloudgourmet.herokuapp.com/patrimony/stocks/new?>

Figura 28 - CloudGourmet - Editar estoque

Fonte: <http://cloudgourmet.herokuapp.com/patrimony/stocks/2/edit>

4 FERRAMENTAS SELECIONADAS PARA O PROJETO

Para realizar o desenvolvimento do sistema CloudGourmet o time de desenvolvimento teve necessidade de organizar o ambiente durante a *Sprint#1*, nessa sessão serão apresentadas as ferramentas selecionadas. Para compreender como está sendo utilizada as ferramentas leia a definição de pronto apresentado na sessão 2.1.7.

Para auxiliar a compreensão das ferramentas utilizadas no projeto, os autores elaboraram uma representação visual apresentado na Figura 29 que representa as ferramentas utilizadas no projeto até o momento. As ferramentas selecionadas estão divididas em:

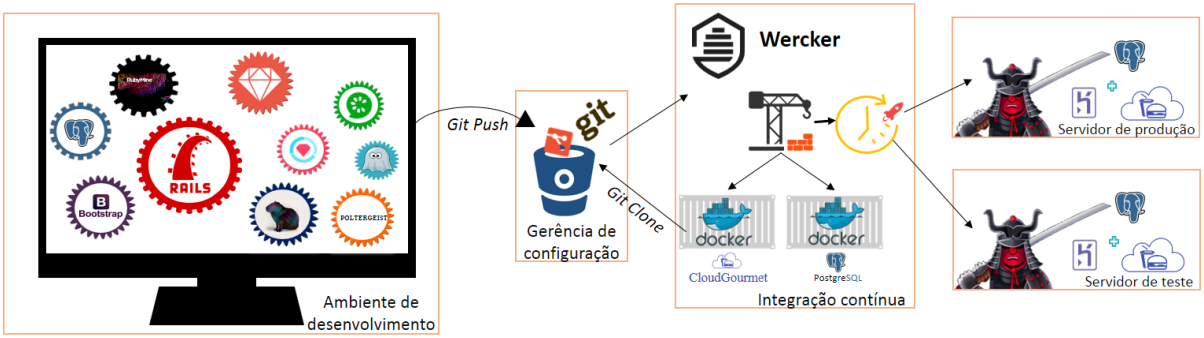
- Ambiente de desenvolvimento: as ferramentas utilizadas pelo time no ambiente de desenvolvimento local, são elas: RubyMine, Rails, Ruby, PostgreSQL, Bootstrap, RSpec, Capybara, Cucumber, PhantomJs e Poltergeist;

- Gerência de configuração: as ferramentas são: Bitbucket (um serviço de gerencia de repositórios git); Git (Repositório e gerenciador de versionamento do código);

- Serviço de integração contínua: Ambiente que executa os testes automatizados, realizando testes de regressão;

- Servidor de produção e teste: disponibilização da aplicação online (Heroku).

Figura 29 – Ferramentas utilizadas até o momento no desenvolvimento do sistema CloudGourmet



Fonte: Elaborado pelos autores.

Nas seções a seguir cada ferramenta será detalhada, justificando seu uso.

4.1 Linguagem de programação Ruby[6]

Ao ser concebida a ideia do produto, o time se reuniu para discutir a arquitetura e qual seria a linguagem de programação que seria utilizada para a concepção do projeto. Neste momento, foi escolhida uma linguagem de programação de expertise do time que viabilizaria o

[6] Para mais informações referentes a linguagem de programação Ruby acesse o site da comunidade: https://www.ruby-lang.org/

desenvolvimento com foco na qualidade e produtividade. Tendo esses pontos como balizadores dessa escolha, optou-se então pela linguagem Ruby.

A linguagem de programação Ruby foi criada pelo japonês Yukihiro Matsumoto (Matz), nasceu em 1993, mas foi apresentada em 1995, com o intuito de ser uma linguagem simples de ler, fácil de compreender e agradável para programar para colaborar com o desenvolvimento e manutenção (CAELUM, 2016).

É uma linguagem *open source*[7], interpretada foi inspirada nas linguagens de programação *Perl*, *Smalltalk* e *Lisp*, sendo uma linguagem de programação multiparadigma contendo características dos paradigmas de orientação a objetos e funcional. Sua disseminação foi auxiliada pelo *framework* Ruby *on* Rails. (SOUZA, [2013 - 2014]).

As variáveis são dinamicamente tipadas, ou seja, o tipo é verificado a cada interação; implicitamente tipada, pois os tipos são deduzidos pelo interpretador; e fortemente tipada, pois em Ruby o tipo da variável é relevante (CAELUM, 2016).

Em Ruby, tudo é objeto, até mesmo os tipos primitivos (números, *strings*, entre outros), cada objeto tem suas variáveis de instância e métodos, o que pode facilitar sua utilização pois as regras que se aplicam aos objetos aplicam a tudo em Ruby (RUBY, 2016). A versão utilizada nesse projeto é Ruby 2.3.1 que foi lançada no dia 24 de abril de 2016.

4.2 *Framework* Rails

Ruby por si só, trouxe ao time uma quantidade considerável de ferramentas para tratamento de dados, porém era necessário que o time construísse uma aplicação com o seguinte requisito não funcional: portabilidade. Levando em consideração que estabelecer uma configuração mínima de terminal de trabalho era limitar a adesão desta solução no mercado, o time optou por ter uma solução que seria acessada de um browser em um servidor local.

Para que isso acontecesse e houvesse total compatibilidade com a linguagem de programação Ruby optou-se por usar o *framework* Rails.

Rails é um *framework open-source* para desenvolvimento de aplicações web, foi criado por David Heinemeier Hansson; em dezembro de 2005 foi disponibilizada a versão 1.0 e em 2006 começou a ganhar muita visibilidade na comunidade de desenvolvimento web (CAELUM 2016).

> O termo "*framework* web" geralmente se refere, na vida real, a um software que lhe permite escrever programas melhores, proporcionando formas específicas ou recomendadas de organização de seu código. Também fornece muito código

[7] "*Open source* é um termo em inglês que significa código aberto. Isso diz respeito ao código-fonte de um software, que pode ser adaptado para diferentes fins" (CANALTECH, 2016)

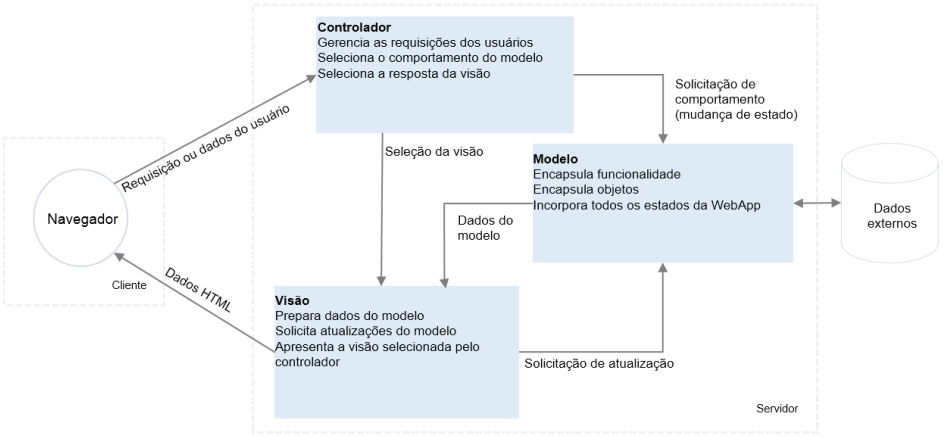
> reutilizável, evitando que você perca tempo de desenvolvimento recriando a roda" (PHAM, A.; PHAM, P., 2011, p. 191, grifo do autor).

As características do *framework* são:

- *Don't Repeat Yourself* (Não se repita) – Incentiva a reutilização de código (FUENTES, 2012);
- *Convention over Configuration* (Convenção à configuração) – Se utilizar padrões de nome, localização de arquivos, nome de classes e métodos entre outras convenções pode-se facilitar a implementação (CAELUM, 2016).

Utiliza o padrão de arquitetura de software MVC (*Model-View-Controller*), padrão considerado adequado para aplicações web (FUENTES, 2012). Uma representação visual referente ao padrão MVC é apresentado na Figura 12.

Figura 30- Arquitetura MVC



Fonte: Adaptado de Jacyntho, Schwabe e Rossi (*apud* PRESSMAN, 2011, p. 349).

4.3 RSpec

Durante a criação/alteração de algumas funcionalidades pelo time, era constatado pelo *Product Owner* que outras funcionalidades já 'prontas' perdiam a característica original apresentando erros decorrentes dessas mudanças, algo considerado natural uma vez que o time não tinha condições de realizar testes de regressão manualmente a cada funcionalidade criada.

O processo de testes de regressão demanda mais tempo à medida que o software vai evoluindo, sofrendo incrementos e etc. Para tanto, era necessário que os cenários de testes de aceitação fossem descritos de forma a automatizar os testes que anteriormente eram realizados apenas pelo *Product Owner* na entrega de uma história de usuário.

Depois de uma reunião com equipe, ficou acordado que a ferramenta a ser escolhida deveria trabalhar bem com a linguagem e o *framework* anteriormente escolhidos, além de

conseguir manter os critérios de aceitação documentados de modo que se criasse uma documentação ‘viva’ e, por fim, desse suporte à testes de unidade, integração e aceitação. Neste momento então foi escolhido o *framework* RSpec.

O RSpec foi criado em 2005 por Seven Baker, com contribuições de Dave Astels e Aslak Hellesoy com o objetivo de ser uma suíte de testes totalmente integrada com o Ruby on Rails. (RSPEC, [2015]).

De acordo com Nando Vieira (2016, p.1, grifo nosso), um grande contribuinte da comunidade Rails e autor de alguns artigos e livros sobre RSpec, segue a definição: “O RSpec é um *framework* BDD escrito em Ruby que permite que você escreva testes em uma linguagem mais natural, em inglês”.

Uma das vantagens em utilizar o RSpec é que a escrita em linguagem natural pode ser considerada uma documentação “viva”, ou seja, a documentação – inserção de novos testes automatizados - será atualizada a cada nova funcionalidade desenvolvida no software.

O RSpec subdivide-se em módulos, são estes: *model*, *controller*, *request*, *feature*, *view*, *helper*, *mailer* e *routing*. Cada qual responsável por uma parte específica da aplicação, dentro da própria estrutura do Rails.

Levando em consideração que testar leva tempo; tempo custa horas extras da equipe de desenvolvimento e consequentemente maiores custos. A automatização de testes começa a ficar inviável quando o custo de testar é maior que o custo e tempo do próprio projeto. Sendo assim a análise do que deve ser testado deve levar em consideração pontos críticos das funcionalidades (CAETANO, 2012).

Por fim, com a adesão dessa ferramenta foi possível que o time de desenvolvimento identificasse os possíveis impactos negativos gerados por cada alteração, diminuindo a possibilidade de que outras funcionalidades do sistema fossem impactadas.

4.4 Capybara

Para que fosse implementado os testes de aceitação (feature) do RSpec era necessário que o time de desenvolvimento aderisse a uma nova ferramenta que simulasse exatamente a ação de um usuário comum com a aplicação, de modo que os cenários do BDD descritos nas histórias de usuário fossem implementados com o mínimo de tempo possível. Para isso, foi escolhido a biblioteca (gem) Capybara.

O Capybara é totalmente integrado com o RSpec, sendo uma biblioteca (*gem*) de automação baseado em web que executa testes funcionais simulando como os usuários

interagem com a aplicação. Os testes são escritos em uma DSL[8] (*Domain Specific Language*) para serem executadas pelo *webdriver* configurado. Escrever as os testes funcionais utilizando DSL facilita a compreensão dos códigos de teste.

Hoje o Capybara suporta três *drivers* e de acordo com (CAPYBARA TEAM [2015]), são eles:

- *rack::test*: *driver* padrão sendo o mais rápido, porém não acessa os recursos HTTP fora do módulo Rails *App* onde os testes foram realizados.
- *selenium-webdriver*: é o *driver* mais utilizado. Simula e abre um *navegador*. Consegue acessar recursos HTTP (*HyperText Transfer Protocol*) fora da aplicação Rails e podendo também ser utilizado para a realização de testes sem necessáriamente abrir o navegador *(headless mode).*
- Capybara-*webkit*: Mais rápido que o selenium, o capybara webkit é puramente *headless* (não permitindo o modo de abertura de navegador).

4.5 Phantom JS

Mesmo tendo um suporte de linguagem DSL, oferecido pelo Capybara, foi preciso necessário selecionar uma ferramenta que realizasse comandos no browser via JQuery, executando externamente cada chamada realizada no script de teste feito na DSL do Capybara. Depois de alguma análise foi selecionado a utilização do PhantomJS.

Com sua primeira *release* em 23 de Janeiro de 2011, feito por Ariya Hidayat, o Phantom Js é um navegador Javascript baseado em webkit utilizado para automação de testes.

É utilizado em testes que carregam toda a aplicação e simula um usuário que execute uma sequência de ações para que os cenários de testes sejam executados e detectados os possíveis erros.

O Phantom JS consegue executar essas ações através de um *script* pré-definido sem necessidade de um navegador aberto. Manipulando o DOM [9](*Document Object Model* – Modelo de objeto de documento) com *JQquery*[10] conseguindo executar todos os comandos que um usuário conseguiria realizar na aplicação. (REINHEIMER, 2012).

[8] DSL é "[...] a prática de se criar pequenas linguagens para resolver um problema bem específico" (SILVEIRA, 2007).
[9] "O DOM é uma multi-plataforma que representa como as marcações em HTML, XHTML e XML são organizadas e lidas pelo navegador" (FRANKLIN, 2011).
[10] JQuery é uma biblioteca de javascript (W3SCHOOLS, 2016).

4.6 Poltergeist

Aliando a vantagem e rapidez do PhantomJS, um navegador webkit que executa ações através de *JQuery*, com a ferramenta de testes automatizados Capybara, o Poltergeist é um driver que faz essas duas ferramentas trabalharem em conjunto para a execução de testes de aceitação. (PhantomJS, 2016).

4.7 Cucumber

Criado para manter uma documentação 'viva' e executável do código. As regras que compunham o sistema definidas no planejamento dos *Sprints*, e traduzidas em histórias de usuário, eram implementadas em casos de teste para que, ao executar os testes, todas as regras que compunham o sistema fossem exibidas na tela demonstrando que a funcionalidade está cumprindo com a proposta definida pela história de usuário. Segue abaixo em breve descrição a definição dessa ferramenta:

> O Cucumber é uma ferramenta que executa testes de aceitação em texto puro, ou seja, em linguagem natural. O Cucumber permite que se escreva histórias em inglês ou português, por exemplo. Isso permite que clientes que não possuem conhecimentos técnicos possam descrever funcionalidades passíveis de implementação pelo time de desenvolvimento (VIEIRA, 2016, p.1).

4.8 PostgreSQL

Para armazenamento e gerenciamento dos dados foi utilizado o SGBD (Sistema Gerenciador de Banco de Dados) PostgreSQL. A decisão do banco de dados levou em consideração muitos fatores, dentre eles: tipos de dados disponíveis, conector com o *framework* adotado (Rails), opção para clusterização e, principalmente se o serviço de hospedagem escolhido (Heroku) oferecia suporte.

Atualmente o Heroku oferece suporte gratuito para o PostgreSQL. Então para este trabalho ele foi adotado como uma boa opção de banco de dados.

O PostgreSQL é um sistema gerenciador de banco de dados relacional *open source* (código aberto) que tem mais de 15 anos de existência. Consegue operar nos principais sistemas operacionais incluindo: *Linux, Unix (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64*) e *Windows* além de ter implementado os principais tipos de dados do SQL (*Structured Query Language*):2008 como *INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL* e *TIMESTAMP* além de binários incluindo imagem, música e vídeo. (*THE POSTGRESQL GLOBAL DEVELOPMENT GROUP*, 2016).

Dessa forma este banco de dados supre completamente a demanda inicial deste projeto.

4.9 *Template* Bootstrap

No time de desenvolvimento existia uma deficiência referente a criação de interfaces, principalmente para tratamento de layout responsivo. Tentando superar esse impedimento chegou-se ao consenso de comprar um *template*, algo que já houvesse uma padronização, um trabalho de web design bem construído e que não dispendesse de muito recurso financeiro por parte da equipe.

O *framework* Bootstrap foi criado pelo Twitter em meados de 2010 por Mark Otto e Jacob, tendo sua primeira versão em 19 de agosto de 2011. Projetado para ser um *framework frontend*, hoje é o projeto *frontend*, *open source* mais popular do mundo (BOOTSTRAP GROUP, 2016).

4.10 Bitbucket

Para rastrear alterações de modo a saber quem as fizeram, trabalhar com repositório Git privado sem custo algum, controlar as versões do projeto com possibilidade de integração com outras ferramentas, tornaram o Bitbucket um bom candidato para ser o gerenciador de configuração do projeto CloudGourmet.

O Bitbucket é um serviço de hospedagem web oferecem planos gratuitos e pagos para seus clientes. Lançado em 2008 e comprado pela Atlassian em 29 de setembro de 2010 Bitbucket era uma *startup* fundada por Jesper Nøhr que oferecia serviços de hospedagem apenas para repositórios *Mercurial* e, em 3 outubro de 2011, Bitbucket anuncia o suporte para repositórios Git. (DAVIS, 2016).

Com essa ferramenta foi possível integração com o gerenciador de integração contínua Wercker, que também foi utilizado nesse projeto, além de controle de acesso de cada integrante do time de desenvolvimento no que tange alterações e configurações do repositório, além de ferramentas visuais para comparação das alterações/inserções de novas linhas de código.

4.10.1 Git

Criado por em 2005 por Linus Torvalds (também criador do *Linux*), depois de um desentendimento com os criadores do *BitKeeper* (ferramenta terceirizada que versionava o *kernel* do *linux*). O Git é um sistema de versionamento de código baseado no *BitKeeper* que tem por características principais: a velocidade, o design simplificado, o suporte a desenvolvimento não linear, ser totalmente distribuído e consegue versionar também grandes projetos eficientemente. (*Software Freedom Conservancy*, 2011)

4.11 Wercker

A Construção do projeto (*build*), a cada atividade é realizada com o objetivo de identificar possíveis impactos. O Wercker é uma plataforma de integração contínua lançada em 2011, que através de um arquivo simples de configuração YAML (*Yet Another Markup Language*) (SHAY, 2016) oferece recursos para construção da aplicação, execução de testes automatizados e *deploy* (atualização do código em um ou vários servidores).

Com essa ferramenta integrada com o Bitbucket, a cada *commit* realizado, uma nova *build* era iniciada, criando o ambiente adequado para a aplicação e executando todos os testes implementados. Ao final de cada build efetuada com sucesso era possível realizar o *deploy* no servidor de produção ou de testes de forma automatizada e de acordo com script implementado no wercker.yml.

Para este projeto foi adotado que todo o código na *Branch Master* do Bitbucket deveria entrar em produção automaticamente, sendo assim sempre que um código era mesclado com a *Branch Master* o Wercker iniciava o processo de construção (*build*) e em seguida começava o *deploy* para o servidor de produção. Esse processo só foi possível através de um fluxo de trabalho (*workflow*) que foi previamente configurado para identificar de qual *branch* se tratava a *build*.

Como parte do processo de desenvolvimento do produto CloudGourmet dentro do Wercker o *Product Owner* poderia apenas realizar o *deploy* para o servidor de testes para que este realizasse os testes de caixa preta e retornasse com um feedback relatando se houve conformidade com o que foi proposto ou se havia alguma não conformidade.

Foi a partir da adesão do Wercker pelo time de desenvolvimento que foi possível controlar as atividades que efetivamente foram para o servidor de teste e/ou para o servidor de produção, mantendo o horário, data, relação com o *commit* no Bitbucket e usuário que realizou a ação. Com esses dados, o time aumentou a rastreabilidade do que realmente se tinha em produção.

4.11.1 Docker

No processo adotado para a concepção do CloudGourmet o Docker separa em *containers* a aplicação do banco de dados PostgresSQL, e aplicação desenvolvida, no momento de cada construção do processo no Wercker.

Docker é um projeto *open source* criado por Solomon Hykes na França como um projeto interno da empresa dotCloud. Docker é a evolução da *dotCloud* e teve sua primeira *release open source* em março de 2013. O Docker é uma plataforma aberta para

desenvolvedores e administradores de sistemas usada para: construir, executar e distribuir "máquinas". Diversas "máquinas" podem ser iniciadas no mesmo *host* pois elas são separadas logicamente, o que muda de uma estrutura de máquinas virtuais é a arquitetura que deixa de ter a figura do hospedeiro com *kernel* e sistema operacional, para ter vários *containters* operando sobre um mesmo *kernel*, o *kernel* do *host*. (DOCKER INC., 2016) (BRITO, 2015).

4.12 Heroku

Algumas soluções de servidores em *cloud* foram analisadas para a primeira versão de demonstração da aplicação CloudGourmet. Foram analisados os seguintes requisitos: suporte à linguagem e o *framework* escolhido, disponibilidade, suporte pela comunidade, valores, curva de aprendizado, entre outros. Até que o time de desenvolvimento escolheu o Heroku, uma solução que tem por arquiteto chefe o criador da linguagem de programação Ruby, Yukihiro Matsumoto.

Heroku é um serviço de plataforma para aplicações *web PAAS (Platform as a Service)* utilizando a plataforma Amazon EC2 que é um IaaS ou *Infrastructure as a Service*, focada em tirar do desenvolvedor as preocupações com infraestrutura, levando-o a ter mais tempo para codificar.

Foi fundado em 2007, por Orion Henry, James Lindenbaum, e Adam Wiggins. Foi comprado pela Salesforce em 2011 e hoje, o Heroku faz parte da Salesforce App Cloud (SALESFORCE COMPANY, 2016).

Como forma de incentivo a aplicações *open source* e até mesmo para captação de novos clientes o Heroku oferece máquinas virtuais "pequenas" que são chamadas de "Dynos". Hoje basicamente um Dyno é uma máquina com 4 cores com até 512 Mb de RAM sem swap e sem suporte à persistência de arquivos, ou seja, não é possível armazenar imagens na pasta pública, por exemplo.

Essa unidade mínima pré configurada para uma aplicação Ruby on Rails foi utilizada neste projeto na versão de demonstração. O desempenho do Dyno gratuito oferecido pelo Heroku foi altamente satisfatório, tendo em vista que seriam poucos acessos. Para uma aplicação com mais acessos seria necessário apenas contratar uma máquina com maior capacidade, revelando assim uma escalabilidade vertical. Logo, com um crescimento massivo de acessos na aplicação seria necessário que o time de desenvolvimento analisasse uma forma de escalar a aplicação horizontalmente, ou seja, a aplicação deveria ter o mesmo comportamento com várias maquinas em *cloud*, atuando simultaneamente para compartilhar a carga de acesso, reduzindo assim o custo com servidores.

5 CONSIDERAÇÕES FINAIS

Durante o processo de desenvolvimento do projeto a equipe teve a oportunidade de aprender novas tecnologias, criar e utilizar uma metodologia ágil, além de explorar a cada atividade a produtividade, a qualidade e a entrega.

No início a equipe teve dificuldades para alinhar o que era para ser realizado, mas com o aumento da comunicação, definição do processo e a maturidade da equipe iam aumentando, o alinhamento e produtividade da equipe foi aumentada.

Cada atividade desempenhada teve como meta a agregação de valor ao produto, de modo que se uma atividade não agregasse tanto valor, ela iria para o *Backlog* e posteriormente seria adicionada em um *Sprint* para ser realizada.

A experiência obtida durante a escolha das ferramentas trouxe a equipe a maturidade necessária para que o projeto continue se consolidando como um produto de fato, tendo plena capacidade para, em uma etapa mais madura, ser lançado no mercado.

Entretanto, baseado no tempo gasto até o momento para o desenvolvimento das funcionalidades, será necessário realizar uma análise nos marcos do *Roadmap* do produto, pois a equipe não conseguirá entregar nos marcos definidos as funcionalidades esperadas. A equipe espera que após o primeiro marco do *Roadmap*, eles consigam compor uma carteira de clientes.

Compreende-se a necessidade de incrementar o processo de desenvolvimento que está sendo utilizado, para conseguir tirar melhor proveito dos indivíduos e está ciente também que qualquer alteração no cenário, seja a captação de clientes ou inserção de novos integrantes ao time de desenvolvimento será necessário acrescentar, métodos e ferramentas ao processo.

Futuramente o planejamento para a infraestrutura tanto de testes quanto de produção deverá ser alterado também de acordo com a quantidade de acessos para garantir maior qualidade no que foi proposto.

O ambiente ágil definido trouxe uma nova experiência referente desenvolvimento de software, pois a comunicação entre os envolvidos foi priorizada, documentação foi gerada apenas para nortear o desenvolvimento, pois todas as dúvidas, analises e detalhamento foram realizados, porém não foram arquivados. Até o momento o processo definido conduziu o time a resultados satisfatórios, onde priorizou-se a entrega de um sistema executável auto documentado.

REFERÊNCIAS

ASTELS, D.; MILLER G.; NOVAK, M. **Extreme programming:** Guia Prático. Rio de Janeiro: Ed. Campus, 2002.

BECK, K *et al*. **O manifesto Agil**. Tradução: João Rotta Neto, 2002. Disponível em: <http://paginapessoal.utfpr.edu.brfrufrek/pos-web/p/arquivos/O_manifesto_agil> Acesso em: 01 de maio de 2015.

BERNARDO, Kleber. **Estória de usuário:** você saberia conta?. 2014. Disponível em: <http://www.culturaagil.com.br/estoria-de-usuario-voce-saberia-contar/>. Acesso em: 10 out. 2016.

_____. ***Product Backlog*:** o que é?. 2014. Disponível em: <http://www.culturaagil.com.br/product-backlog-o-que-e/>. Acesso em: 10 out. 2016.

BOOTSTRAP GROUP. **Bootstrap:** About. 2016. Disponível em: <http://getbootstrap.com/about/>. Acesso em: 09 nov. 2016.

BRITO, Rafael. **Docker – Infraestrutura simples e rápida.** 2015. Disponível em: <https://www.vivaolinux.com.br/artigo/Docker-Infraestrutura-simples-e-rapida>. Acesso em: 09 nov. 2016.

CAELUM. **Desenvolvimento ágil para web com Ruby on rails**. Disponível em: <https://www.caelum.com.br/download/caelum-ruby-on-rails-rr71.pdf > Acesso em: 15 out. 2016.

CAETANO, Cristiano. **Melhores práticas e desafios na automação de testes.** 2012. Disponível em: <http://www.qualister.com.br/blog/melhores-praticas-e-desafios-na-automacao-de-testes>. Acesso em: 08 nov. 2016.

CANALTECH. **O que é open source**. Disponível em: < https://canaltech.com.br/o-que-e/o-que-e/O-que-e-open-source/> Acesso em: 14 out 2016.

CAPYBARA TEAM. **Capybara.** [2015]. Disponível em: <https://github.com/teamcapybara/capybara>. Acesso em: 08 nov. 2016

DAVIS, Justine. **Bitbucket Cloud: 5 million developers and 900,000 teams.** Bitbucket, 2016. Disponível em: <https://blog.bitbucket.org/2016/09/07/bitbucket-cloud-5-million-developers-900000-teams/>. Acesso em: 08 nov. 2016.

Docker Inc. **What is Docker.** 2016. Disponível em: <https://www.docker.com/what-docker>. Acessado em: 09 nov. 2016.

ENDEAVOR BRASIL. **Roadmap:** a bússola para desenvolver seu produto ou projeto. 2015. Disponível em: <https://endeavor.org.br/roadmap/ >. Acesso em: 20 out. 2016.

FOWLER, Martin; SCOTT, Kendall. **UML essencial:** Um breve guia para a linguagem-padrão de objetos, 3. ed. Porto Alegre: Bookman Editora, 2005.

FRANKLIN, Alysson. **Tenha o DOM**.2011. Disponível em <http://tableless.com.br/tenha-o-dom/ >. Acesso em: 08 nov. 2016.

FUENTES, V. B. **Ruby on Rails:** Coloque sua aplicação web nos trilhos. São Paulo: Casa do código. 2012.

GUIMARÃES D.; CABRAL P. **Significado de comunicação**. Disponível em: <https://www.significados.com.br/comunicacao/>. Acesso em: 10 out. 2016.

HELM, Rafael; WILDT, Daniel. **User stories:** Por que e como escrever requisitos de forma ágil. Disponível em: <http://medias.ciranda.me/entities/23/8eeece0dadaa2be14908965b4e20e5e715030746.pdf>. Acesso em: 10 out. 2016.

HIRAMA, Kechi. **Engenharia de software**: qualidade e produtividade com tecnologia. Rio de Janeiro: Elsevier, 2011.

KOCH, Richard. **O gestor 80/20.** [S.l.]: Ed. Vogais, [2014].

KOSCIANSKI, A.; SOARES, M. S. **Qualidade de software**: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2. ed. São Paulo: Novatec Editora, 2007.

LANA, F. V. D. **Desenvolvimento de software:** Comunicação X Satisfação. 2009. Disponível em: <http://www.baguete.com.br/artigos/608/francielle-venturini-dalla-lana/22/04/2009/desenvolvimento-de-software-comunicacao-x-sat>. Acesso em: 10 out. 2016.

LARMAN, Craig. **Utilizando UML e padrões**: Uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo. 3. ed. São Paulo: Bookman Editora, 2002.

PAULA FILHO, Wilson de Pádua. **Engenharia de software**. 3. ed. Ed LTC, 2003.

PHAM, Andrew; PHAM, Phuong-Van. **SCRUM em ação**: Gerenciamento e Dsenvolvimento Ágil de Projetos de Software. São Paulo: Novatec, 2012.

PhantomJS. **Poltergeist – A PhantomJS driver for Capybara.** 2016. Disponível em: <https://github.com/teampoltergeist/poltergeist>. Acesso em: 08 nov. 2016.

PRESSMAN, Roger S. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre: Mc Graw Hill, 2011.

REINHEIMER, Paul. **Conhecendo o PhantomJS. 2012.** Disponível em: <http://imasters.com.br/*framework*/conhecendo-o-phantomjs/?trace=1519021197&source=single> Acesso em: 08 nov. 2016.

RSpec; **History.** [2015]. Disponível em: <http://rspec.info/about/>. Acesso em: 08 nov. 2016.

RUBY. **Sobre o Ruby.** Disponível em: < https://www.ruby-lang.org/pt/about/> Acesso em: 15 out. 2016.

SABBAGH, Rafael. **Scrum:** Gestão ágil para projetos de sucesso. Casa do Código, 2014.

SALESFORCE COMPANY. **Heroku:** About. 2016. Disponível em: <https://www.heroku.com/about>. Acessado em: 09 nov. 2016.

SCHACH, Stephen R. **Engenharia de software**: os paradigmas clássico & orientado a objetos. 7. ed. São Paulo: Mc Graw-Hill, 2009.

SHAY, Xavier. **Robot Has Ho Heart.** 2016. Disponível em: <http://rhnh.net/2011/01/31/yaml-tutorial>. Acesso em: 08 nov. 2016.

SILVEIRA, Paulo. **Domain Specific Languages em ação**. 2007. Disponível em <http://blog.caelum.com.br/domain-specific-languages-em-acao/ >. Acesso em: 08 nov. 2016.

Software Freedom Conservancy. **Git:** About. 2011. Disponível em: <https://git-scm.com/about>. Acesso em: 08 nov. 2016.

SOMMERVILLE, Ian. **Engenharia de software**. 8. ed. São Paulo: Pearson Prentice Hall, 2007.

_____. **Engenharia de software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SOUZA, Lucas. **Ruby:** Aprenda a programar na linguagem mais divertida. São Paulo: Casa do código, [s.d].

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL:** About. 2016. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 08 nov. 2016.

VERGARA, Sylvia Constant. **Métodos de pesquisa em administração.** 4.ed. São Paulo: Atlas, 2010.

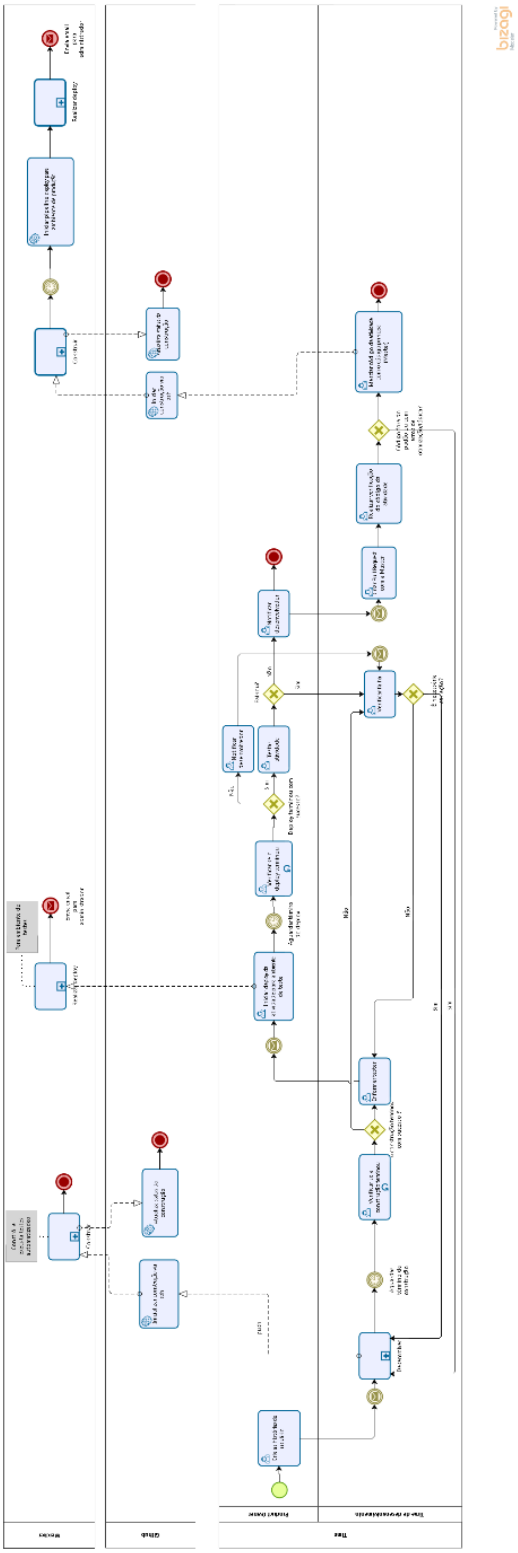
VIEIRA, Nando. **Usando RSpec para testar sua aplicação Rails – Modelo.** 2008. Disponível em: <https://nandovieira.com.br/usando-o-rspec-para-testar-sua-aplicacao-rails-modelos>. Acesso em: 08 nov. 2016.

W3SCHOOLS**. jQuery Introduction.** Disponível em <http://www.w3schools.com/jquery/jquery_intro.asp>. Acesso em: 08 nov. 2016.

WAKE, Bill. **INVEST in Good Stories, and SMART Tasks**. 2003. Disponível em: <http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>. Acesso em: 15 out. 2016.

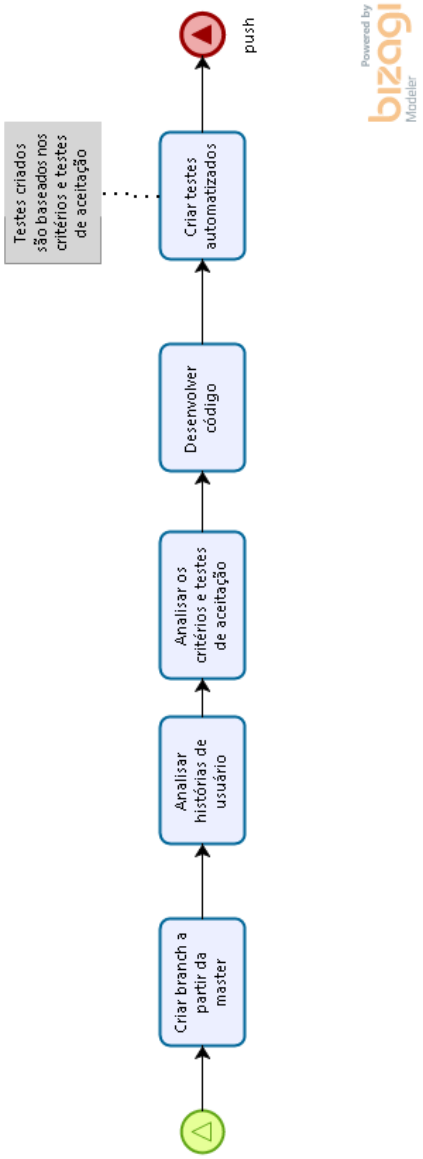
APÊNDICE A

Fluxo geral apresentando as atividades, ações e tarefas referentes a definição de pronto



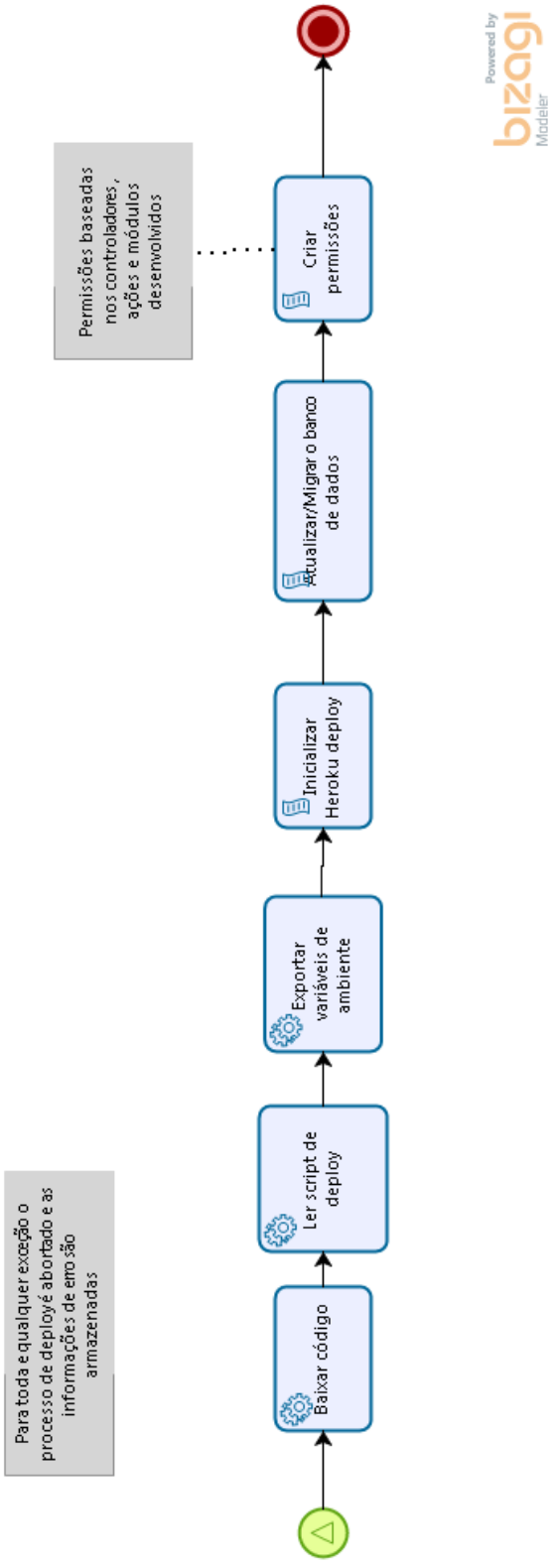
APÊNDICE B

Fluxo apresentando as atividades, ações e tarefas que o Time de desenvolvimento realizam



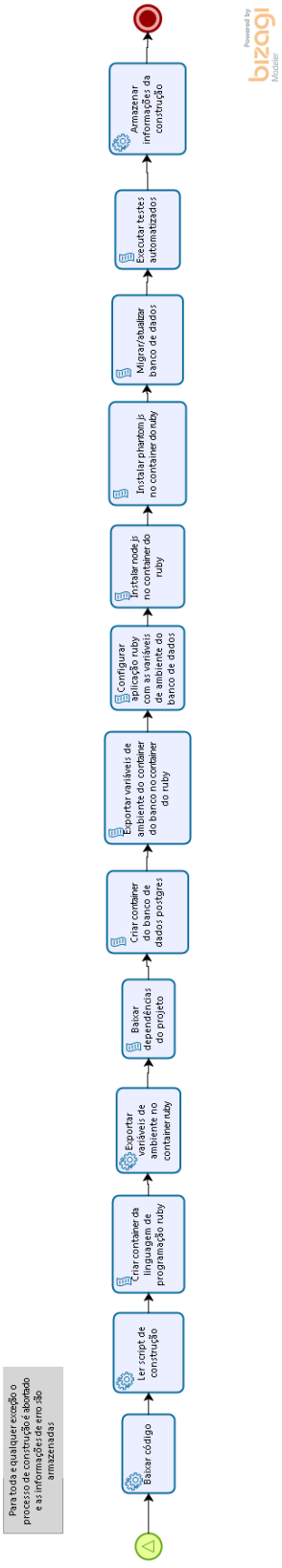
APÊNDICE C

Processo de *deploy* para o ambiente de testes



APÊNDICE D

Processo de construção no Wercker realizando *Merge* com a *Branch Master*



APÊNDICE E

Sprint #1 - Possibilitaremos que os usuários tenham acesso ao sistema

História de Usuário:

Como administrador do estabelecimento devo cadastrar os usuários que vão utilizar o sistema.

Critérios de aceitação:

- Deve ser informado, nome, CPF, email, senha e confirmação de senha.
- Pode ser inserido uma imagem.
- O CPF deve ser válido.
- O CPF não pode se repetir.
- O email não pode se repetir.
- O email deve ser válido.
- A senha deve ter de 6 a 10 caracteres.
- A senha inserida no campo senha e no campo confirmação de senha devem ser idênticas.
- Usuários não serão excluídos, apenas inativados.
- Na edição podem ser alterados todos os dados.
- Na listagem deve aparecer a opção de visualizar usuários ativos e inativos.
- Na listagem o usuário pode buscar por nome, email e CPF.
- O usuário inativado não pode acessar o sistema.
- Caso um usuário seja inativado e esteja acessando o sistema sua sessão deve ser encerrada.
- O administrador poderá alterar a senha de acesso dos usuários
- O usuário não pode se inativar.

Testes de aceitação:

Cenário: Cadastrar um novo usuário

Dado que o usuário clicou no menu administração
Dado que o usuário clicou no menu usuários
Dado que o usuário clicou em novo
Dado que o usuário está na tela de cadastro de usuário
Quando ele digita o nome
Quando ele digita o CPF
Quando ele digita o email.
Quando ele digita a senha
Quando ele digita a confirmação de senha idêntica a senha
Quando ele clica no botão salvar
Então o cadastro é realizado com sucesso
E o usuário é direcionado para a tela de visualizar e apresenta a mensagem

Cenário: Usuário, Senhas não correspondem

Dado que o usuário clicou no menu administração
Dado que o usuário clicou no menu usuários
Dado que o usuário clicou em editar em um dos usuários da listagem
Dado que o usuário está na tela de editar usuário
Quando ele digita o nome.
Quando ele digita o CPF.
Quando ele digita o email
Quando ele digita a senha
Quando ele digita a confirmação de senha diferente
Quando clica no botão salvar

Então é apresentado a mensagem “Usuário não pode ser cadastrado”
E a mensagem “Senha não corresponde com a confirmação”.

Cenário: Inativar usuário

Dado que o usuário clicou no menu administração
Dado que o usuário clicou no menu usuários
Dado que ele clicou em inativar de um dos usuários da listagem
Quando apresentou a mensagem “Você tem certeza? ”, ele confirmou
Então o usuário é inativado

História de Usuário:

Como proprietário do estabelecimento quero que meus funcionários tenham que realizar login para acessar o sistema.

Critérios de aceitação:

- Para acessar o sistema o usuário deve informar o CPF e senha.
- No primeiro acesso o usuário deve alterar a senha, confirmar a senha inserida e informar uma palavra secreta.
- O usuário inativado não pode acessar o sistema.

Testes de aceitação:

Cenário: Usuário acessa o sistema

Dado que o usuário está na tela de login
Quando ele digita o CPF.
Quando ele digita a senha
Quando ele clica no botão salvar
Então é direcionado para a tela inicial do sistema..

Cenário: Usuário inativado tenta acessar o sistema

Dado que o usuário está na tela de login
Quando ele digita o CPF
Quando ele digita a senha
Quando ele clica no botão salvar
Então é apresentado a mensagem “O usuário informado está inativo”.

Cenário: Usuário erra CPF

Dado que o usuário está na tela de login
Quando ele digita o CPF errado.
Quando ele digita a senha
Quando ele clica no botão salvar
Então é apresentado a mensagem “CPF e/ou senha inválidos”.

História de Usuário:

Como funcionário do estabelecimento quero conseguir trocar minha senha quando eu quiser, pois gosto de troca-las com frequência .

Critérios de aceitação:

- O usuário poderá alterar a senha
- Ao alterar a senha o usuário deve confirmar a senha
- O usuário não pode reutilizar as 4 últimas senhas

- O usuário poderá alterar a palavra secreta
- O sistema deve armazenar a palavra secreta em minúsculo e sem acentuação
- O usuário poderá alterar a imagem

Testes de aceitação:

Cenário: Usuário altera sua senha sem repetir as 4 ultimas senhas

Dado que o usuário está na tela inicial do sistema
Dado que o usuário clica em seu nome do canto superior direito
Dado que o usuário clica em editar perfil
Quando ele digita a senha
Quando ele confirma a senha
Quando ele clica no botão salvar
Então é direcionado para a tela inicial e apresentado a mensagem "Usuário editado com sucesso.".

Cenário: Usuário altera sua senha inserindo uma das 4 ultimas senhas utilizadas recentemente

Dado que o usuário está na tela inicial do sistema
Dado que o usuário clica em seu nome do canto superior direito
Dado que o usuário clica em editar perfil
Quando ele digita uma senha que já foi utilizada recentemente por ele
Quando ele confirma a senha
Quando ele clica no botão salvar
Então é direcionado para a tela inicial e apresentado a mensagem "Essa senha já foi utilizada recentemente.".

História de Usuário:

Como gerente do estabelecimento quero que os funcionários consigam recuperar a senha para acessar o sistema sem necessidade de contatar-me.

Critérios de aceitação:

- O usuário deve informar seu CPF, email e palavra secreta
- O sistema deve informar uma nova senha
- Ao logar, utilizando a nova senha o usuário deve alterar senha, confirmá-la e poderá alterar a palavra secreta.

Testes de aceitação:

Cenário: Usuário esqueceu sua senha

Dado que o usuário está na tela de login
Dado que o usuário clica em esqueceu sua senha?
Quando ele digita o CPF
Quando ele digita o email
Quando ele digita a palavra secreta
Quando ele clica no botão resetar a senha
Então o sistema compara as informações, para compara a palavra secreta é necessário deixá-la em minúsculo e sem acentuação
Então apresenta a mensagem Senha alterada com sucesso! Nova senha:
Então apresenta a nova senha

Cenário: Usuário realiza login com a nova senha

Dado que o usuário está na tela de login
Quando ele digita o CPF corretamente
Quando ele digita a senha que foi disponibilizada pelo sistema quando ele informou que esqueceu sua senha
Quando ele clica no botão resetar a senha

Então o usuário deve alterar sua senha, confirmá-la e pode alterar a palavra secreta.

APÊNDICE F

Sprint #2 - Possibilitaremos que o administrador possa definir o que cada funcionário pode acessar no sistema

História de Usuário:

Como administrador do estabelecimento devo determinar as ações que cada funcionário pode acessar no sistema.

Critérios de aceitação:

- O administrador terá acesso a todas as funcionalidades disponíveis no sistema
- O administrador deverá determinar para cada usuário quais ações estarão disponíveis para ele acessar
- A ação de editar perfil (imagem, senha, confirmação de senha e palavra secreta) devem estar disponíveis para todos os usuários
- A alteração de senha, confirmação de senha e palavra secreta ao acessar o sistema a primeira vez ou após ter esquecido sua senha devem estar disponíveis para todos os usuários
- Caso uma ação não esteja disponível para um determinado usuário, o sistema não poderá aceitar acessar pela URL.
- Quando o usuário executar uma ação que ele não tenha permissão deve ser apresentado uma mensagem informando "Acesso negado".

Testes de aceitação:

Cenário: Administrador quer que o usuário possa dar permissões para outros usuários

Dado que o administrador realizou login.
Dado que o administrador clica em Administração.
Dado que o administrador clica em Usuários.
Dado que o administrador clica em Visualizar do usuário que deseja dar permissão
Dado que o administrador clica em Permissões.
Dado que o administrador clica em Administração.
Dado que o administrador clica em Permissões.
Quando o administrador selecionar todos os itens referentes a permissões
Então o usuário selecionado poderá atribuir novas permissões para os usuários.

Cenário: Usuário não tem permissão, mas tenta cadastrar um novo usuário pela URL

Dado que o usuário realizou login.
Dado que o usuário não tem permissão para cadastrar um usuário
Quando o usuário acessa pela URL
Então é apresentado a página de erro "403 – Acesso negado".

Cenário: Usuário tem permissão para cadastrar usuário, mas não tem permissão para visualizá-lo

Dado que o usuário clicou no menu administração
Dado que o usuário clicou no menu usuários
Dado que o usuário clicou em novo
Dado que o usuário está na tela de cadastro de usuário
Quando ele digita o nome corretamente.
Quando ele digita o CPF corretamente.
Quando ele digita o email corretamente.
Quando ele digita a senha
Quando ele digita a confirmação de senha idêntica a senha
Quando ele clica no botão salvar
Então o cadastro é realizado com sucesso
Então é apresentado a página de erro "403 – Acesso negado".

APÊNDICE G

Sprint #3 - Possibilitaremos que o controle de entrada e saída dos clientes sejam realizados através do CPF ou Leitura biométrica

História de Usuário:

Como recepcionista do estabelecimento desejo que o cliente selecione se deseja utilizar CPF e senha ou leitura biométrica para realizar a entrada no estabelecimento.

Critérios de aceitação:

O cliente pode utilizar a leitura biométrica para dar entrada no estabelecimento

- O cliente pode utilizar o CPF e uma senha para dar entrada no estabelecimento
- Caso opte por utilizar CPF e senha a senha deve ser confirmada
- A senha deve ser numérica com 6 dígitos
- Deve ser inserido a data e hora da entrada no estabelecimento
- Deve haver controle de permissão referente as ações

Testes de aceitação:

Cenário: Entrada utilizando leitura biométrica

Dado que a recepcionista clicou no ícone referente a entrada
Dado que a recepcionista clicou no ícone referente a leitura digital
Dado que o cliente posicionou o dedo no leitor biométrico
Quando a biometria for capturada pelo scanner
Então é apresentada a mensagem "Entrada confirmada, bem vindo(a)"

Cenário: Entrada, erro com a leitura biométrica

Dado que a recepcionista clicou no ícone referente a entrada
Dado que a recepcionista clicou no ícone referente a leitura digital
Dado que o cliente posicionou o dedo no leitor biométrico
Quando a biometria não foi capturada pelo scanner
Então é apresentada a mensagem "Erro na leitura, por favor tente novamente"

Cenário: Entrada utilizando CPF e senha válidos

Dado que a recepcionista clicou no ícone referente a entrada
Dado que a recepcionista clicou no ícone referente a entrada utilizando CPF e senha
Dado que o cliente digitou o CPF
Dado que o cliente digitou a senha
Quando que o cliente clica em entrar
Então apresenta a mensagem "Entrada confirmada, bem-vindo (a)"

Cenário: Entrada utilizando CPF e senha válidos, primeira vez que é utilizado

Dado que a recepcionista clicou no ícone referente a entrada
Dado que a recepcionista clicou no ícone referente a entrada utilizando CPF e senha
Dado que o cliente digitou o CPF
Dado que o cliente digitou a senha
Quando que o cliente clica em entrar
Então o CPF deve ser validado
Então o cliente deve inserir a senha novamente
Então o cliente clica em entrar
Então apresenta a mensagem "Entrada confirmada, bem-vindo (a)"

História de Usuário:

Como recepcionista do estabelecimento desejo que o cliente selecione se deseja utilizar CPF e senha ou leitura biométrica para realizar a saída do estabelecimento.

Critérios de aceitação:

- A saída do pode ser realizada através da leitura biométrica
- A saída pode ser realizada através do CPF e uma senha
- Caso opte por utilizar CPF e senha a senha deve ser confirmada
- A senha deve ser numérica com 6 dígitos idêntica a senha utilizada na entrada
- Deve ser inserido a data e hora de saída no estabelecimento
- Deve haver controle de permissão referente as ações

Testes de aceitação:

Cenário: Saída utilizando leitura biométrica

Dado que a recepcionista clicou no ícone referente a saída
Dado que a recepcionista clicou no ícone referente a leitura digital
Dado que o cliente posicionou o dedo no leitor biométrico
Quando a biometria for capturada pelo scanner
Então verifica se a digital scanneada deu entrada no estabelecimento
Então é apresentada a mensagem “Saída confirmada”

Cenário: Saída erro com a leitura biométrica

Dado que a recepcionista clicou no ícone referente a saída
Dado que a recepcionista clicou no ícone referente a leitura digital
Dado que o cliente posicionou o dedo no leitor biométrico
Quando a biometria não foi capturada pelo scanner
Então é apresentada a mensagem “Erro na leitura, por favor tente novamente”

Cenário: Entrada usando CPF e senha saída utilizando leitura biométrica

Dado que a recepcionista clicou no ícone referente a saída
Dado que a recepcionista clicou no ícone referente a leitura digital
Dado que o cliente posicionou o dedo no leitor biométrico
Quando a biometria for capturada pelo scanner
Então verifica se a digital scanneada deu entrada no estabelecimento
Então é apresentada a mensagem “Não há entrada registrada utilizando essa biometria”

Cenário: saída utilizando CPF e senha válidos

Dado que a recepcionista clicou no ícone referente a saída
Dado que a recepcionista clicou no ícone referente a saída utilizando CPF e senha
Dado que o cliente digitou o CPF
Dado que o cliente digitou a senha
Quando que o cliente clica em sair
Então verifica o CPF e senha deu entrada no estabelecimento
Então apresenta a mensagem “Saída confirmada”

APÊNDICE H

Sprint #4 - Possibilitaremos que funcionários realizem a entrada de produtos nos estoques do estabelecimento

História de Usuário:

Como gerente do estabelecimento quero nomear o estoque para ficar mais fácil de identificar a localidade dos produtos

Critérios de aceitação:

- Deve ser informado um nome;
- O nome do estoque não pode ser duplicado
- Pode ser informada uma descrição
- Haverá apenas um estoque principal
- Deve haver controle de permissão referente as ações

Testes de aceitação:

Cenário: Cadastro de estoque

Dado que o gerente clicou no ícone referente a patrimônio
Dado que o gerente clicou no ícone referente a estoques
Dado que o gerente clicou no ícone novo
Quando o gerente digita um nome
Quando o gerente digita uma descrição
Então verifica se o nome não é duplicado
Então direciona para o visualizar do estoque
E é apresentada a mensagem "Estoque cadastrado com sucesso."

Cenário: Cadastro de estoque, nome duplicado

Dado que o gerente clicou no ícone referente a patrimônio
Dado que o gerente clicou no ícone referente a estoques
Dado que o gerente clicou no ícone novo
Quando o gerente digita um nome
Quando o gerente digita uma descrição que já existe no sistema
Então é apresentada a mensagem "Estoque não pode ser cadastrado."
E é apresentada a mensagem "já está em uso."

História de Usuário:

Como gerente do estabelecimento quero cadastrar os produtos que são vendidos no estabelecimento

Critérios de aceitação:

- Deve ser informado o nome;
- O nome do produto não pode ser duplicado.
- Pode ser informado o código de barras
- Pode ser informado o valor de venda do produto
- Deve haver controle de permissão referente as ações

Testes de aceitação:

Cenário: Cadastro de produto

Dado que o gerente clicou no ícone referente a patrimônio

Dado que o gerente clicou no ícone referente a produtos
Dado que o gerente clicou no ícone novo
Quando o gerente digita um nome
Então verifica se o nome não é duplicado
Então direciona para o visualizar do produto
E é apresentada a mensagem “Produto cadastrado com sucesso”

Cenário: Cadastro de estoque, nome duplicado

Dado que o gerente clicou no ícone referente a patrimônio
Dado que o gerente clicou no ícone referente a produtos
Dado que o gerente clicou no ícone novo
Quando o gerente digita o nome que já existe no sistema
Então é apresentada a mensagem “Produto não pode ser cadastrado.”
E é apresentada a mensagem “já está em uso.”

História de Usuário:

Como funcionário do estabelecimento quero inserir produtos nos estoques para conseguir identificar suas localidades

Critérios de aceitação:

- Deve ser selecionado um dos produtos já cadastrados;
- Deve ser informado a quantidade de produtos;
- A quantidade informada deve ser soma a quantidade anterior
- Deve haver controle de permissão referente as ações

Testes de aceitação:

Cenário: Cadastro de produto

Dado que o funcionário clicou no ícone referente a patrimônio
Dado que o funcionário clicou no ícone referente a estoques
Quando o funcionário clicou no ícone novos produtos apresentados nos itens da listagem de estoque
Quando o funcionário seleciona o produto
Quando o funcionário digita a quantidade de produtos
Então a quantidade inserida é somada a quantidade anterior
Então direciona para o visualizar de estoques
E é apresentada a mensagem “Produtos adicionados ao estoque com sucesso”

Cenário: Cadastro de estoque, nome duplicado

Dado que o gerente clicou no ícone referente a patrimônio
Dado que o gerente clicou no ícone referente a produtos
Dado que o gerente clicou no ícone novo
Quando o gerente digita o nome que já existe no sistema
Então é apresentada a mensagem “Produto não pode ser cadastrado.”
E é apresentada a mensagem “já está em uso.”