IVAN RIBEIRO DE CARVALHO FILHO

RELATÓRIO DE ESTÁGIO NA EMPRESA EQUALS:

DESENVOLVIMENTO DE SISTEMA WEB

LAVRAS – MG

2021

IVAN RIBEIRO DE CARVALHO FILHO

RELATÓRIO DE ESTÁGIO NA EMPRESA EQUALS:
DESENVOLVIMENTO DE SISTEMA WEB

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Raphael Winckler de Bettio
Orientador

LAVRAS – MG
2021

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Filho, Ivan Ribeiro de Carvalho

Relatório de estágio na empresa Equals : Desenvolvimento de sistema web / . – Lavras : UFLA, 2021.

51 p. : il.

Relatório de Estágio (Graduação)–Universidade Federal de Lavras, 2021.

Orientador: Prof. Raphael Winckler de Bettio.

Bibliografia.

1. TCC. 2. Monografia. 3. Dissertação. 4. Tese. 5. Trabalho Científico – Normas. I. Universidade Federal de Lavras. II. Título.

CDD-808.066

AGRADECIMENTOS

Agradeço a minha mãe e irmã, por todo apoio ao longo da graduação.

Agradeço aos meus amigos, em especial: Nicolas Bicalho, Danilo Guarizzo, Gustavo Costa, Gustavo Reis, Jean Roberto, Bruno Amaral, Leonardo Campelo, Gabriel Nori e Emanuel Júnior por me acompanharem na graduação do início ao fim.

À Comp Júnior, que me ensinou a ir além e fazer acontecer, pelas portas que se abriram e pelas amizades que ganhei.

Por fim, agradeço meu orientador Raphael Winckler de Bettio por estar presente comigo desde o início da graduação, fazendo a diferença não só no âmbito acadêmico, mas no profissional também.

RESUMO

O objetivo deste documento é relatar as atividades desenvolvidas durante um estágio desempenhado na empresa Equals, uma *fintech* de gestão financeira e conciliação de vendas a crédito. Durante o estágio pude trabalhar com as linguagens de programação Java, JSP, JavaScript, HTML, CSS e SQL. Também coloquei em prática os conceitos da metodologia ágil Scrum. Neste documento faço a comparação dos aprendizados do estágio com os concebidos durante a graduação, incluindo as disciplinas de programação web, processos de software, banco de dados e programação orientada a objetos.

Palavras-chave: Equals. Desenvolvimento Web. Metodologias ágeis.

ABSTRACT

This document consists of reporting the activities developed during the internship performed at Equals company, a fintech of financial management and credit sales reconciliation. During internship I could work with Java, JSP, JavaScript, HTML, CSS and SQL programming languages. I also put into practice the concepts of agile methodology Scrum. In this document I will compare the earned knowledges of the internship with those conceived during graduation, including the disciplines of web programming, software processes, database and object-oriented programming.

Keywords: Equals. Web development. Agile methodologies.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O curso de bacharelado em Ciência da Computação busca capacitar os alunos em habilidades referentes ao desenvolvimento de software. Esse curso permite que o aluno realize um estágio na área e relate a experiência obtida em um documento a ser utilizado como Trabalho de Conclusão de Curso (TCC). As experiências devem ser correlacionadas às disciplinas cursadas ao longo da graduação, a fim de estender os aprendizados dessas disciplinas evidenciando as experiências práticas obtidas foras da sala de aula.

Sendo assim, este documento descreve um relatório de estágio realizado pelo autor, no período de 21/12/2017 a 23/04/2018. O relatório conta como era o dia a dia dentro da empresa, dando um foco nas competências voltadas ao desenvolvimento de sites, mas não fica restrito a isso, incluindo também outras áreas do curso como criação e manutenção de bancos de dados, elaboração de sistemas distribuídos, versionamento de código e uso de metodologias ágeis.

A empresa, Equals (EQUALS, 2021), na qual foi realizado o estágio é uma *fintech*[1] que atua há mais de 14 anos no mercado, possui um escritório em Lavras e um outro em São Paulo (capital), foi fundada com o objetivo de desenvolver soluções para melhorar a gestão financeira das empresas de médio e grande porte. Sua principal aplicação é responsável por conciliar e simplificar informações sobre as vendas para o lojista. O processo de conciliação consiste em verificar se as taxas contratadas em uma maquininha de cartão de crédito foram aplicadas com os valores corretos pela bandeira dona daquela maquininha. Esse processo pode ser facilmente realizado de forma manual, mas a plataforma Equals vem com a solução de automatizar esse processo a fim de executa-lo em larga escala e em questão de segundos.

Os demais capítulos deste documento estão organizados da seguinte forma:

- Capítulo 2: Descrição das tecnologias e ferramentas utilizadas;

[1] Empresa do ramo financeiro

- Capítulo 3: Descrição do sistema, de como era o dia a dia de trabalho e como as tecnologias eram utilizadas;
- Capítulo 4: Conclusão e avaliação do estágio relatado.

2 TECNOLOGIAS

Este capítulo descreve as tecnologias, termos técnicos e ferramentas mencionadas ao longo deste documento.

2.1 Frontend e Backend

Frontend e *backend* são palavras usadas para definir áreas de atuação de um desenvolvedor. Também existe o termo *fullstack*, que designa o profissional que atua em ambas áreas, *frontend* e *backend*, dentro de um time (W3SCHOOLS, 2021).

A área *frontend* contempla a manipulação de elementos de interface gráfica da aplicação. Tipicamente faz o uso das linguagens CSS[1], HTML[2] e JavaScript para atingir esse propósito. Seu código é executado na máquina do cliente. Sua aplicação mais comum é em sites e aplicativos móveis (FRONTENDMASTERS, 2019).

A área de *backend* contempla a manipulação de dados e processamento de requisições. Normalmente é utilizado para formatar as informações do banco de dados para a aplicação *frontend* e também aplicar regras de negócio que cercam o projeto. Para essa área podemos encontrar exemplos das linguagens Java, C# e JavaScript para desenvolver suas aplicações (TECHTERMS, 2020a).

2.2 Metodologias ágeis

As metodologias ágeis surgiram na indústria de Tecnologia da Informação para resolver os problemas das organizações que precisam gerenciar projetos.

Segundo Nathália Tameirão (TAMEIRÃO, NATHÁLIA, 2021), foi em 2001 que um grupo composto por 17 pessoas se reuniu para debater as abordagens usadas em gerenciamento de projetos e criaram o chamado Manifesto Ágil,

[1] Cascading Style Sheets
[2] Hyper Text Markup Language

que oficializa a existência das metodologias e estabelece princípios que as caracterizam.

A partir desse documento (TAMEIRÃO, NATHÁLIA, 2021), pode-se dizer que os princípios mais importantes e que orientam a aplicação de um método ágil são:

- Comunicação: onde os indivíduos e a interação entre eles devem ser mais importantes que processos e ferramentas;
- Praticidade: um software em funcionamento é mais importante que a documentação;
- Alinhamento de expectativas e colaboração: a colaboração com o cliente e membros do projeto deve ser mais importante que a negociação de contratos;
- Adaptabilidade e flexibilidade: responder a mudanças é mais importante que seguir um plano.

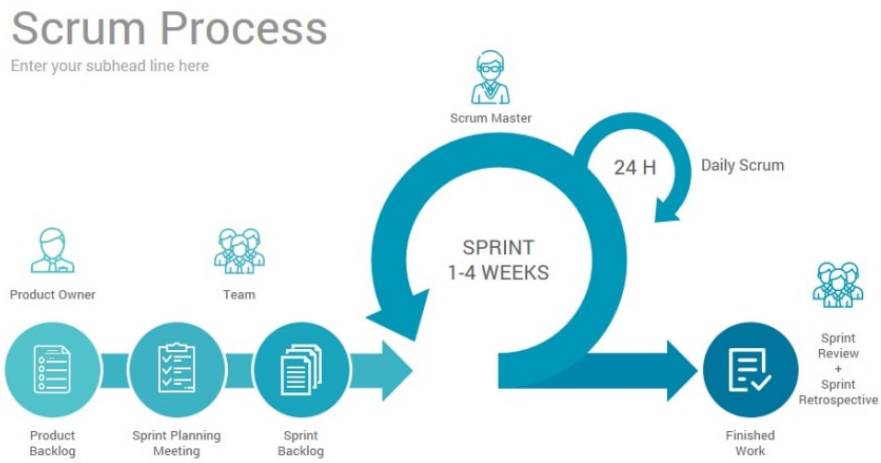
A metodologia ágil adotada neste estágio foi a *Scrum*, que possui os seguintes processos: *Product backlog, Sprint Planning, Sprint Backlog, Sprint, Daily Scrum, Sprint Review* e por fim *Sprint Retrospective*. De forma geral, o fluxo pode ser observado na figura 2.1, mas nos tópicos seguintes cada processo estará descrito de forma mais detalhada.

2.2.1 Daily Scrum

Em projetos onde mudanças acontecem constantemente e o trabalho é feito em pequenos ciclos, é essencial acompanhar constantemente as atividades realizadas e por isso, temos a *Daily Scrum*.

Thiago Coutinho (COUTINHO, THIAGO, 2019a) define a *Daily Scrum* como uma reunião que deve ser realizada de forma rápida, com duração máxima de 15 minutos. Ela deve ser realizada todos os dias de desenvolvimento da *Sprint*

Figura 2.1 – Processos do *Scrum*



red(Alterei a fonte da img) Fonte: https://dev.to/eudaimonia_ar/improve-your-performance-with-scrum-methodology-idg

e visa repassar as atividades do dia anterior, além de planejar as atividades que serão realizadas no dia. Durante a *Daily Scrum*, todos os membros do time devem responder três perguntas[3]:

- O que eu fiz desde a última *Daily Scrum*?
- O que vou fazer hoje?
- Quais são os impedimentos que estão me atrapalhando a realizar meu trabalho?

2.2.2 Grooming

Para aprimorar o *Product Backlog*, reuniões de *Backlog Grooming* são realizadas.

[3] Respondido essas três perguntas, a equipe pode se organizar para resolver os impedimentos e dar andamento no projeto, além de poder saber o quão perto da meta da *Sprint* o Time de Desenvolvimento está. Vale ressaltar que a resolução desses impedimentos levantados durante a reunião não é feita durante a reunião, mas sim após ela com as pessoas envolvidas.

Segundo Kleber Bernardo (BERNARDO, KLEBER, 2015), durante uma reunião de *Backlog Grooming*, a equipe e o *Product Owner* devem discutir os principais itens do *Backlog* a fim de facilitar a discussão e adiantar possíveis perguntas que podem surgir durante o Planejamento da *Sprint*.

Já Robson Camargo (CAMARGO, ROBSON, 2019a) define que "o *Grooming* nada mais é que um momento específico em que se iniciam os preparativos para a próximo *Sprint* ou *Release*".

Enquanto o time trabalha na *sprint* atual, o *Product Owner*, com mais alguns convidados da equipe, começa a refinar e amadurecer as histórias e objetivos da próxima *Sprint* ou *release*.

Robson Camargo (CAMARGO, ROBSON, 2019a) também diz que podemos definir um *grooming* como sendo uma "preparação da lista de pendências", ou seja, um refinamento da lista de pendências para preparar os itens de *backlog* de *Sprints* futuras, assim evitando também a ocorrência de impedimentos no decorrer das *Plannings* ao ser repassado item por item da lista de pendências.

2.2.3 Sprint Planning

Robson Camargo (CAMARGO, ROBSON, 2019b) diz que durante a reunião de *Sprint Planning*, o *Product Owner* deve descrever as funcionalidades de maior prioridade para a equipe. E a equipe deverá fazer perguntas durante a reunião de modo que, possa tirar suas dúvidas e dividir as funcionalidades das histórias em tarefas técnicas. Essas tarefas irão dar origem ao *Sprint Backlog*.

Depois de passar por todas as funcionalidades, dividir todas as funcionalidades, o time e o *Product Owner* devem definir um objetivo para a *Sprint*, que é uma breve descrição daquilo que se tentará alcançar na *Sprint* (CAMARGO, ROBSON, 2019b).

De acordo com Vitor (VITOR L. MASSARI, 2016) a principal regra da *Sprint Planning* é: "Jamais uma reunião de planejamento da *Sprint* deve ser adi-

ada ou começar sem um *Product Backlog* priorizado". O *Product Backlog* é um item crucial para que a *Planning* ocorra, pois usamos ele para gerar o *Sprint Backlog*. Sendo assim o *Product Backlog* se torna um item básico para execução da *Planning*. Ademais, não basta ter um *Backlog* definido, ele deve estar com as atividades bem descritas para serem discutidas durante a reunião de forma proveitosa, sem perder muito tempo. Caso contrário a *Sprint* não terá um conjunto bom de atividades para serem desenvolvidas. Outro ponto da regra do Vitor é o fato de que a reunião nunca deve ser adiada, um time sem uma *Sprint* definida é um time sem estar produzindo. Caso a ausência do PO para guiar a *Planning*, podemos usar o *Scrum Master* ou então o líder técnico do time.

2.2.4 Scrum Poker

Com o objetivo de ter uma maior exatidão nos prazos para concluir os projetos, uma equipe pode usar o *Scrum Poker* (também conhecido como *Planning Poker*), uma metodologia baseada em consenso.

Segundo Técia (CARVALHO, TÉCIA, 2019), podemos definir o *Scrum Poker* como "uma estratégia usada pelas equipes em projetos ágeis para buscar uma estimativa via consenso da equipe. A ferramenta foi definida e nomeada inicialmente por James Grenning, em 2002, mas foi com o livro *Agile Estimating and Planning*, de Mike Cohn, que ela se popularizou no mundo de projetos. Em um jogo *Planning Poker*, cada membro da equipe envolvido no desenvolvimento deverá receber um conjunto de cartas com os valores de uma sequência definida pela própria equipe". Posteriormente, eles deverão discutir entre si sobre as atividades que precisam fazer na *sprint* e, ao final do jogo, eles determinam uma estimativa de tempo e esforço necessário para realizar as histórias do *Product Backlog* com base nas cartas.

Vitor (VITOR L. MASSARI, 2016) destaca que alguns times tem o hábito contabilizar os pontos inicialmente em uma métrica arbitrária e no final da

Planning converter essa pontuação acumulada em horas, equivalendo um ponto estimado a 4 horas de desenvolvimento, por exemplo.

2.2.5 Sprint

Como dito anteriormente, as Metodologias ágeis de desenvolvimento de software são iterativas, ou seja, o trabalho é dividido em iterações, que são chamadas de *Sprints* no caso do *Scrum*. Carlos Júnior (JUNIOR, CARLOS, 2017) nos diz que uma *Sprint* representa um tempo dentro do qual um conjunto de atividades deve ser executado.

Ainda segundo Carlos Júnior (JUNIOR, CARLOS, 2017), "uma *sprint* pode ser considerado o principal evento do método *Scrum*, porque é nela que vamos aplicar todos os demais eventos realizados no *Scrum* e desenvolveremos de fato o produto". É Durante a *sprint* que realmente ocorre a produção de um produto ou parte dele.

As funcionalidades a serem implementadas em um projeto são mantidas em uma lista que é conhecida como *Product Backlog*, que será apresentada posteriormente. Como foi apresentado, durante uma *Sprint Planning* a equipe seleciona as atividades que ela será capaz de implementar durante a *Sprint* que se inicia (JUNIOR, CARLOS, 2017).

2.2.6 Burndown

Luiz Duarte (DUARTE, LUIZ, 2021) define que o *Burndown* não é uma invenção dos métodos ágeis, ele já era uma ferramenta popular para acompanhamento da diminuição de um recurso ou métrica ao longo do tempo.

Já Robson Camargo (CAMARGO, ROBSON, 2020) nos diz que o *Burndown* é uma ferramenta do *Scrum* que permite aos gerentes de projeto e à equipe verificar se o trabalho está dentro do esperado no que se refere ao cronograma.

O fato é que podemos definir que um Gráfico de *Burndown* nos mostra a relação de trabalho a ser realizado versus o tempo que ainda resta para fazê-lo. Além disso, ele pode ser macro a ponto de englobar todo o ciclo de desenvolvimento de uma *release* do produto ou micro a ponto de englobar apenas o trabalho de uma pessoa, sendo incomum este uso, mas é possível para metas pessoais.

2.2.7 Product Backlog

Segundo Thiago Coutinho (COUTINHO, THIAGO, 2019b), o *Product Backlog* pode ser definido como uma lista de todos os requisitos que precisam estar no produto. Essa lista está em constante mudança e é sempre atualizada, sendo que pode ser organizada de uma forma que vá de itens mais importantes e detalhados até itens em que se tem uma visão não tão clara ainda.

Isso permite que o projeto seja constantemente adaptado a mudanças, que é uma característica dos projetos *Scrum*. O *Product Owner* é o único responsável por gerenciar, definir e ordenar os itens do *Product Backlog* (COUTINHO, THIAGO, 2019b).

2.2.8 Retrospectiva

A instituição Scrum Org (SCRUM ORG, 2021), define que o objetivo da Retrospectiva da *Sprint* é planejar maneiras de aumentar a qualidade e eficácia. O Time *Scrum* deve inspecionar como foi a última *Sprint* em relação aos indivíduos, interações, processos, ferramentas e sua Definição de Feito. As suposições que os desencaminharam deverão ser identificadas e suas origens exploradas. O Time deverá discutir o que deu certo durante o *Sprint*, quais problemas foram encontrados, como esses problemas foram (ou não) resolvidos e o que podem fazer para que não voltem a acontecer. O Time deverá identificar as mudanças mais úteis para melhorar sua eficácia.

A Retrospectiva da *Sprint* conclui a *Sprint*. É limitada a no máximo três horas, quando se trata de uma *Sprint* de um mês. Para *Sprints* mais curtas, essa reunião também é mais curta.

Ainda segundo a Scrum Org (SCRUM ORG, 2021), durante a Retrospectiva da *Sprint*, a equipe deverá discutir:

- O que deu certo no *Sprint*
- O que poderia ser melhorado
- O que vamos nos comprometer a melhorar na próxima *Sprint*

2.2.9 Product Owner (PO)

Rafael Vieira (VIEIRA, RAFAEL, 2017) define o *Product Owner*, ou PO, como o membro do time responsável por definir histórias e priorizar o *backlog* de um produto ou projeto. Ele deve manter a integridade conceitual das novas funcionalidades, *bugs* ou melhorias, para que essas sigam uma visão definida para o produto ou projeto. Além disso, ele também é responsável pela qualidade final das entregas, sendo o único que deve ter poder de aceitar as histórias como concluídas.

O PO é como o elo entre a equipe de desenvolvimento e os clientes, e ele deve colaborar de perto com ambos os grupos para garantir que há uma compreensão clara de quais recursos são necessários no produto ou aplicação. O *Product Owner* deve ter uma sólida compreensão e domínio do negócio e das diferentes necessidades de diferentes tipos de usuários (VIEIRA, RAFAEL, 2017).

2.2.10 Analista de Qualidade (QA ou tester)

Segundo Ludmila Vilaverde (VILAVERDE, LUDMILA, 2019), o Analista de *Quality Assurance* (também chamado de Analista de Qualidade ou Analista de Controle de Qualidade) é o profissional responsável por realizar testes, procurar erros e melhorias e reportá-los para as equipes de desenvolvimento, além

de sugerir estratégias de otimização para programas e sistemas. Afinal, o sistema que será entregue deverá ser funcional, otimizado e satisfazer as necessidades da empresa ou de seus clientes, de forma confiável e acessível.

A organização Kalendae (KALENDAE, 2019) complementa o conceito de *Quality Assurance* (QA) definindo que ele faz referência a um profissional ou uma equipe cuja função é garantir a qualidade no desenvolvimento de um produto ou serviço. E que deve ter na sua atuação a checagem do cumprimento de certos critérios e métodos ao longo dos processos operacionais.

2.3 Ferramentas

2.3.1 Jira

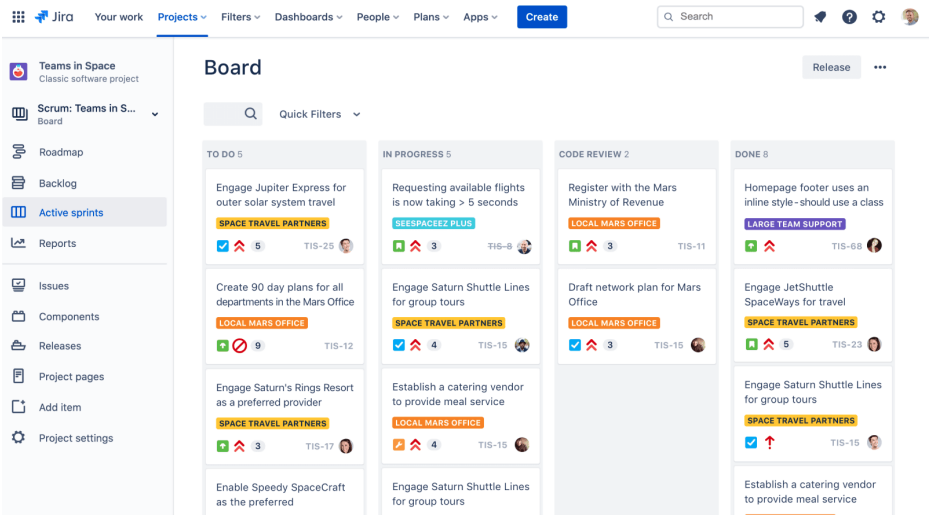
Jira é uma plataforma online que permite criação de *boards* seguindo as metodologias ágeis: *Scrum*, *Kanban* e qualquer outra, pois permite o alto nível de customização de seus componentes a fim de adaptar bem a qualquer necessidade do processo de desenvolvimento de um time (ATLASSIAN, 2021). No jira é possível criar *cards* de uma atividade e permitir que o time todo acompanhe o progresso dessa atividade dentro do processo de desenvolvimento estabelecido pela equipe, como é possível ver pela figura 2.2.

2.3.2 GitHub

O GitHub é uma plataforma com diversas ferramentas que auxiliam o desenvolvimento de código em grupo, sua ferramenta mais importante é o repositório de código com o gerenciamento de versões proporcionado pela ferramenta Git. Um repositório pode conter diversas *branchs*, e cada *branch* pode conter diversos *commits* (SOFTWARE FREEDOM CONSERVANCY, 2021) (GITHUB, 2021).

Andrei (ANDREI L., 2021) define a nomenclatura *branch* sendo: "ramificações de recursos (ou feature branches). Isso significa que cada engenheiro de

Figura 2.2 – Exemplo de um *board* do jira



Fonte: https://www.atlassian.com/software/jira

software na equipe pode separar uma ramificação de recursos que oferece um repositório local isolado para promover mudanças para os códigos". Essas mudanças de código que comenta são os *commits*, que além da linhas de código alteradas também contém um *hash* para identificação única do *commit* dentro da *branch* e uma mensagem de texto com o título resumindo qual a alteração proporcionada pelo *commit*.

2.3.3 Wildfly

Wildfly, também conhecido como JBoss, é um projeto de código aberto suportado pela empresa Red Hat. Ele é um servidor flexível e leve que te ajuda a manejar aplicações em tempo de execução. O Wildfly é capaz hospedar online APIs em Java e *Servlets applications* (RED HAT, 2021).

No Wildfly conseguimos servir e fazer *deploys de aplicações Java, assim como seus concorrentes*[4]. *Ele é amplamente utilizado junto a versão 7 e 8 da*

[4] Tomcat e GlassFish

linguagem Java. Já as versões mais novas do Java contam com um outro servidor de aplicação, o Springboot.

2.4 Linguagens

As linguagens de programação empregaram um papel extremamente importante para o desenvolvimento das atividades do estágio.

2.4.1 Java

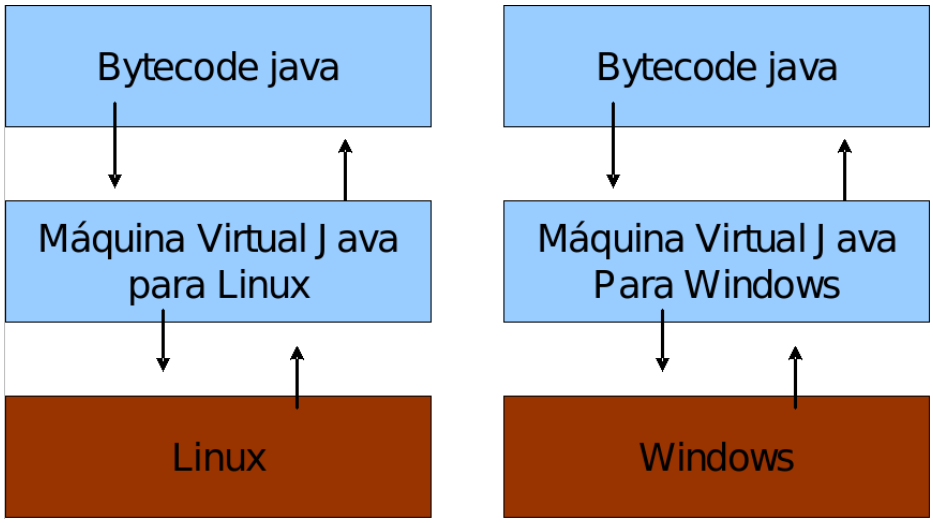
Java é uma linguagem de programação muito bem estabelecida e madura no mercado de trabalho hoje em dia. É amplamente utilizada para aplicações *backend* e aplicações móveis para sistemas Android. A linguagem segue o paradigma de programação orientada a objetos. Ela acaba sendo uma ótima opção para quem esta começando a programar, pois possui elementos de linguagem-base, mas também pode se tornar poderosa com uso de *frameworks* que incrementam muito bem o campo de atuação da linguagem (RUSSEL, W.; ROBERTS, G., 2009). Helder (ALVES, 2017) destaca outros pontos fortes da linguagem: Independência de plataforma, alto desempenho, segurança e coleta de lixo. A documentação oficial da linguagem nos informa que ela foi lançada oficialmente em sua primeira versão pela Sun Microsystems em 1995 (ORACLE, 2021a).

A instituição Caelum (CAELUM, 2021) explica que o Java é executado em uma máquina virtual (JVM)[5], que acaba sendo uma camada extra entre o sistema operacional e a aplicação. Essa camada tem como propósito traduzir os comandos da aplicação de forma que o sistema operacional, independente de qual está sendo utilizado, o interprete da forma correta e retorne os valores esperados. Dessa forma, não importa se você está usando Windows ou Linux, o seu código fica responsável apenas em comunicar com a JVM. Isso provê a vantagem de escrever o código apenas uma vez e conseguir implantar ele em vários siste-

[5] *Java Virtual Machine*

mas operacionais diferentes. A figura 2.3 ilustra essa comunicação de uma forma melhor.

Figura 2.3 – Comunicação do código Java com a JVM



Fonte: https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java

2.4.1.1 JavaServer Pages (JSP)

JSP é uma linguagem de código aberto criada e mantida pela Sun Microsystems e tem como função a composição de páginas HTML dinâmicas, a partir de objetos e funções contida na linguagem Java (DEVMEDIA, 2012). A documentação oficial da ferramenta (ORACLE, 2010) complementa dizendo que o JSP também pode ser usado para geração de páginas estáticas, que podem ser codificadas em qualquer tipo de arquivo baseado em texto, tais como: HTML, SVG[6], e XML[7].

Como o próprio nome indica, o JSP é uma página que é executada dentro de um servidor Java, pra isso podemos usar os servidores: Wildfly, Tomcat, GlassFish, entre outros.

[6] Scalable Vector Graphics
[7] Extensible Markup Language

2.4.2 JavaScript (JS)

A documentação oficial da linguagem nos introduz informando que com JavaScript podemos criar páginas WEB dinâmicas, com mapas interativos e gráficos 2D/3D animados (ORACLE, 2021b). Já Ugo (ROVEDA, UGO, 2021) completa que "o JS não se restringe mais apenas às páginas e aos navegadores, como foi durante vários anos: com o advento de diversos *frameworks*, APIs, melhorias e criação de centenas de funções, hoje já é possível utilizar JavaScript em aplicativos *mobile*, softwares para desktop e até mesmo em *backend*".

Fazendo uma análise mais profunda das citações, podemos ver que o JavaScript consegue ser uma linguagem poderosa nos tempos atuais, no sentido que consegue abranger diversas áreas de atuação, seja do *frontend*, *backend* até as aplicações móveis. Isso torna a linguagem bem atrativa entre os desenvolvedores e mais atrativa ainda para as empresas, pois consegue economizar gastos que teria para treinar seus funcionários a usarem uma linguagem nova para determinada solução que a empresa eventualmente viessem a usar.

2.4.3 SQL

A linguagem SQL (Structured Query Language) é utilizada para executar comando em bancos de dados relacionais podendo dizer que essa linguagem foi criada para que os desenvolvedores conseguissem que sua aplicação pudesse acessar, consultar e modificar os dados de uma empresa (PRAVALER, 2020).

Um elemento da linguagem SQL é que referenciado neste relatório são as *procedures*. Thiago (THIAGO, 2008) descreve este elemento da seguinte forma "*Stored Procedure*, que traduzido significa Procedimento Armazenado, é uma conjunto de comandos em SQL que podem ser executados de uma só vez, como em uma função. Ele armazena tarefas repetitivas e aceita parâmetros de entrada para que a tarefa seja efetuada de acordo com a necessidade individual".

2.5 Frameworks e bibliotecas

Um *framework* é um conjunto de bibliotecas de código, classes e funções cujo o propósito é auxiliar o desenvolvimento de uma aplicação em uma linguagem de programação (TECHTERMS, 2020b). Durante o estágio foi possível trabalhar com as seguintes bibliotecas e frameworks: Jquery, Symmetric e Treegrid, que serão descritas nos tópicos a seguir.

2.5.1 JQuery

JQuery é uma biblioteca em JavaScript, que auxilia a manipulação de elementos HTML, disparos de eventos em JavaScript e realização de requisições HTTP via Ajax, tornando mais simples o desenvolvimento web (OPENJS FOUNDATIONS, 2021). A biblioteca jQuery pode ser estendida a partir da inclusão do *plugin* jQuery *validator*, que vem com o propósito de tornar mais simples a validação de elementos *inputs* e *forms* do código HTML (JQUERY, 2021).

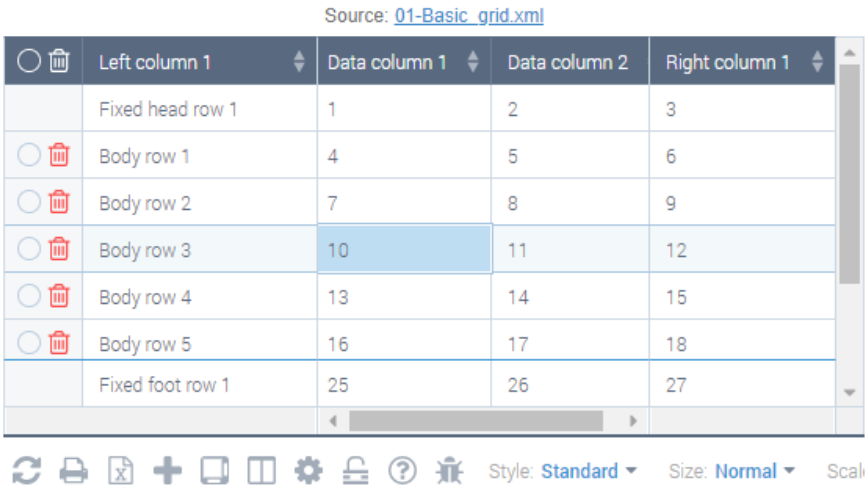
2.5.2 TreeGrid

TreeGrid é uma biblioteca feita com código de JavaScript puro e tem como propósito exibir e editar planilhas (Figura 2.4), tabelas e gráficos em páginas HTML (COQSOFT, 2021).

2.5.3 Symmetric

SymmetricDS (ou Symmetric para encurtar) é um framework de replicação de banco de dados projetado para tornar o banco de dados escalável. Ele suporta diversos tipos de banco de dados e consegue replicar os dados entre sistemas gerenciadores de bancos de dados diferentes, incluindo os mais famosos: Oracle, MySQL e PostgreSQL (JUMPMIND, 2021).

Figura 2.4 – Exemplo de uma planilha da Treegrid



Fonte: http://www.treegrid.com/Grid

3 DESENVOLVIMENTO

Neste capítulo abordarei a metodologia e processo de desenvolvimento adotados, assim como a arquitetura de software utilizada.

O time no qual fiz o estágio se denominava *"WEB/API"*, composto por um desenvolvedor estagiário, dois desenvolvedores, um líder técnico, um *Product Owner* (PO) (2.2.9) e dois analistas de qualidade (testers) (2.2.10). O time era responsável por dois sistemas web, um denominado *Equals Core* o qual contemplava os principais clientes da empresa e a *Retaguarda* o qual tinha como objetivo gerenciar os acessos dos clientes ao *Equals Core* a nível administrativo e atendimento a suporte, sendo possível também emular o acesso a um cliente para prestar algum tipo de auxílio a ele.

O time não tinha contato direto com os clientes, mas atendia as demandas do time de produtos, o qual filtrava algumas solicitações dos clientes e também elaboravam novas ideias a serem implementadas na aplicação.

3.1 Processos de desenvolvimento

Aqui relatarei os ciclos de reuniões que o time possuía e a metodologia ágil adotada pelo time.

Ao longo do período do estágio, os processos foram sempre se adaptando conforme a necessidade da equipe, mas sempre seguiam como base a metodologia ágil *Scrum* (2.2), trazendo consigo as estratégias: Scrum Poker (2.2.4), Daily Scrum (2.2.1), Scrum Board, Sprint (2.2.5), dentre outras estratégias e processos que serão abordados com mais detalhes a seguir.

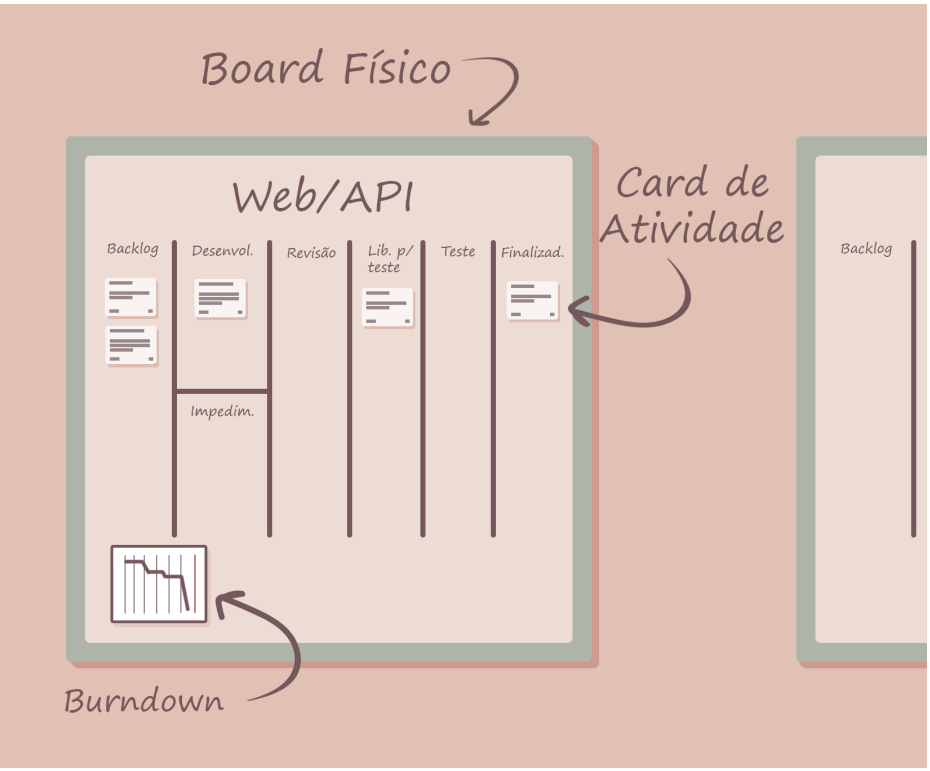
3.1.1 Daily Scrum

Nós realizávamos rigorosamente as *Daily Meetings* sempre com o time todo, salvo algumas raras exceções caso algum membro específico já possuísse alguma outra reunião mais importante no mesmo horário. O horário dessa reunião

era alterado de tempo em tempo, a fim de encontrar um horário fixo na semana em que todos os membros estivessem livres. Frequentemente essa alteração era devido a troca de semestre letivo dos estagiários, a fim de conciliar o horário da *daily* com a grade curricular destes. Quando havia mais de um membro ausente, a *daily* era adiada para o dia seguinte.

A estrutura da reunião segue a seguinte ordem de falas: Primeiramente pelos desenvolvedores, depois os analistas de qualidade, em seguida o Líder técnico e então finalizando com o *Product Owner*. E para acompanhar as atividades usávamos um *board* físico (Figura 3.1) e durante a respectiva vez de fala, cada membro era responsável por atualizar os *cards* das atividades no *board* e também, caso houvesse algum impedimento, o orador atual se manifestava naquele momento.

Figura 3.1 – Ilustração do board físico utilizado nas *Dailys*



Fonte: autoria própria

Também era cronometrada a duração dessa reunião, para fins de apontamentos de horas. A reunião durava de 10 a 15 minutos, salvo algumas exceções, pois haviam muitos impedimentos relatados. Ficávamos cobrando uns aos outros para não nos prolongarmos muito nas discussões dos impedimentos para a duração da reunião não ficar muito extensa. E como forma de garantir a presença pontual de todos nas *dailys*, caso alguém se atrasasse, era cobrado um real de contribuição para o porquinho do time.

Para finalizar a *daily*, nós atualizávamos o gráfico de *burndown da Sprint* (2.2.6) anexado ao *board* físico para que fosse possível ter uma visão mais clara do andamento da *Sprint* em relação às entregas feitas em contraste com sua *deadline*[1].

3.1.2 Scrum Board

O *Scrum Board* era o nosso *board* virtual feito na ferramenta *Jira* (2.3.1). Diferente do *board* físico, o *board* no *Jira* era atualizado em tempo real pelos membros do time.

Nós trabalhávamos com ele usando as seguintes colunas:

- *Backlog* (2.2.7) priorizado - Atividades disponíveis para serem desenvolvidas dentro da *Sprint*;
- Em desenvolvimento - Atividade atribuída a algum desenvolvedor;
- Impedimento - Atividades que faltam alguma definição de um cenário não previsto durante o planejamento ou com algum impedimento externo;
- Aguardando correção - Atividades que contém algum *Bug*[2] e necessitam da atenção do desenvolvedor responsável;

[1] Prazo final da entrega de algum compromisso
[2] Comportamento indesejado do código ou tela

- *Code review* - Atividade grandes que necessitam de uma análise de código feita via *Pull Request*[3] no GitHub (2.3.2);
- Revisão - Revisão principal da atividade, na qual um outro desenvolvedor faz uma análise mais técnica de performance e segurança do código novo escrito;
- Liberado para teste - Atividades que requerem atenção de algum analista de qualidade;
- Em teste - Atividade atribuída a algum analista de qualidade;
- Finalizadas - Atividades que terminaram o fluxo principal de desenvolvimento.

Com exceções de impedimentos que poderiam surgir em qualquer etapa do fluxo de desenvolvimento e atividades simples que não necessitavam de nenhum tipo de revisão, o ciclo de vida ideal de uma atividade está representado na Figura 3.2.

Após o término do fluxo principal da atividade, o time realizava uma validação da mesma no ambiente de homologação, o qual é um ambiente com configurações mais próximas ao de produção e também com o banco de dados limpo, sem alterações manuais.

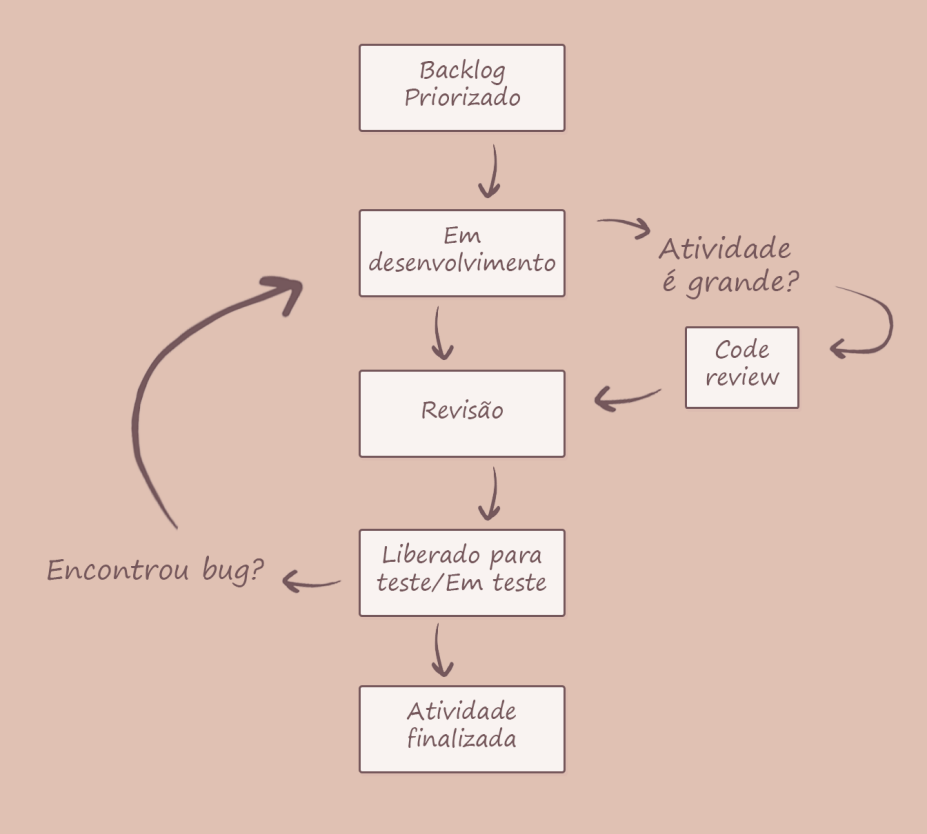
Além do *board* padrão, também tínhamos um outro destinado a chamados de suporte do time de atendimento ao cliente. Mas os estagiários não tinham como responsabilidade em atuar nesse *board*.

3.1.3 Product Owner

Assim como dito antes, o nosso time não atendia as demandas diretamente de um cliente específico, mas sim do time de produtos, que elaboravam e priori-

[3] Processo usado para verificar apenas o código novo descrito na atividade, dentro da plataforma Git

Figura 3.2 – Ciclo de vida ideal de uma atividade



Fonte: autoria própria

zavam as demandas de acordo com as necessidades dos nossos clientes. Quem conectava o nosso time com o time de produtos era o PO. Nele se concentravam todas as regras de negócio acerca das demandas e limitava a quantidade de demandas que o time iria desenvolver durante o período de uma *Sprint*. Diferente do comum, o PO também era responsável pelas obrigações de um *Scrum Master*, pois era ele quem organizava e fazia acontecer as reuniões de *planning* e retrospectiva. E após a finalização da *planning*, ele ficava como responsável por criar os *cards* que seriam anexados no *board* físico usado na *daily* do time e a organização inicial do *board* virtual.

O PO carregava consigo um papel muito importante no dia a dia de desenvolvimento, porquê sempre que surgia alguma dúvida ou impedimento, era ele quem possuía a responsabilidade e conhecimento para sanar o problema no mesmo instante.

3.1.4 Sprint e Retrospectivas

Como parte da metodologia ágil adotada pelo time, utilizávamos os rituais de *Grooming* (2.2.2), *Planning* (2.2.3), *Sprint* e por fim uma retrospectiva (2.2.8).

O primeiro passo do ciclo de reuniões era o *Grooming*, que consistia em uma breve reunião de até uma hora. Durante ele, escolhíamos geralmente um desenvolvedor e um *tester* para analisar de forma superficial possíveis atividades que seriam estimadas durante a *Planning*. O propósito principal do Grooming é de esclarecer as dúvidas do PO para saber se uma atividade estava confusa, faltando informação ou grande demais. Assim, o PO consegue levar as atividades de forma mais bem definida para a *Planning*. Contudo, o *Grooming* não acontecia sempre, pois em alguns casos o próprio PO conseguia subdividir as atividades e defini-las bem, ficando em seu critério a necessidade de realizar ou não a reunião.

Em seguida realizávamos a *Planning* com o time todo. Nela, nós discutíamos as atividades e pontuávamos usando a dinâmica de *Scrum Poker*. Diferente do *Scrum Poker* tradicional, utilizávamos um número reduzido de cartas a fim de evitar muitos valores divergentes entre as pontuações, usando então apenas o seguinte conjunto de cartas: 1, 3, 5, 8, 13. Vale ressaltar que a pontuação a ser estimada pelo *Scrum Poker* contemplava não só o esforço do desenvolvimento, mas o esforço de teste da atividade também.

Cada pontuação representava a seguinte dificuldade:

- 1: Alterações de texto no portal, ou atividades que não gastavam mais de 15 minutos para serem executadas;
- 3: Atividades simples, que gastavam algumas horas para serem resolvidas;

- 5: Atividades médias, que gastavam mais de um dia de desenvolvimento;
- 8: Atividades grandes, que gastavam em torno de uma semana inteira de desenvolvimento;
- 13: Quando uma atividade atingia essa pontuação, tentávamos subdividir ela em outras atividades menores. Mas se isso não fosse possível, elas provavelmente teriam a duração da *Sprint* toda.

Finalizado a *Planning*, começava então a *Sprint*, que durava geralmente 14 dias corridos.

Após a *Sprint*, executávamos uma reunião retrospectiva. Com o propósito de rever todos os aspectos positivos e negativos durante a execução da *Sprint*. O PO sempre trazia algumas dinâmicas diferentes para deixar essa reunião mais intuitiva e lúdica. Mas, geralmente essas reuniões se resumiam em escrever três *post-it*[4] e pregá-los em cada uma das colunas dentre: "Devemos continuar fazendo", "Podíamos ter feito"e "Devemos parar de fazer", representado pela figura 3.3. E após cada um expor sua opinião com o *post-it*, votávamos em qual representava melhor a *Sprint* da equipe e então conversávamos a fim de ver se era necessário tomar alguma ação a respeito do *post-it* eleito. Em caso positivo, era delegado essa ação a algum membro da Equipe com o prazo de conclusão de até a próxima retrospectiva. Por fim, também conversávamos sobre as ações delegadas na retrospectiva passada.

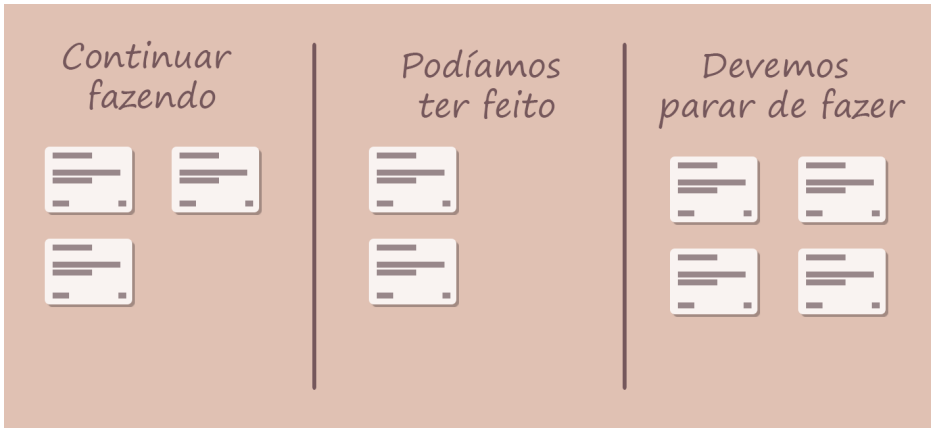
Ao fim da retrospectiva, o ciclo de ritos era reiniciado começando pelo *Grooming* ou *Planning* dependendo da situação.

3.2 Arquitetura da aplicação

A aplicação mantida pela equipe consistia em um *frontend* (2.1) renderizado pelo servidor. Isso nos obrigava a manter tanto o *frontend* quanto *backend* no

[4] Papel adesivo usado para escrever anotações e rascunhos

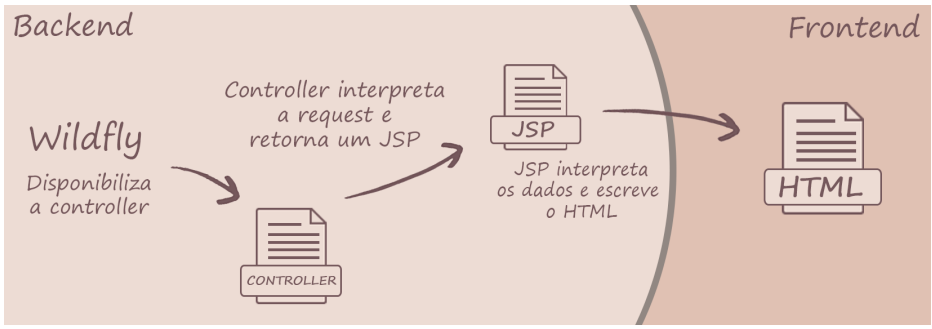
Figura 3.3 – Ilustração *board* da retrospectiva



Fonte: autoria própria

mesmo repositório do Git. Para isso acontecer, usamos o servidor *Wildfly* (2.3.3), que rodava o *backend* em Java (2.4.1) e o *frontend* em *JavaServer Pages* (JSP) (2.4.1.1). Essa comunicação entre as duas camadas era feita a partir de uma *controller* através de um arquivo JSP, a figura 3.4 .

Figura 3.4 – Ilustração conexão *frontend* e *backend*



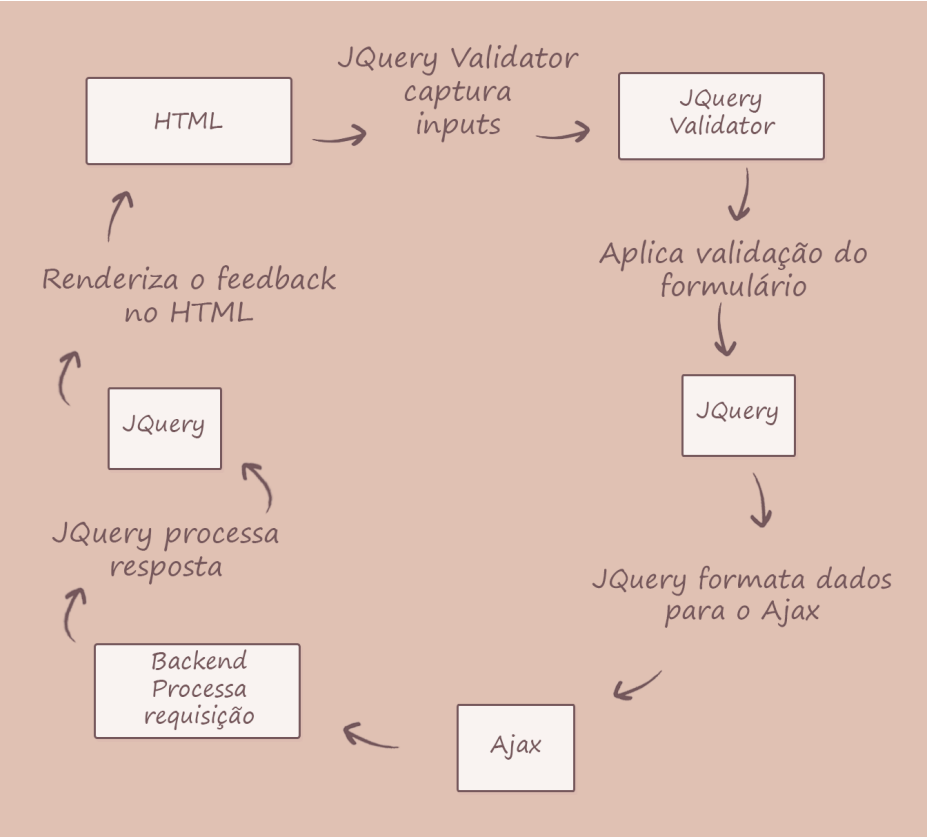
Fonte: autoria própria

3.2.1 Estrutura frontend

Para auxiliar no desenvolvimento do *frontend*, contávamos com o JQuery (2.5.1) para auxiliar na identificação e manipulação dos elementos HTML em lin-

guagem JavaScript (JS) (2.4.2). Além das manipulações padrões do JQuery, estendíamos o seu uso com o JQuery *Validator* para facilitar na validação de formulários, criando regras de *inputs*[5] obrigatórios ou *inputs* que aceitavam apenas alguns valores específicos, bloqueando então o envio do formulário caso alguma regra falhasse. Também estendíamos o JQuery Ajax para realizar as *requests*[6] para o *backend* de forma mais prática. Esse fluxo pode ser visualizado melhor pela figura 3.5.

Figura 3.5 – Fluxo de dados no *frontend*



Fonte: autoria própria

[5] Elemento HTML
[6] Ação em que o navegador aciona algum método do *backend* para receber dados

O *frontend* contava com diversas telas de relatórios, seja de vendas realizadas, vendas canceladas ou fluxo de caixa, todos eles contabilizavam cerca de 20 relatórios diferentes. E para a exibição desses dados no formato de planilha, usávamos a biblioteca TreeGrid (2.5.2) feita em JavaScript. O time contava com um arquivo XML que configurava alguns ícones, cores e estilo que seriam usados por padrão em todos os relatórios que possuíam as planilhas da TreeGrid. Também havia para cada relatório, um arquivo JSON[7] o qual configurava o cabeçalho da planilha, definindo o nome de cada coluna. Por fim, cada relatório possuía uma *string*[8] que armazenava uma rota da API (Application Programming Interface)[9] para buscar os dados de cada linha a ser exibida na planilha, respeitando os filtros aplicados em tela pelo usuário.

3.2.2 Estrutura backend

A estrutura do *backend* consistia em além da camada *controller*, uma camada *service* para concentrar as regras de negócio acerca do projeto, uma camada DAO (*Data Access Object*) para comunicar com o banco de dados. Esta comunicação era feita usando o *driver* de banco de dados MyBatis, ele ainda possuía uma camada de arquivos só dele, na estrutura de marcação XML, para elaboração dos *scripts* em SQL (*Structured Query Language*) (2.4.3). O MyBatis ainda obriga a utilização de *domains*, para fazer a representação das entidades existentes no banco de dados em objetos em Java.

O fluxo de dados consistia em:

- Primeiramente a *controller* recebe uma requisição do *frontend* e faz alguns tratamentos para assegurar que os parâmetros enviados estejam dentro do esperado para aquela rota;

[7] Estrutura de dados usada pra armazenar chaves e valores de um objeto genérico
[8] Estrutura de dados usada pra armazenar textos
[9] Conjunto de rotas que disparam a execução de código no *backend*

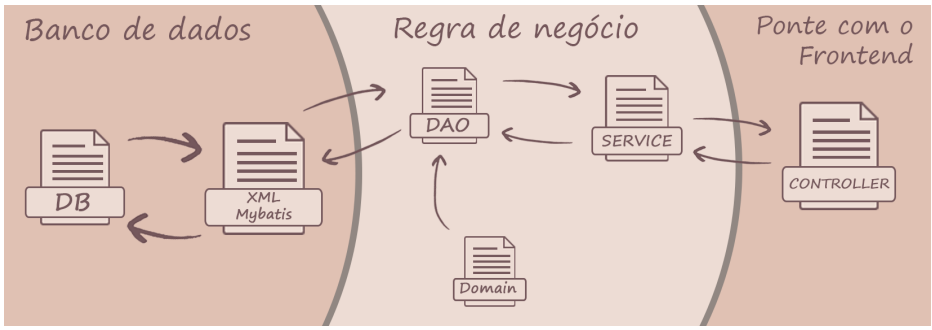
- Em seguida a *service* fica encarregada de fazer algumas validações a respeito de regra de negócio e determinar qual fluxo prosseguir;

- Depois uma *interface DAO* formata os parâmetros recebidos para o formato que o MyBatis aceita e já o executa informando qual *script XML* deverá ser executado. Antes de rodar o *script*, é necessário informar ainda na DAO se o retorno esperado é de um único registro, que será mapeado como um objeto da *domain*, ou se então múltiplos registros serão retornados para serem mapeados uma estrutura de dados do tipo *List* do Java;

- A partir da consulta elaborada em SQL contida no arquivo XML, o MyBatis converte as linhas retornadas do banco de dados em objetos usando como modelo o objeto definido na *domain* e então o agrupa em uma estrutura de dados *List* caso fosse indicado pela DAO. E então o fluxo vai retornado camada por camada, até chegar novamente na *controller*, que irá mapear a *response* de acordo com o esperado no *frontend*.

O fluxo está representado na figura 3.6.

Figura 3.6 – Fluxo de dados no *backend*



Fonte: autoria própria

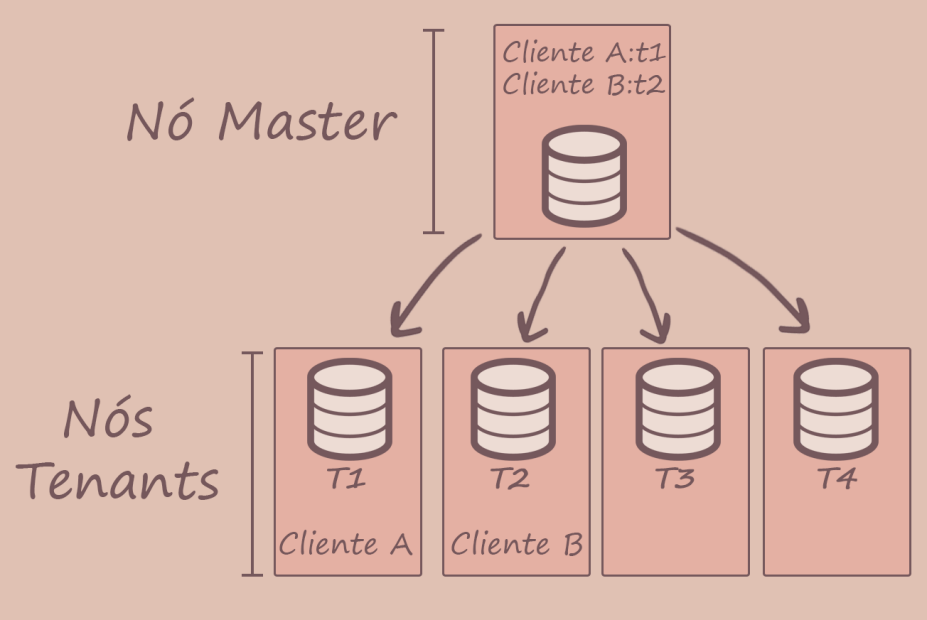
3.2.3 Banco de dados

Na Equals, fazemos o uso de banco de dados remoto, pois grande parte da base de dados é gerada a partir de processamento de arquivos dos clientes. Sendo assim inviável ficar reprocessando os arquivos no computador de cada desenvolvedor. O nosso time contava com 3 bancos de dados, um de desenvolvimento, um de homologação e um de produção. Este primeiro continha acesso livre para escrita e leitura para qualquer usuário que possuísse acesso, já o de homologação e o de produção, apenas a aplicação e alguns usuários específicos podiam escrever alterações na base de dados, os demais usuários possuíam um acesso de leitura com restrições impostas. Isso acontecia para manter o ambiente de homologação o mais próximo possível do de produção, pois no banco de desenvolvimento é possível encontrar vários registros manipulados e corrompidos devido ao próprio desenvolvimento em si. Os dados presentes no banco de dados de teste e homologação possuía origem de clientes que consentiram em oferecer seus arquivos de vendas para serem processados e utilizados com propósito de teste e evolução da plataforma.

Por conter uma quantidade muito grande de transações de nossos clientes utilizávamos o *Symmetric* (2.5.3) para fazer um balanceamento de carga no volume de transações dos clientes, separando eles em *nós tenants* diferentes. Cada *nó tenant* representa um banco de dados separado, então ao fazer uma consulta onerosa de um cliente específico de um *nó*, os outros clientes alocados em outros *nós* não serão afetados. E para orquestrar tudo isso, há um *nó master* que identifica em qual *tenant* cada cliente está alocado e também salva algumas informações da aplicação que não se referem a cliente nenhum. A hierarquia da estrutura de banco de dados da empresa pode ser melhor vista pela figura 3.7.

Algumas poucas tabelas do nosso banco de dados, se encontram tanto no *nó master* quanto no *nó tenant*. Quando precisamos alterar algum dado dessas

Figura 3.7 – Estrutura *Symmetric*



Fonte: autoria própria

tabelas específicas, precisamos alterar apenas no *nó master*, que então o próprio *Symmetric* irá se encarregar de reaplicar a alteração para o *nó tenant*

3.3 Gerenciamento de versões

A ferramenta para gerenciamento de versão de código era o Git, com repositório no GitHub e também havia uma integração com o *Jira* para vincular uma *branch* do Git a um *card* de atividade do *Jira*. Além das *branchs*, os *commits* também eram vinculados ao *Jira*.

O vinculo era feito a partir da utilização do código do *card* com o nome inicial das *branchs* e *commits*. Por exemplo:

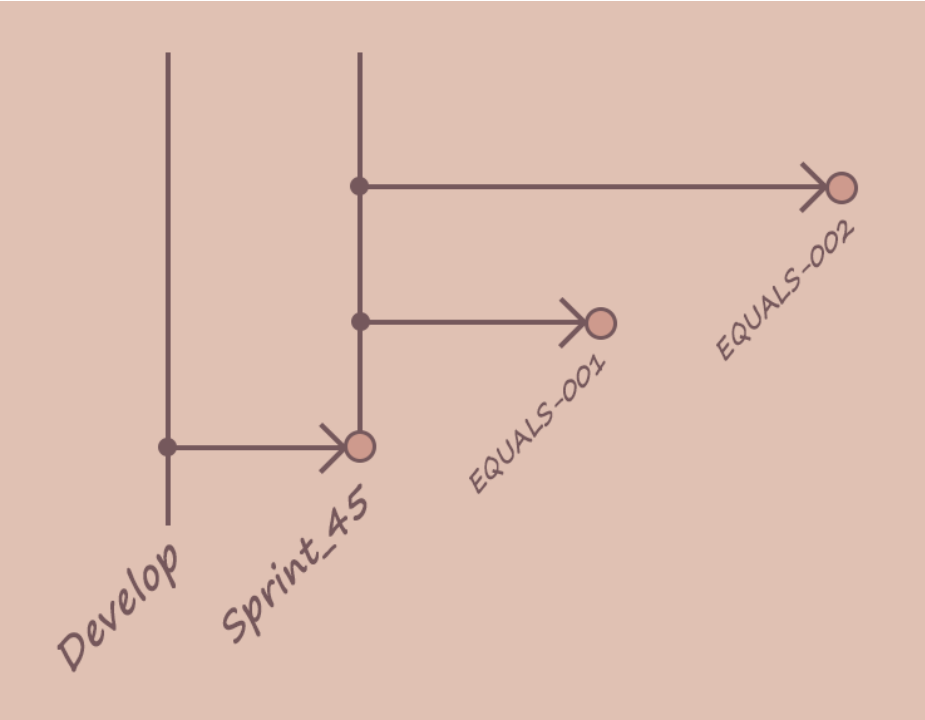
Código do card: "EQUALS-123"

Branch: "EQUALS-123_nome_da_branch"

Commit: "EQUALS-123 - descricao do commit"

O fluxo do Git consistia em sempre que ao iniciar uma *Sprint* criar a *branch* do pacote, que englobaria todas as atividades da *Sprint*, a partir da *develop*. Essa *branch* tinha como nome *"sprint-45"* por exemplo. E ao início de cada uma atividade da *Sprint*, a atividade deve conter uma *branch* criada a partir da *branch* do pacote (Figura 3.8).

Figura 3.8 – Representação da estrutura inicial de um pacote de atividades no Git

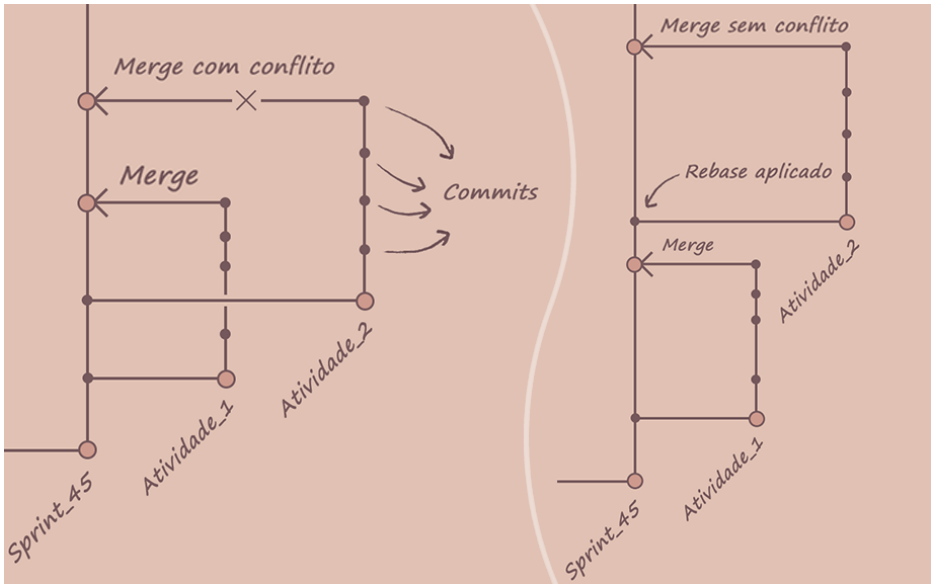


Fonte: autoria própria

Ao finalizar o desenvolvimento e teste de uma atividade, devíamos fazer o *rebase* da sua respectiva *branch* com a *branch* do pacote. O processo de *rebase* consiste em reescrever o histórico de *commits* da *branch* da atividade, para que os *commits* dela fossem aplicados após todos os outros *commits* já existentes na *branch* do pacote. Dessa forma conseguíamos evitar vários conflitos e dos poucos que restavam, eram bem mais simples de serem resolvidos. Após o *rebase* feito,

podíamos então fazer o *merge* da *branch* da atividade na *branch* do pacote (Figura 3.9).

Figura 3.9 – Demonstração da aplicação de um *rebase*



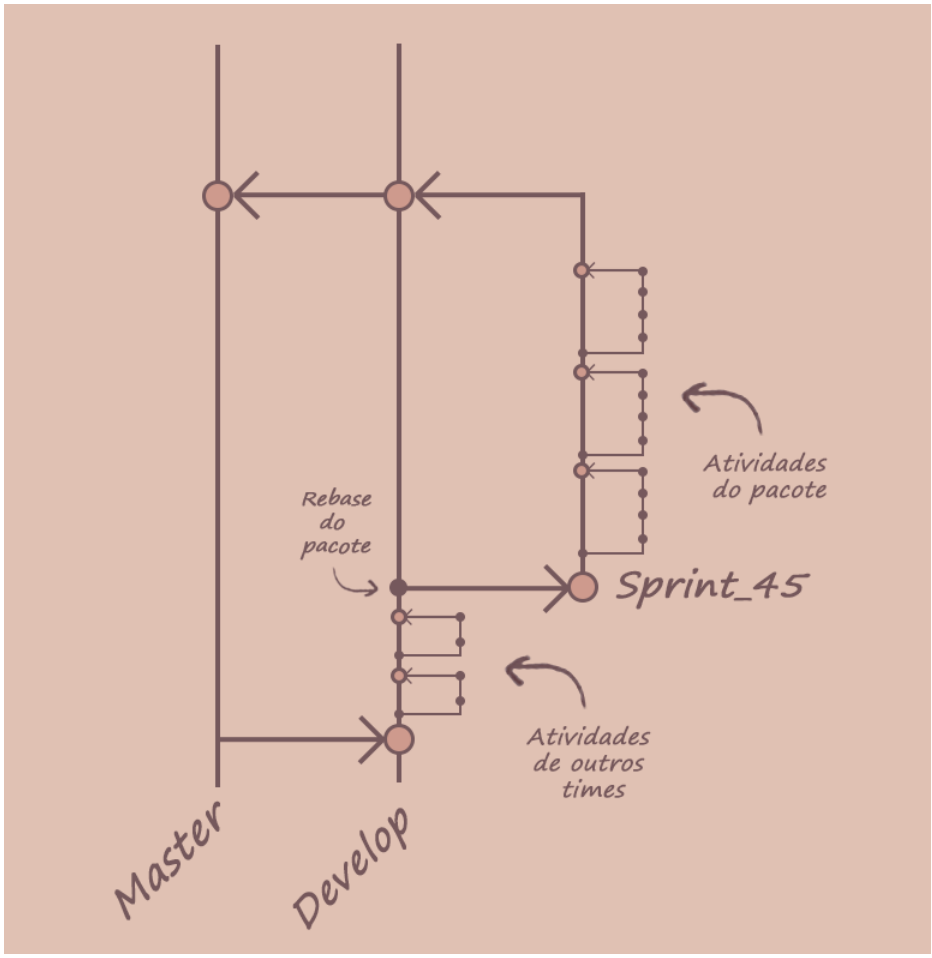
Fonte: autoria própria

Após finalizado o desenvolvimento e teste de todas as atividades, o pacote era dado como concluído e então gerávamos um *build*[10] dele no ambiente de homologação para ser feito uma validação de todas as atividades, revisando se tudo que foi proposto a ser desenvolvido realmente foi desenvolvido sem deixar passar nenhum detalhe. Caso fosse encontrado alguma divergência, era criado um *card* de atividade no jira referente a correção e seguia-se o mesmo fluxo de *branchs* citado anteriormente.

Uma vez que tudo do pacote estivesse pronto e validado, fazíamos um *rebase* do pacote com a *develop* para receber possíveis alterações de outras equipes, depois realizávamos o *merge* do pacote na *develop* e por fim o *merge* na *master* (Figura 3.10).

[10] Versão compilada e otimizada de um código

Figura 3.10 – Demonstração da entrega de um pacote para a *master*



Fonte: autoria própria

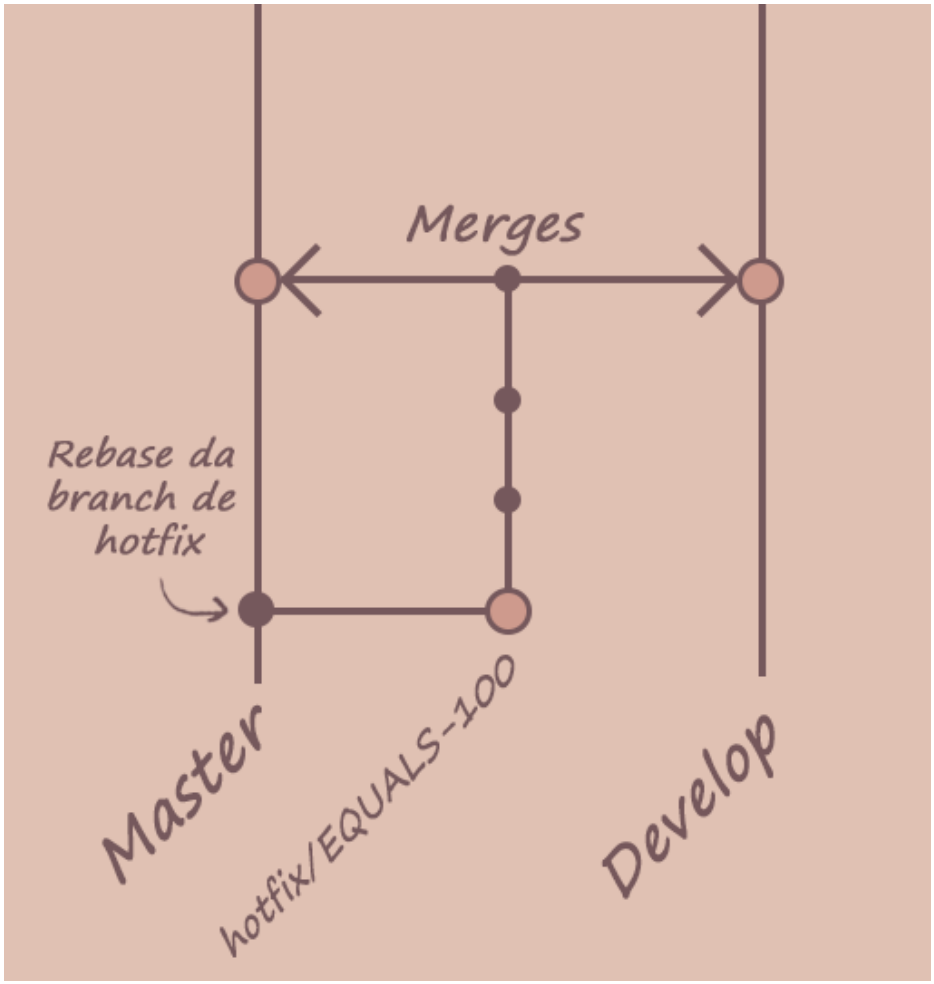
Além desse fluxos de desenvolvimento, também tínhamos o fluxo de *hotfixes*. Os *Hotfixes* são erros encontrados no ambiente de produção, que requerem uma atenção urgente para a resolução do problema. O fluxo desse tipo de correção emergencial consistia em criar uma *branch* com o nome inicial *"hotfix/"*, seguido do código do *card* no *jira*. Exemplo:

Código do card: "EQUALS-100"

Branch: "hotfix/EQUALS-100"

Quando desenvolvido, o *hotfix* seguia pelo mesmo fluxo do *board* padrão, passando por revisão e teste, levando consigo a prioridade alta devido a urgência da correção. E ao finalizar o teste, era feito um *rebase* com a *master*, um *merge* dessa *branch* na *master* e na *develop*. Durante o período de estagiário, pude resolver apenas alguns *hotfixes*, que foram designados para que eu pudesse ter essa experiência, pois esse tipo de atividade era de responsabilidade dos desenvolvedores contratados (Figura 3.11).

Figura 3.11 – Fluxo desenvolvimento de *hotfix*



Fonte: autoria própria

3.4 Ambientes e VPN

O time comportava vários ambientes para assegurar o desenvolvimento correto do projeto. Os desenvolvedores contavam com o ambiente local da sua própria máquina para desenvolver as atividades e faziam o uso do banco de dados compartilhado com os *testers* e outros times da empresa. Já os *testers* possuíam ambiente com um servidor apontando para o banco de dados de desenvolvimento, utilizado para o teste das atividades e outro ambiente apontando para o banco de homologação, utilizado para validação do pacote contendo atividades prontas para liberar para produção. E por fim tínhamos o ambiente de produção que continha banco de dados e servidor dedicados, seu uso era exclusivo dos clientes da empresa.

A empresa possui colaboradores localizados em Lavras e em São Paulo. Para compartilharmos esses servidores e banco de dados, fazíamos o uso do acesso via VPN. Alguns dos bancos de dados e servidores citados continham acesso disponível apenas através da VPN por estarem alocados fisicamente em São Paulo. Enquanto outros bastava acessar a rede local da empresa pra ter acesso.

3.5 Avaliações de desempenho

Para além das rotinas padrões de ciclo de atividades diárias e reuniões semanais, temos também um ciclo maior exigências com um período maior de fechamento, que é o caso das avaliações de desempenho. A empresa conta com o esquema de avaliação 360°, no qual temos que avaliar os pares do nosso time e também o nosso líder. A avaliação de membros de outras equipes era algo opcional, ficando a critério do colaborador se queria incluir alguém a mais ou não.

A empresa também incentivava a cultura de *feedbacks* mais informais, os *1-on-1*, que consistiam em algumas conversas rápidas de cerca de vinte minutos de duração. Elas aconteciam de forma mais comum com um colaborador e seu

líder, mas nada impedia de chamar algum colega para dar um *feedback* positivo ou negativo sobre algum acontecimento.

4 CONCLUSÃO

Este documento relata um estágio realizado na empresa Equals, uma *fintech* voltada para gestão das vendas de seus clientes, no qual o estagiário tinha como responsabilidade atuar em um time que comportava dois sistemas web. Com isso foi capaz de desenvolver e aprimorar as competências de processos ágeis, desenvolvimento de sites, desenvolvimento de banco de dados e trabalho em equipe.

Pude desenvolver diversas atividades ao longo estágio e com base em relatórios criados pelo PO, é possível ver que participei de um total de 8 ciclos de Sprint. O numero total de atividades desenvolvidas por mim foi de 30 e o valor total dessas atividades representadas em pontos jogados pelo Scrum Poker foi de 106. Passaram-se por mim 29 atividades que precisei revisar de outros desenvolvedores.

Durante esses 5 meses de estágio, foi possível colocar em prática e reforçar os conceitos teóricos de diversas disciplinas ofertadas pelo curso de Ciência da Computação da UFLA. Dentre as disciplinas obrigatórias posso destacar suas contribuições para o estágio da seguinte forma:

- Algoritmos e estrutura de dados - Proporcionando conhecimento acerca das várias estrutura de dados que existem, no qual muitas delas foram utilizadas durante o estágio. Melhorando também habilidades de lógica de programação, característica principal para definir um bom programador.

- Banco de Dados - Disciplina crucial para o estágio, visto que estava constantemente lidando com banco de dados. Na Equals pude aprofundar ainda mais na área por conta da arquitetura robusta do banco.

- Programação Orientada a Objetos - Ensinou o paradigma de programação da principal linguagem usada no estágio. Também descreveu diversos padrões de projetos em que alguns deles eram utilizados no projeto do time.

- Redes de Computadores - Contemplando conhecimentos sobre protocolos de internet HTTP/HTTPS, TCP/UDP e uso de VPN[1].

- Sistemas Distribuídos - Aprendizados sobre utilização de *Web Sockets*.

E para as disciplinas eletivas que escolhi cursar ao longo da minha graduação, posso citar os seguintes aprendizados obtidos:

- Processos de Software - Nesta disciplina foi apresentado várias metodologias que compõe o desenvolvimento de software. A Equals usava na prática alguns conceitos abordados nesta disciplina.

- Modelagem e Implementação de Software - Disciplina muito semelhante a Processos de Software, mas com foco mais técnico e prático, reforçando ainda mais as habilidades de desenvolvimento de softwares complexos e o trabalho em equipe.

- Programação WEB - Conteúdo desta disciplina foi totalmente proveitoso para o estágio, visto que tinham o mesmo propósito de trabalhar com aplicações WEB.

Dessas citadas, em banco de dados pude me aprofundar muito mais conhecendo uma nova arquitetura de banco de dados, que seria o balanceamento de carga usando o *framework Symmetric*, além de lidar constantemente com *procedures* e *scripts* em SQL, proporcionando um domínio maior ainda acerca dessa área. Em desenvolvimento Web também pude ir muito além, pelo fato da Equals possuir um sistema web complexo, com diversas telas e funcionalidades diferentes, ampliando ainda mais os conhecimentos adquiridos.

A empresa Equals pôde proporcionar um estágio de grande valia, pois é uma empresa muito bem estabelecida no mercado e com processos de desenvolvimentos bem maduros. O time em que realizei o estágio possuía profissionais

[1] *Virtual Private Network* (Rede Privada Virtual)

excelentes que fizeram total diferença no meu aprendizado e crescimento. Como iniciei o estágio na metade da minha graduação, ao longo deste período pude ir correlacionando os aprendizados práticos das disciplinas de forma que um complementasse o outro, trazendo para sala de aula alguns conceitos que vivenciei, agregando mais conteúdo a certas matérias da graduação de Ciências da Computação.

REFERÊNCIAS

ALVES, H. C. Java: Introdução, características, estrutura e ambientes de programação. **Revista conexão eletrônica**, 2017. Disponível em: <http://revistaconexao.aems.edu.br/wp-content/plugins/download-attachments/includes/download.php?id=1533>.

ANDREI L. **O Que é GitHub e Para Que é Usado?** 2021. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-github>. Acesso em: 12 mai. 2021.

ATLASSIAN. **The best software teams ship early and often**. 2021. Disponível em: <https://www.atlassian.com/software/jira>. Acesso em: 8 mai. 2021.

BERNARDO, KLEBER. **Backlog Grooming: Refinando o Backlog**. 2015. Disponível em: <https://www.culturaagil.com.br/backlog-grooming-refinando-o-backlog/>. Acesso em: 11 mai. 2021.

CAELUM. **O que é Java**. 2021. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java#maquina-virtual>. Acesso em: 12 mai. 2021.

CAMARGO, ROBSON. **Backlog grooming: as melhores práticas para refinar o seu produto**. 2019. Disponível em: <https://robsoncamargo.com.br/blog/Conheca-as-melhores-praticas-de-uma-backlog-grooming>. Acesso em: 11 mai. 2021.

CAMARGO, ROBSON. **Sprint planning: dicas para reuniões mais dinâmicas e produtivas**. 2019. Disponível em: <https://robsoncamargo.com.br/blog/Sprint-planning>. Acesso em: 11 mai. 2021.

CAMARGO, ROBSON. **Burndown: conheça o gráfico que mede produtividade**. 2020. Disponível em: <https://robsoncamargo.com.br/blog/Burndown>. Acesso em: 11 mai. 2021.

CARVALHO, TÉCIA. **Planning Poker: o que é e como funciona com o Scrum**. 2019. Disponível em: <https://www.voitto.com.br/blog/artigo/planning-poker>. Acesso em: 11 mai. 2021.

COQSOFT. **Editable JavaScript TreeGrid**. 2021. Disponível em: <http://www.treegrid.com/Grid>. Acesso em: 8 mai. 2021.

COUTINHO, THIAGO. **Aprenda o que é Daily Scrum e qual seu papel na gestão ágil com Scrum**. 2019. Disponível em: <https://www.voitto.com.br/blog/artigo/daily-scrum>. Acesso em: 11 mai. 2021.

COUTINHO, THIAGO. **Product Backlog: o que é isso?** 2019. Disponível em: <https://www.voitto.com.br/blog/artigo/product-backlog>. Acesso em: 11 mai. 2021.

DEVMEDIA. **Introdução ao Java Server Pages - JSP**. 2012. Disponível em: <https://www.devmedia.com.br/introducao-ao-java-server-pages-jsp/25602>. Acesso em: 12 mai. 2021.

DUARTE, LUIZ. **O que é Burndown Chart e como usar?** 2021. Disponível em: <https://www.luiztools.com.br/post/o-que-e-burndown-chart-e-como-usar/>. Acesso em: 11 mai. 2021.

EQUALS. **Equals - Gestão Financeira**. 2021. Disponível em: <https://www.equals.com.br/>. Acesso em: 3 mai. 2021.

FRONTENDMASTERS. **Front-end Developer Handbook**. 2019. Disponível em: <https://frontendmasters.com/books/front-end-handbook/2019/#2>. Acesso em: 8 mai. 2021.

GITHUB. **O Que é GitHub e Para Que é Usado?** 2021. Disponível em: <https://github.com/>. Acesso em: 12 mai. 2021.

JQUERY. **JQuery Validation Plugin**. 2021. Disponível em: <https://jqueryvalidation.org/>. Acesso em: 8 mai. 2021.

JUMPMIND. **About SymmetricDS**. 2021. Disponível em: <https://www.symmetricds.org/about/overview>. Acesso em: 8 mai. 2021.

JUNIOR, CARLOS. **Scrum: o que é sprint e como executá-lo?** 2017. Disponível em: <https://www.projectbuilder.com.br/blog/scrum-o-que-e-sprint-e-como-executa-lo/>. Acesso em: 11 mai. 2021.

KALENDAE. **Quality Assurance: entenda o que é e como aplicar na gestão de TI**. 2019. Disponível em: <https://www.kalendae.com.br/blog/quality-assurance/>. Acesso em: 12 mai. 2021.

OPENJS FOUNDATIONS. **What is jQuery**. 2021. Disponível em: <https://jquery.com/>. Acesso em: 8 mai. 2021.

ORACLE. **What Is a JSP Page?** 2010. Disponível em: <https://docs.oracle.com/javaee/5/tutorial/doc/bnagy.html>. Acesso em: 12 mai. 2021.

ORACLE. **O que é a Tecnologia Java e porque preciso dela?** 2021. Disponível em: <https://www.java.com/pt-BR/download/help/whatis_java.html>. Acesso em: 12 mai. 2021.

ORACLE. **O que é JavaScript?** 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript>. Acesso em: 13 mai. 2021.

PRAVALER. **SQL – o que é e como funciona na prática?** 2020. Disponível em: <https://www.pravaler.com.br/sql-o-que-e-e-como-funciona-na-pratica/>. Acesso em: 13 mai. 2021.

RED HAT. **About the WildFly Project**. 2021. Disponível em: <https://www.wildfly.org/about/>. Acesso em: 8 mai. 2021.

ROVEDA, UGO. **JavaScript: o que é, para que serve e como funciona o JS?** 2021. Disponível em: <https://kenzie.com.br/blog/javascript/>. Acesso em: 13 mai. 2021.

RUSSEL, W.; ROBERTS, G. **Desenvolvendo Software em Java**. 2009. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/978-85-216-1994-9/>. Acesso em: 1 jun. 2021.

SCRUM ORG. **What is a Sprint Retrospective?** 2021. Disponível em: <https://www.scrum.org/resources/what-is-a-sprint-retrospective>. Acesso em: 12 mai. 2021.

SOFTWARE FREEDOM CONSERVANCY. **Git**. 2021. Disponível em: <https://git-scm.com/>. Acesso em: 12 mai. 2021.

TAMEIRÃO, NATHÁLIA. **O manifesto ágil**. 2021. Disponível em: <https://sambatech.com/blog/insights/metodos-ageis/>. Acesso em: 11 mai. 2021.

TECHTERMS. **Backend Definition**. 2020. Disponível em: <https://techterms.com/definition/backend>. Acesso em: 8 mai. 2021.

TECHTERMS. **Framework Definition**. 2020. Disponível em: <https://techterms.com/definition/framework>. Acesso em: 8 mai. 2021.

THIAGO. **Introdução aos Stored Procedures no SQL Server**. 2008. Disponível em: <https://www.devmedia.com.br/introducao-aos-stored-procedures-no-sql-server/7904>. Acesso em: 13 mai. 2021.

VIEIRA, RAFAEL. **O que é um Product Owner?** 2017. Disponível em: <https://medium.com/produto-di%C3%A1rio/o-que-%C3%A9-um-product-owner-c44bb29a9f66>. Acesso em: 12 mai. 2021.

VILAVERDE, LUDMILA. **JOB DESCRIPTION: ANALISTA DE QUALITY ASSURANCE**. 2019. Disponível em: <https://tutano.trampos.co/18598-job-description-quality-assurance/>. Acesso em: 12 mai. 2021.

VITOR L. MASSARI. **Agile Scrum Master no Gerenciamento Avançado de Projetos**. 2016. Disponível em: <www.brvirtual.com.br>. Acesso em: 1 jun. 2021.

W3SCHOOLS. **What is Full Stack**. 2021. Disponível em: <https://www.w3schools.com/whatis/whatis_fullstack.asp>. Acesso em: 8 mai. 2021.