



CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS

Recredenciado pela Portaria Ministerial nº 1.162, de 13/10/16, D.O.U. nº 198, de 14/10/2016

AELBRA EDUCAÇÃO SUPERIOR - GRADUAÇÃO E PÓS-GRADUAÇÃO S.A.

Matheus Silva Morais

DEVMASTER: Ambiente gamificado para desenvolvedores

Palmas – TO

2018

Matheus Silva Morais

DEVMASTER: Ambiente gamificado para desenvolvedores

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso II (TCC II) do curso de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.e Jackson Gomes de Souza.

Palmas – TO

2018

Matheus Silva Moraes

DEVMASTER: Ambiente gamificado para desenvolvedores

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso I (TCC I) do curso de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.e Jackson Gomes de Souza.

Aprovado em: ____/____/____

BANCA EXAMINADORA

Prof. M.e Jackson Gomes de Souza

Orientador

Centro Universitário Luterano de Palmas – CEULP

Abreviação de Professor. Abreviação da Titulação. Nome do Avaliador

Centro Universitário Luterano de Palmas – CEULP

Abreviação de Professor. Abreviação da Titulação. Nome do Avaliador

Centro Universitário Luterano de Palmas – CEULP

Palmas – TO

2018

RESUMO

MORAIS, Matheus Silva. **DEVMASTER: ambiente gamificado para desenvolvedores**. 2018. 37 f. Trabalho de Conclusão de Curso (Graduação) - Curso de Ciência da Computação, Centro Universitário Luterano de Palmas, Palmas/TO, 2018.

O presente trabalho aborda a gamificação do desenvolvimento de *software* por meio da criação do DevMaster, que por sua vez, através da utilização de elementos de jogos, gamifica o ambiente de desenvolvimento de *software*, de uma equipe que utilize técnicas de versionamento de código e, adote o Scrum como metodologia de desenvolvimento. Nesse contexto a Fábrica de Software do CEULP/ULBRA foi escolhida, pois além de utilizar o Scrum também utiliza o Gitlab como repositório de código-fonte. Diante disso foi realizado pesquisas teorias a respeito do Scrum, da gamificação e da mineração de repositório de *software*. Onde na gamificação além de analisar os elementos de jogos, propostos por Werbach e Hunter, para a utilização dos que podem ser mais relevantes para o problema abordado, também foi utilizado o *framework* D6, também proposto por Werbach e Hunter, que dispõe de seis etapas para a criação de uma boa gamificação, onde no presente trabalho foram aplicadas.

Palavras-chave: Gamificação. Scrum. Mineração de repositório de software. *Framework* D6. Elementos de jogos.

SUMÁRIO

RESUMO.....	2
1INTRODUÇÃO.....	5
2REFERENCIAL TEÓRICO.....	7
2.1 SCRUM.....	7
2.1.1. Estrutura	7
2.1.2. Papéis.....	8
2.1.3. Cerimonias.....	9
2.1.4. Artefatos	11
2.2 GAMIFICAÇÃO.....	13
2.2.1. Jogos	13
2.2.2. Elementos de Gamificação	25
2.3 MINERAÇÃO DE REPOSITÓRIO DE SOFTWARE.....	33
2.3.1. Ferramentas de mineração de repositório de software	34
3MATERIAIS E MÉTODOS.....	37
3.1. MATERIAIS	37
3.1.1. Octalysis	37
3.1.2. Gitlab	37
3.1.3. Angular	38
3.1.4. Plataforma de programação	38
3.2. MÉTODOS.....	38
3.2.1. Etapa de compreensão do contexto	39
3.2.2. Etapa de projeto	40
3.3.1. Etapa de implementação	41
4RESULTADO E DISCUSSÃO	42
4.1. COMPREENSÃO DO CONTEXTO	42
4.1.1. Questionário	42
4.1.2. Pontuação das respostas do questionário.....	43
4.1.3. Octacore.....	44
4.2. GAME DESIGN.....	45
4.2.1. Objetivo de Negócios	45
4.2.2. Delimitar Comportamento Alvo	47
4.2.3. Descrever Jogador	47
4.2.4. Ciclo de Atividades	48

4.2.5. Garantir Diversão	49
4.2.6. Implementar Ferramentas Apropriadas	50
4.3. PLATAFORMA DESENVOLVIDA	59
4.3.1. Backend	59
4.3.2. Frontend.....	85
5 CONSIDERAÇÕES FINAIS.....	107
5.1. TRABALHOS FUTUROS	108
REFERÊNCIAS	110

1 INTRODUÇÃO

O processo de desenvolvimento de software é marcado principalmente com a escolha de uma metodologia de desenvolvimento. Segundo Pereira (2017) essa é uma etapa de suma importância, pois pode impactar diretamente no sucesso da equipe de desenvolvimento. Uma falha na escolha da metodologia ideal pode ocasionar em diversos erros, como: um produto com qualidade inferior, tempo para desenvolvimento demasiadamente grande, conflitos dentro da equipe, entre outros problemas.

Visando a procura de uma metodologia de desenvolvimento adequada, pode ser observado o Scrum que é uma metodologia que traz consigo aspectos que tornam o time de desenvolvimento mais produtivo e efetivo. A divisão de papéis é um desses aspectos, no Scrum ela é bem definida e isso faz com que o time Scrum tenha uma ideia bastante detalhada do que deve ser feito.

Após a escolha da metodologia de desenvolvimento de software, devem ser adotadas práticas que impulsionem o time de desenvolvimento a obter um produto de qualidade. Uma dessas práticas é o uso de versionamento de código, que além de deixar o desenvolvimento mais otimizado também pode ser utilizado para medir o desempenho do desenvolvedor ou da equipe de desenvolvimento.

O versionamento de código tem como produto um repositório de software que segundo Silva (2013) são nesses repositórios que são mantidas todas as informações do desenvolvimento realizado pela equipe em questão. Onde, através da mineração de repositório de código é possível a retirada de informações (como *commits*, *issues* e *milestones*) a respeito do desenvolvimento.

Partindo para um contexto de jogos, segundo Tameirão (2016) os jogos atraem e motivam as pessoas, e, que semanalmente são gastados 3 bilhões de horas com jogos, isso apenas por diversão ou obstinação em alcançar determinado objetivo em um jogo. Buscando trazer essa obstinação dos jogos para esse âmbito de desenvolvimento de software, surge a ideia da gamificação, que segundo Kuutti (2013) e Deterding et al. (2011) é o ato de buscar elementos de jogos e trazer para um contexto não-jogo.

Voltando para a questão da metodologia, Riker (2016) acredita que o Scrum tem características (como o time *scrum* e a *sprint retrospective*) que se assemelham a jogos,

por consequência a gamificação, e o que deveria ser mais trabalhando nele, no contexto de gamificação, seria a diversão.

O contexto do presente trabalho é levar em consideração uma equipe de desenvolvimento de *software* que não só utiliza o *scrum* como metodologia de desenvolvimento, mas que também conta com práticas de versionamento de código através dos repositórios de *software*.

Trazendo a mineração de repositório de *software* para o contexto, com as informações (como *commits*, *issues* e *milestones*) retiradas dos repositórios de *software* é possível um acompanhamento do que é desenvolvido pela equipe em questão, e com isso o desenvolvimento da gamificação.

A ideia da gamificação no presente trabalho é trazer os benefícios de jogos para um ambiente de desenvolvimento de software. Nesse sentido, o ambiente de desenvolvimento de software considera o contexto da Fábrica de Software do CEULP/ULBRA, que utiliza metodologias ágeis no desenvolvimento de software (*Scrum*) e ferramentas para gerenciamento dos artefatos de software e versionamento de código, como o Gitlab.

Em vista disso, o capítulo 2 aborda toda a revisão teórica, dividida em *scrum*, gamificação e mineração de repositório de *software*. Já o capítulo 3 aborda os materiais e os métodos usados no presente trabalho. O capítulo 4 aborda os resultados, tal como o *game design* e a plataforma desenvolvida. E por último, o capítulo 5 aborda a conclusão.

2 REFERENCIAL TEÓRICO

2.1 SCRUM

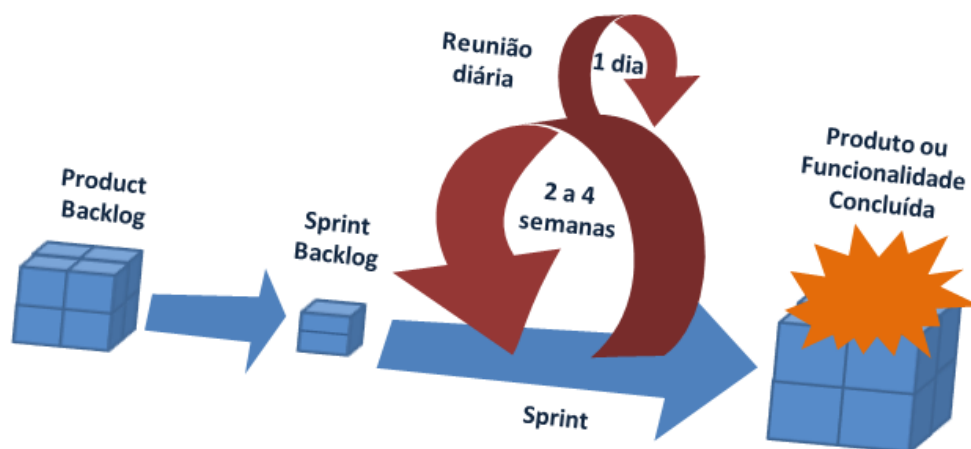
O Scrum é um modelo ágil de processo, foi criado inicialmente por Jeff Sutherland e sua equipe em 1990, a princípio o *Scrum* foi formalizado para projetos de desenvolvimento de *software*, mas se adapta muito bem para qualquer tipo de projeto inovador (NUNES, 2017). Schwaber e Sutherland (2018) abordam o *Scrum* como um framework onde pode-se empregar vários processos ou técnicas que auxiliam no desenvolvimento de produtos complexos.

Com isso é possível afirmar que o *Scrum* é um processo de desenvolvimento ágil, que não é necessariamente implementado apenas para o desenvolvimento de *software*, mas, pode ser utilizado para qualquer projeto inovador que apresenta um ambiente que se adeque às características do *Scrum*. Os tópicos seguintes explicam o funcionamento do Scrum, divididos em: Estrutura, papéis, cerimônias e artefatos.

2.1.1. Estrutura

A Figura a seguir mostra como é estruturado o *Scrum*, e posteriormente uma explicação sobre cada item, em sua estrutura.

Figura 1 - Estrutura do Scrum.



Fonte: Vieira (2014)

O *Product Backlog* é o conjunto de ideias que devem ser desenvolvidas no decorrer de todo o processo, ideias que são concebidas a partir do *Product owner*, que é o dono do produto (VIEIRA, 2014).

No *Scrum* é realizado iterações para o desenvolvimento do trabalho, essas iterações são chamadas de *Sprints*, podendo chegar a durar de duas a quatro semanas, a *Sprint Backlog* é uma parte das funcionalidades descritas no *Product Backlog*, onde cada *Sprint Backlog* deve ser desenvolvida no decorrer da *Sprint* (VIEIRA, 2014).

As reuniões diárias ou, *Daily Scrum*, são reuniões que matem o grupo unido, pois gera uma interação entre todo o time *Scrum*, reuniões essas que devem ocorrer todos os dias, no mesmo horário e no mesmo local (LOPES, 2014).

O processo de desenvolvimento no *Scrum*, é realizado seguindo todos os passos estruturados na Figura 1, para um entendimento mais abrangente do *Scrum* os tópicos seguintes apontaram de forma detalhada o que faz cada item.

2.1.2. Papéis

Segundo Schwaber e Sutherland (2018) os papéis no *Scrum* são: *Product owner*, Time de desenvolvimento e *Scrum master*.

Para Schwaber e Sutherland (2018) o ***Product owner*** (dono do produto) é responsável por maximizar o valor produto e do trabalho do time de desenvolvimento, além disso, também é responsável por gerenciar o ***product backlog*** realizando, dentre outras, as atividades:

- Expressar claramente os itens do ***product backlog***;
- Ordenar os itens do ***product backlog*** para alcançar melhor as metas e missões;
- Garantir o valor do trabalho realizado pelo **time de desenvolvimento**;
- Garantir que o ***product backlog*** seja visível, transparente, claro para todos, e mostrar o que o **time Scrum** vai trabalhar a seguir; e,
- Garantir que o **Time de Desenvolvimento** entenda os itens do ***product backlog*** no nível necessário.

Sendo o **product owner** o detentor do **product backlog**, apenas ele pode fazer alterações na mesma, assim para alguma alteração na prioridade dos itens, o **product owner** deve ser convencido.

Segundo Lopes (2014) o time de desenvolvimento é a equipe que trabalha para entregar uma versão usável do **product backlog**, ao final de cada **sprint**, o **time de desenvolvimento** é estruturado para organizar e gerenciar seu próprio trabalho, de modo que nem mesmo o **Scrum master** pode apontar como deve ser feito o trabalho.

Uma **equipe de desenvolvimento** deve ser pequena, para continuar ágil, e, necessariamente grande para poder completar as tarefas que são propostas e ainda cabe a um time de desenvolvimento possuir as seguintes características (SCHWABER E SUTHERLAND, 2018):

- **Times de desenvolvimento** são multifuncionais, possuindo todas as habilidades necessárias, enquanto equipe, para criar o incremento do produto.
- O **Scrum** não reconhece títulos para os integrantes do time de desenvolvimento que não seja o desenvolvedor, independentemente do trabalho que está sendo realizado pela pessoa;
- Individualmente os integrantes do **time de desenvolvimento** podem ter habilidades especializadas e área de especialização, mas a responsabilidade pertence ao time de desenvolvimento como um todo; e,
- Times de desenvolvimento não contém sub-times dedicados a domínios específicos de conhecimento, tais como teste ou análise de negócios.

Vieira (2014) aponta o **Scrum Master** como o “ responsável por ajudar a todos os envolvidos a entender e abraçar os valores, princípios e práticas do Scrum. Ela age como um *Coach*, executando a liderança do processo e ajudando a equipe Scrum (e o resto da organização) a desenvolver sua própria abordagem do Scrum, que tenha a melhor performance, respeitando as particularidades da organização”.

2.1.3. Cerimônias

Nunes (2017) afirma que no *Scrum* tem “quatro reuniões, ou cerimônias, como são chamadas na literatura.”. Cerimônias essas descritas por Schwaber e Sutherland (2018), como sendo: **Sprint planning**, **daily scrum**, **sprint review** e **sprint retrospective**.

A etapa da **sprint planning** (planejamento da *sprint*) como foi citado na seção 1.1., são as funcionalidades descritas no **product backlog**, que é um artefato que será descrito na seção 1. 4.. Sendo realizado pelo *scrum master* e o time de desenvolvimento, o **sprint planning** tem o objetivo de definir os subconjuntos mais importantes do *product backlog*, para ser desenvolvido na próxima *sprint* (VIEIRA, 2014).

Ainda na etapa de planejamento Schwaber e Sutherland (2018) definem que uma *sprint* deve ter no máximo, um mês de duração, sendo que o *scrum master* garanta que esse planejamento ocorra da maneira correta e seus participantes entendam os seus propósitos, além disso, ainda apontam duas perguntas fundamentais nessa etapa, sendo elas:

- O que pode ser entregue como resultado do incremento da próxima *sprint*?
- Como o trabalho necessário para entregar o incremento será realizado?

Já a **daily scrum**, deve ser feita todo dia, preferencialmente no mesmo horário e local das anteriores (KNIBERG, 2007 apud NUNES, 2017). Esta reunião é considerada uma das práticas mais importantes do *scrum*, com ela é feito a interação diária de todos os elementos do time *scrum*, que leva o time de desenvolvimento comunicar com o *scrum master* as dificuldades encontradas durante o dia anterior, e, uma forma prática de resolver-las (LOPES, 2014). Schwaber e Sutherland (2018) apontam que dentro da reunião diária é esclarecido 3 perguntas, sendo elas:

- O que eu fiz ontem que ajudou o time de desenvolvimento a atender a meta da *sprint*?
- O que eu farei hoje para ajudar o time de desenvolvimento atender a meta da *sprint*?
- Eu vejo algum obstáculo que impeça a mim ou o time de desenvolvimento no atendimento da meta da *sprint*?

Ao final da reunião é feito uma **sprint burndown** para verificar o progresso da *sprint*, além de ver o progresso da *sprint*, essa reunião também é usada para evitar a duplicação de trabalho, sendo uma reunião chave para inspeção e adaptação (LOPES, 2014).

A **sprint review** segundo Nunes (2017) é realizada “depois de concluído o *sprint*, onde o time apresentará para o **product owner** o incremento de produto gerado nesta

iteração. Apenas tarefas 100% concluídas devem ser apresentadas. ” Vieira (2014) consente que a *sprint review* destina-se a motivar, obter comentários e promover a colaboração.

Lopes (2014) aponta que a *sprint retrospective* é uma reunião que ocorre após a *sprint review* e conta com a participação apenas do *scrum master* e o time de desenvolvimento, e, seu objetivo é, além de discutir o que deu errado durante a *sprint*, também tem a função de reforçar a visão do produto de forma a fazer com que os elementos dentro do time *scrum* interajam e analisem como podem ajudar os outros membros em prol do projeto, deste modo, tornando a equipe mais auto-organizável.

Schwaber e Sutherland (2018) definem o propósito da *sprint retrospective*, com os seguintes pontos:

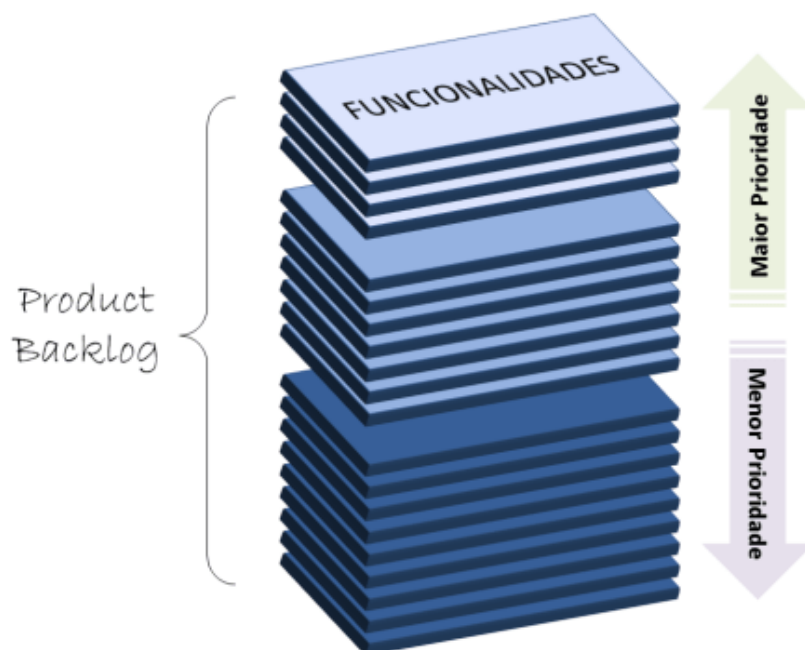
- Inspecionar como a última *sprint* foi em relação às pessoas, aos relacionamentos, aos processos e às ferramentas;
- Identificar e ordenar os principais itens que foram bem e as potenciais melhorias; e,
- Criar um plano para implementar melhorias no modo que o time *scrum* faz seu trabalho;

2.1.4. Artefatos

O *scrum* tem artefatos definidos especialmente para maximizar a transparência das informações, dessa forma, fazendo com que todos tenham o mesmo entendimento dos artefatos, artefatos que estão divididos em: ***Product backlog***, ***sprint backlog*** e ***increment*** (SCHWABER e SUTHERLAND, 2018).

Vieira (2014) defende que o ***product backlog*** é uma lista de determinadas funcionalidades que ao final do processo de desenvolvimento, deve estar totalmente pronta, esta lista é criada em conjunto com o ***time scrum*** e, deve ser feita de forma a respeitar as funcionalidades de maior prioridade dentro do ***product backlog***, como mostra a imagem a seguir.

Figura 2 - Estrutura do *product backlog*.



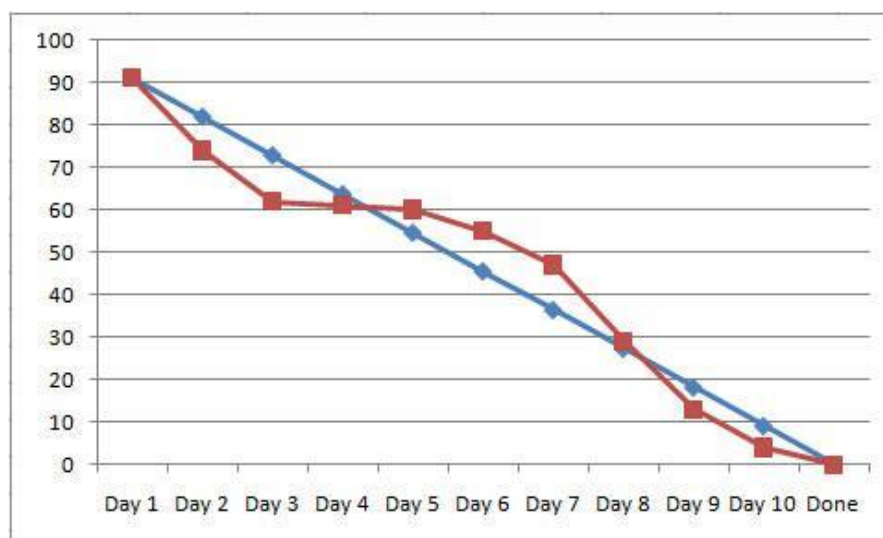
Fonte: Vieira (2014)

O *product backlog* não é estático, ou seja, conforme for andamento do desenvolvimento, a lista de prioridade pode ser alterada, ou até mesmo pode ser acrescentada novas funcionalidades (SCHWABER e SUTHERLAND, 2018). Com isso é possível afirmar que ao decorrer do desenvolvimento, as mudanças no *product backlog* gera uma interação entre o *product owner* e o time de desenvolvimento, que por sua vez, precisa adaptar-se ao constante fluxo de alterações que pode surgir ao decorrer do trabalho.

Para Schwaber e Sutherland (2018) a *sprint backlog* é um determinado conjunto de funcionalidades do *product backlog*, funcionalidades que devem ser desenvolvidas ao decorrer da *sprint*, também, apontam que com o gerenciamento do trabalho restante dentro da *sprint*, o time de desenvolvimento pode gerenciar seu progresso, com a utilização de várias práticas, por exemplo o gráfico de *burndown*.

Lopes (2014) afirma que o gráfico de *burndown* é um artefato ligado ao *sprint backlog*, que monitora o desenvolvimento da equipe. A imagem a seguir mostra o gráfico de *burndown*, onde a linha azul é estimativa ideal para o desenvolvimento da *sprint* e, a linha vermelha é o progresso da equipe de desenvolvimento.

Figura 3 - Gráfico de *burndown*.



Fonte: <http://codebalance.blogspot.com.br/2011/08/10-scrum-methodology-best-practices.html>

Para Schwaber e Sutherland (2018) o ***increment*** é a soma de todos os itens do ***product backlog*** ao final da *sprint*, onde independente da liberação do ***product owner*** ou não, deve estar pronto para ser utilizado.

2.2 GAMIFICAÇÃO

A gamificação segundo Kuuti (2013) e Deterding et al. (2011) é trazer elementos de jogos para um contexto não-jogo. A partir da conclusão dessa premissa não é possível chegar a uma explicação ampla respeito da gamificação sem mostrar os elementos de jogos e como são usados. As seções posteriores têm como foco mostrar os elementos de jogos, a gamificação e como os dois andam em conjunto.

2.2.1. Jogos

O jogo é uma atividade recreativa utilizada com o objetivo de distração e desfrute para a mente e o corpo, além disso servem para que as pessoas desenvolvam habilidades mentais (QUECONCEITO, 2018). Um jogo pode ser definido por (MELHORAMENTOS, 2018):

- “Qualquer atividade recreativa que tem por finalidade entreter, divertir ou distrair; brincadeira, entretenimento, folguedo.”;

- “Divertimento ou exercício de crianças em que elas demonstram sua habilidade, destreza ou astúcia”; e
- “Essa atividade, quando diferentes indivíduos ou grupos de indivíduos se submetem a competições em que um conjunto de regras determina quem ganha ou perde”.

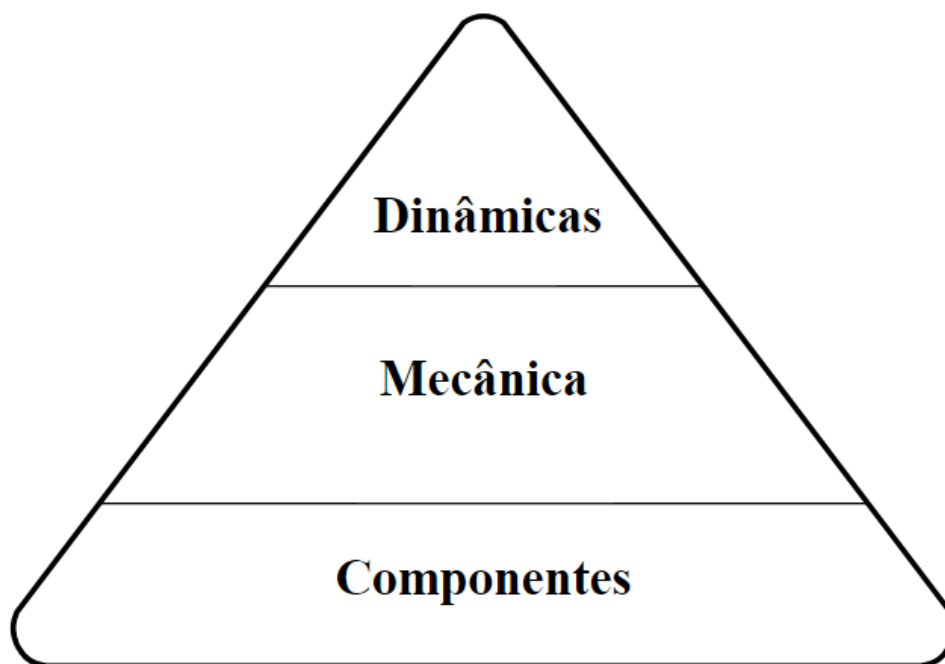
Existem diversos tipos de jogos, dos mais variados formatos, resoluções, plataforma ou público alvo, tipos de jogos que segundo Pantoja e Pereira (2018) podem ser classificados em 3, sendo eles:

- “Jogos Analógicos, conhecidos por serem um dos primeiros formatos de jogos criados pelo homem que ainda hoje se faz presente, e também propõem uma maior interação entre os jogadores, entre estes estão, jogos de cartas, jogos de tabuleiro, damas, xadrez e etc.”;
- “Os Jogos Pervasivos são aqueles que exploram uma experiência mais interativa com o jogador, o obrigando a sair do conforto do sofá para fazer movimentos físicos que são capturados por tecnologias como o Kinect, Oculus Rift, Nintendo Wii, dentre outros.”; e
- “E por fim temos os Jogos Digitais, que estão disponíveis nas mais diversas plataformas, como consoles, celulares, computadores e etc.”

Para o fim de determinar o que é um jogo e o que não é, é necessário avaliar se aquele “jogo” contém alguns elementos de jogos. Elementos jogos que estão presentes em todos os jogos e que na seção (2.1.1) posterior estão expostos.

2.2.1.1. Elementos de jogos

Os elementos de jogos descritos nesta seção são os mesmo que foram definidos por Werbach e Hunter (2012, p. 77), que estão organizados em forma de pirâmide sendo eles: Dinâmicas, mecânica e componentes. Como mostra a Figura abaixo.

Figura 4: Elementos de jogos

Kuutti (2013) aponta que as **dinâmicas** são os elementos presente em quase todos os jogos, mas, não é necessário que um jogo apresente todas. Werbach e Hunter (2012) definiu cinco dinâmicas, sendo elas: Restrições, emoções, narrativa, progressão e relacionamentos.

As restrições são o que impedem o jogador de realizar determinada ação, ou seja, limitam sua liberdade, assim, forçando o jogador a tomar decisões para conseguir ou não sair daquele estado. Em um jogo de cartas, como por exemplo o poker, é o que restringe que certa carta não seja maior que outra.

As emoções usam os sentimentos do jogador para deixá-lo sempre focado no jogo, ou seja, usam o sentimento do mesmo para que ele continue jogando. Isso pode ser por exemplo um sentimento de vitória, alegria, derrota, ou até mesmo apenas alegria por jogar.

A narrativa tem como foco mostrar para o jogador que o jogo tem o propósito final, não precisa ser literalmente uma história, mas basta dar coerência ao jogo. Como por exemplo um jogo de tabuleiro, como a dama, o jogador tem como propósito eliminar todas as peças do outro e assim cada jogada pode levá-lo para a vitória.

A progressão é o que faz com que os jogadores sintam que estão avançando no jogo. Em um RPG (*role-playing gamer*, do português, jogo de interpretação de papéis) a

progressão surge com os níveis, isso mostra para o jogador que quanto mais esforço ele colocar no jogo, mais “forte” ele fica e, mais níveis adquire.

Os relacionamentos se fazem no momento em que é disponibilizado, no jogo, que as pessoas que estão jogando podem ser companheiras, amigos ou adversários. Em um MOBA (*multiplayer online battle arena*, do português, arena de batalha multijogador online) como o *League of Legends* (do português, liga das lendas) fica evidente esses relacionamentos, pois o jogo se trata de duas equipes se enfrentando, uma aliada com cinco jogadores (companheiros) e um adversária com mais cinco jogadores (adversários).

Cardoso (2015) afirma que a **mecânica** representa os processos básicos para o jogador progredir no jogo, além disso ele também afirma que uma mecânica é uma forma de conseguir uma ou mais dinâmicas. Werbach e Hunter (2012) aponta a mecânica como a responsável por não só atrair novos jogadores, mas também manter os mais experientes dentro do jogo. Werbach e Hunter (2012) definem dez mecânicas, sendo elas: Desafios, chance, cooperação e competição, feedback, aquisição de recursos, recompensa, transação, turno e estado de vitória.

Os desafios são algo que o jogador deve cumprir. Em alguns jogos podem ser chamados de missões, geralmente com uma recompensa ao final da missão. Recompensa que também faz parte da mecânica, e, é algo que o jogador ganha não só por realizar uma missão, mas também por diversas outras atividades dentro do jogo.

Para Cardoso (2015) a chance “é o elemento sorte do jogo que cria uma sensação de surpresa e incerteza”. Analisando essa informação pode se tirar que a chance é um momento que depende unicamente da sorte do jogador, que pode resultar em vitória ou derrota.

A cooperação e competição são algo que trabalham em conjunto, pois na cooperação pode surgir a competição através do possível sentimento de derrota, ou vitória, que aparece com o decorrer do tempo.

O feedback é o que permite o jogador visualizar seu progresso no jogo (CARDOSO, 2015). O feedback é uma parte chave, pois com ele o jogador não fica perdido com o que ele realizou dentro do jogo, como por exemplo as missões realizadas,

as que não foram realizadas, a experiência, e etc. Em um ambiente gamificado com o Habitica, pode ser observado como a imagem a seguir:

Aquisição de recursos é uma maneira do jogador adquirir o que o jogo fornece para ele, por exemplo dinheiro, itens, chances. E a partir desses recursos os jogadores podem realizar transações dentro do jogo com algum NPC (*non-player character*, do português, personagem não jogável) ou com outros jogadores.

O turno é o tempo de cada jogador para a interação com o jogo, não necessariamente um jogo precisa de turnos. Em um jogo com o Super Mario Bros, lançado pela Nintendo, numa partida com dois jogadores, os turnos podem ser dividido em dois, onde em cada turno um dos jogadores joga.

Figura 5: Estado de vitória no *League of Legends*

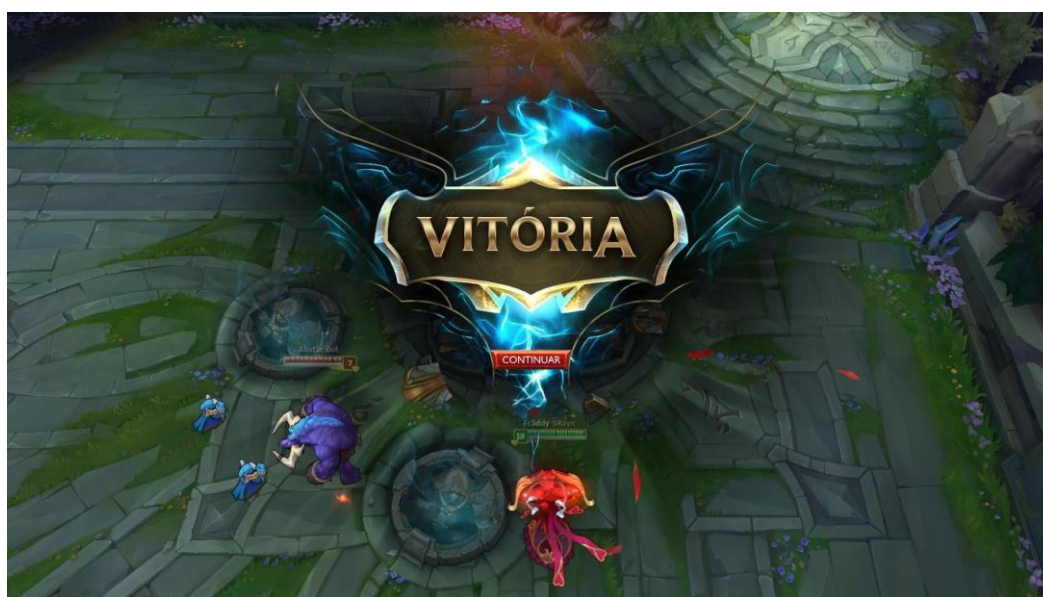
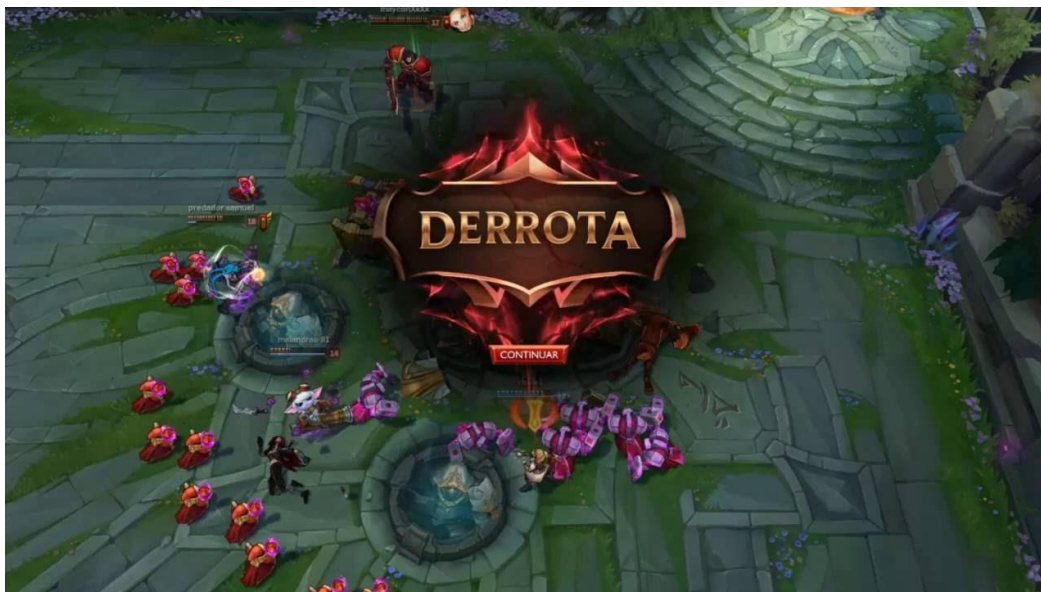


Figura 6: Estado de derrota no *League of Legends*



Estado de vitória é o que define quem ganha e quem perde. Por exemplo no *League of Legends*, ao final de cada partida o estado é mostrado para cada jogador, como mostra as Figuras acima.

Cardoso (2015) afirma que os **componentes** podem ser vistos usando a interface do jogo. Werbach e Hunter (2012) define 15 componentes, sendo eles: Conquistas, missão, emblemas, avatar, lutas contra chefe, coleções, combates, conteúdo de desbloqueio, presentes, quadro de líderes, níveis, pontos, gráfico social, equipes e bens virtuais. Explicados por Cardoso (2015) como:

As **conquistas** segundo Cardoso (2015) “dão ao jogador recompensas, geralmente, quando ele completa uma **missão**, que é composta por um conjunto de tarefas específicas”. No *League of Legends* isso pode ser visto como os resultados das missões, como mostra as Figuras abaixo.

Figura 7: Missões no *League of Legends*

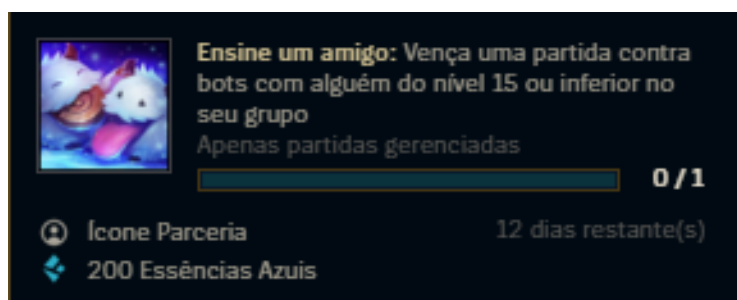


Figura 8: Recompensas no *League of Legends*



Deste modo no *League of Legends* as conquistas seriam as recompensas dos jogadores ao realizar uma missão, não necessariamente essa é a única forma de conquista no *League of Legends*, mas sim, uma das mais evidentes.

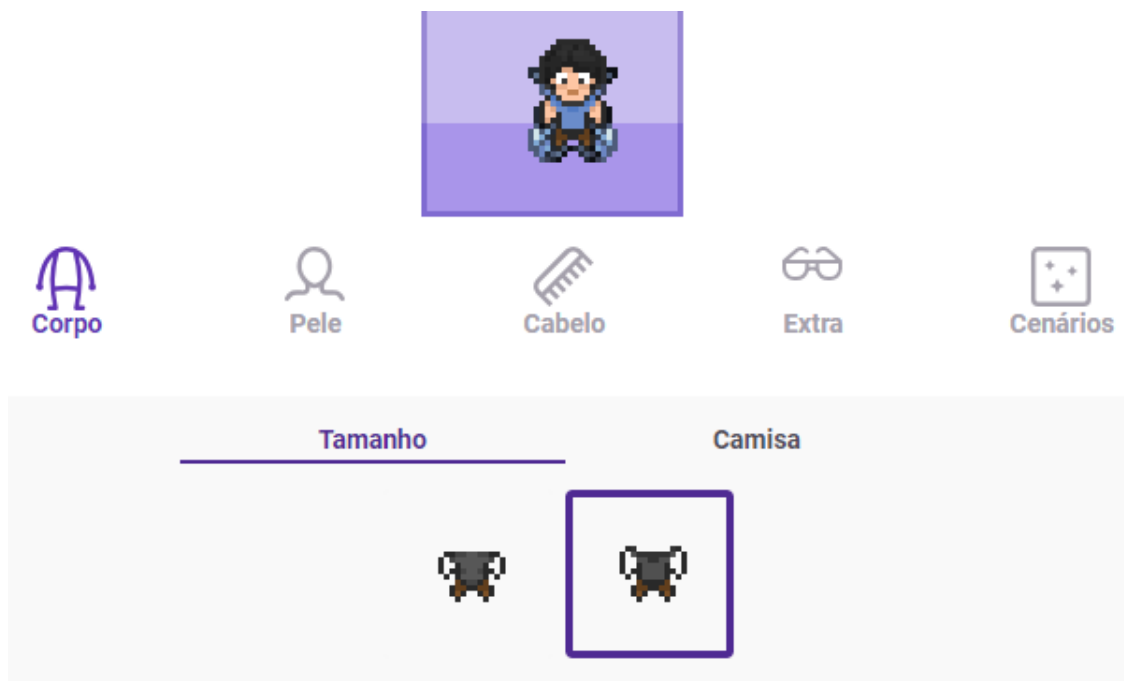
Cardoso (2015) aponta que os **emblemas** são “a representação visual de uma conquista. Ainda no *League of Legends* além do ícone de parceria (da Figura 8) como um emblema também é tido o ícone de maestria, que é um emblema entregue para jogadores que conquistaram uma boa performance no jogo com aquele personagem, como mostra a imagem abaixo.

Figura 9: Ícone de maestria para campeões do *League of Legends*



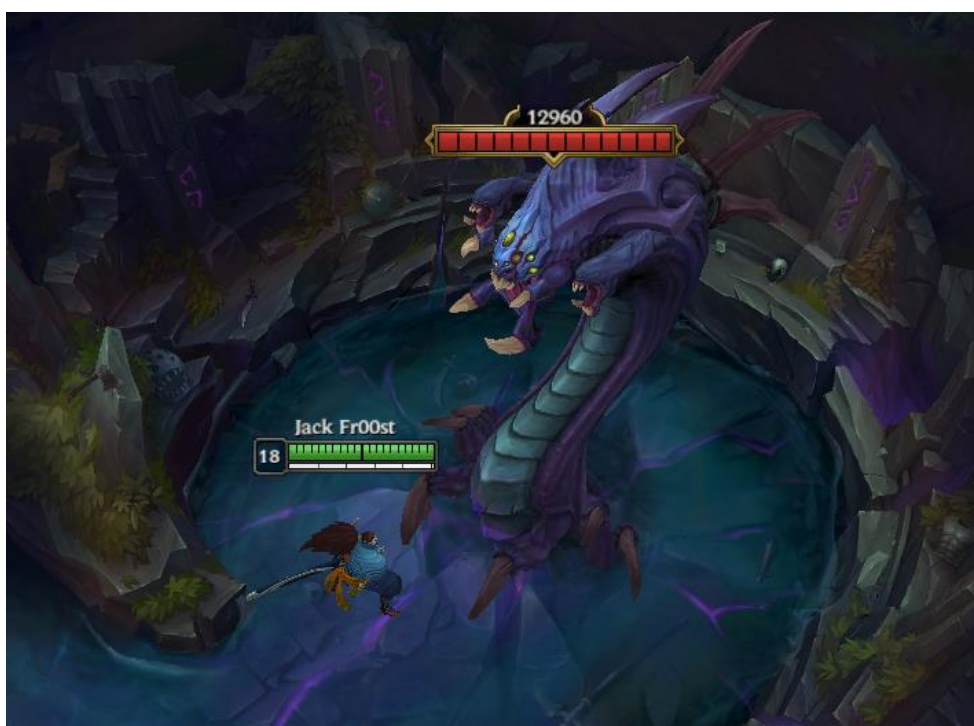
O **avatar** “é a representação visual do personagem do jogador” (CARDOSO, 2015). Em um sistema gamificado com o Habitica, o jogador pode criar o seu personagem e o caracterizar da maneira que preferir.

Figura 10: Edição de avatar no Habitica.



Cardoso (2015) aponta que as **lutas contra chefes** “são desafios muito difíceis, normalmente no final de um nível, que tem de ser derrotado a fim de progredir no jogo”. E no *League of Legends* isso pode ser entendido como lutar contra o Barão, que é o um dos maiores desafios dentro do jogo.

Figura 11: Barão *League of Legends*.



Já as **coleções** “são um conjunto de itens em jogos que se relacionam entre si” (CARDOSO, 2015). E em um jogo como o *PokeXGames* se dão através dos itens que o jogador consegue com o decorrer do jogo.

Figura 12: Coleções no PokeXGames.



O **combate** pode ser entendido como “o ato de lutar e derrotar oponentes no jogo”. No *League of Legends* isso é algo bastante evidente, pois o jogo se trata de duelos diretos contra outros jogadores.

Figura 13: Duelo entre jogadores



Conteúdo de desbloqueio “são conteúdos que originalmente são bloqueados nos jogos, sendo necessário o jogador fazer uma ação específica para desbloqueá-los” (CARDOSO, 2015). No *PokeXGames* pode ser visto como algo que só é liberado após determinado nível, como por exemplo, certas habilidades. Como mostra a Figura abaixo.

Figura 14: Nível das habilidades no *PokeXGames*



Os **presentes** “permitem o jogador dar itens ou moeda virtual para outros jogadores” (CARDOSO, 2015). Não só no *PokeXGames*, mas em diversos jogos com suporte para multijogadores, isso pode ser visto com um sistema de *trade* (do português, comercio).

Figura 15: Sistema de *trade* no *PokeXGames*.



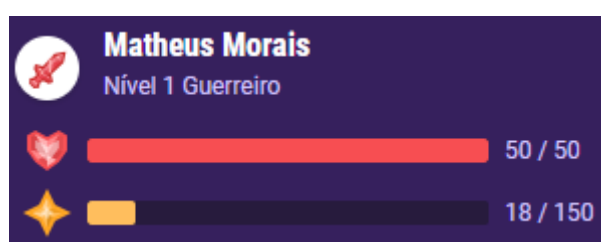
Os quadros **de líderes** “listam os jogadores de acordo com suas pontuações” (CARDOSO, 2015). No *League of Legends* isso acontece separando os jogadores em divisões (bronze, prata, ouro, platina, diamante, mestre e desafiante), sendo o desafiante a divisão com uma melhor colocação.

Figura 16: Top dez jogadores do servidor brasileiro de *League of Legends* da data 14/03/2018.

#	INVOCADORES	V/D	PONTOS
1	Eu sou o Jojo	263/177	1352
2	CNB Hy0g4	558/449	1343
3	1  tinowns2	178/123	944
4	1  YoYo JoJo 	258/159	914
5	3242363	99/43	856
6	1  HKS Caos	320/267	852
7	1  Sky s2 BR	230/161	840
8	1  Rhaegalarys	212/153	823
9	1  God Needles	363/317	817
10	4  RED Guigo 	332/269	816

Segundo Cardoso (2015) os **níveis** “são a representação de quão bom o jogador está no jogo, que é medido, geralmente, através da quantidade de **pontos** que o jogador obtém ao realizar algumas ações no jogo”. No *Habitica* os níveis seguem a lógica explicada por Cardoso (2015), representados da seguinte maneira.

Figura 17: Representação de níveis no *Habitica*.



Já o **gráfico social** segundo Cardoso (2015) “permite ver amigos que também estão no jogo e ser capaz de interagir com eles”. No *League of Legends* isso acontece na aba de amigos, que é onde ocorre a interação entre o jogador e os amigos conectados.

Figura 18: Aba de amigos no *League of Legends*.



As **equipes** “são um conjunto de jogadores trabalhando em conjunto em prol de um mesmo objetivo” (CARDOSO, 2015). No *League of Legends* por se tratar de um MOBA (*multiplayer online battle arena*, do português arena de batalha multijogador online), torna a existência de equipe essencial.

Figura 19: Formação de equipes no *League of Legends*.



Cardoso (2015) define os **bens virtuais** como os “itens que os jogadores podem coletar e utilizar”. Como o dinheiro virtual do *PokeXGames*, que é ilustrado na Figura a seguir.

Figura 20: Dinheiro virtual do *PokeXGames*.



Kuutti (2013) afirma que todos os elementos de jogos descritos por Werbach e Hunter (2012) podem ser usados como elementos de gamificação, mas, alguns podem não ser igualmente relevantes. Ou seja, cabe ao desenvolvedor da determinada gamificação escolher os elementos de jogos que mais se encaixam em seu problema, isto é, escolher os mais relevantes.

2.2.2. Elementos de Gamificação

Para Espíndola (2018) a gamificação está presente em quase tudo, e tem como foco aumentar o engajamento e despertar a curiosidade dos usuários, tudo isso com base em usar mecanismos de jogos para incentivar as pessoas. A gamificação traz como foco melhorar o engajamento dos usuários nas atividades propostas por ela, mas para isso, é necessário criar uma boa gamificação. Neste contexto para a criação de uma boa gamificação, não é somente trazer os elementos de jogos que forem mais convenientes, para isso é necessário um estudo avançado a respeito da área que vai ser gamificada.

Costa e Marchiori (2015) afirmam quanto “quanto mais informações são coletadas sobre o negócio, maior é a chance de se projetar uma experiência eficaz e relevante de gamificação. Definidas essas questões, a matriz derivada da pesquisa pode auxiliar na escolha dos elementos mais adequados ao contexto e objetivo de negócio”.

Nesse quesito Werbach e Hunter (2012) criaram seis passos para a criação de uma gamificação eficiente, que mais tarde foi nomeado de *framework D6* por Kuutti (2013). Técnicas essas que são descritos na seção abaixo.

2.2.2.1. Framework D6

Kuutti (2013) o nomeou de *framework D6* pelo fato de todas os seus seis passos começam com a letra “D”. Definidos por Werbach e Hunter (2012) seus passos são: *Define business objectives, delineate target behaviors, describe your players, device activity cycles, don't forget the fun e deploy the appropriate tools*. Como mostra a imagem abaixo:

Figura 21: Os seis passos para a implantação da gamificação (KUUTTI, 2013)
adaptado por Cardoso (2015).



Para um melhor entendimento os nomes foram traduzidos e adaptados por Cardoso (2015). E segue uma explicação acerca de cada um dos passos.

A etapa de **definir objetivos de negócios** deve ser definido o que a gamificação vai fazer, ou seja, o objetivo dela. Werbach e Hunter (2012) aponta que para a gamificação ser eficaz, deve-se saber os objetivos, o público alvo, entre outros. Esta etapa se assemelha bastante a etapa de definição de requisitos no desenvolvimento de software.

Cardoso (2015) elaborou duas perguntas que auxiliam nesse passo, sendo elas: “Por que gamificar o negócio?” e “Quais as necessidades do negócio?”. Partindo dessa premissa pode-se realizar outras perguntas a respeito dessa etapa, por exemplo: Como a gamificação pode resolver o problema de negócio? Quais benefícios a gamificação traria para o negócio? Quais os possíveis problemas no negócio com a gamificação?

Cardoso (2015) aponta que na etapa de **delimitar comportamento alvo** deve “definir o que os jogadores podem fazer no sistema e como medir isso”. Ou seja, nessa

etapa vai ser definido como o ambiente se comporta, levando em consideração a etapa anterior.

Werbach e Hunter (2012) afirmam que a gamificação deve se adequar a mais de um subgrupo de jogadores, pois, nem todos os jogadores são iguais. Dessa forma na etapa de **descrever seus jogadores** deve ser formado um perfil de jogador, ou seja, deve ser criado um perfil de jogador de acordo com o público alvo da gamificação.

Em um MMORPG (*Massively* ou *Massive Multiplayer Online Role-Playing Game* ou *Multi massive online Role-Playing Game*, do português, jogo de interpretação de personagens *online* e em massa para multijogadores) como o *Black Desert* (do português, deserto negro) lançado pela *RedFox Games*, o jogador conta com várias classes e raças para se adequar ao perfil de cada um.

A etapa de **planejar ciclos de atividades** é um ponto bastante importante dentro do *framework D6* pois é ele que define como vai ser a vida do jogador dentro do jogo. Werbach e Hunter (2012) apontam que a maneira mais eficaz de moldar as ações dentro de um sistema gamificado é através do ciclo de atividades. O conceito dos ciclos de atividades pode ser comparado com a terceira lei de Newton onde fala que toda ação tem uma reação. Kuutti (2013) diz que esse conceito mostra que uma ação do jogador desencadeia uma outra ação, que por sua vez produz outra ação.

Werbach e Hunter (2012) afirmam que existem dois ciclos de atividades, sendo eles: Laços interligados e escadas de progressão. Os laços interligados descrevem o que o jogador faz, porque ele faz isso e a resposta do sistema, já as escadas de progressão dão uma perspectiva macro da jornada do jogador (CARDOSO, 2015).

Para um exemplo dos laços interligados, pode-se tirar uma partida de *League of Legends* quando um jogador atinge um total de três abates inimigos, sem morrer, entra em *killling spree*. ou seja, ao abater 3, sem morrer, ativa no sistema para todos verem que o jogador está em *killling spree*, o que motiva os outros jogadores (adversários) a forçar o abate em cima do jogador em *killling spree*, pelo fato dele valer mais *gold* (do português, ouro. Moeda dentro do jogo) ou para chegar em *killling spree*.

Para um exemplo de escadas de progressão pode ser citado vários jogos como *Rappelz*, *Black Desert*, *Aion*, *Mu*, *Tibia*, *Cabal*, entre outros. O que é importante ressaltar é que conforme o nível do jogo aumenta, a dificuldade para o mesmo passar de

nível aumenta também. Ou seja, com o passar dos níveis é necessário um empenho maior do jogador. Cardoso (2015) afirma que “isso é feito propositalmente para que o jogo se torne cada vez mais desafiante de acordo com que a experiência do usuário aumenta”.

Segundo Werbach e Hunter (2012) na etapa de **garantir diversão** a última coisa a se fazer antes da implementação é uma única pergunta: É divertido? Eles ainda afirmam que a diversão é algo que em nenhuma hipótese pode ser rejeitada de um jogo como um sistema gamificado. “Quando há diversão no sistema, os jogadores serão mais propensos a retornarem e jogarem novamente, já que a diversão causa também um efeito motivador”(CARDOSO, 2015).

A diversão pode ser vista como algo que deixe o jogadores com mais êxtase de jogar, algo parecido como um sentimento de nostalgia de um adulto a jogar um jogo que possa ter marcado a sua infância.

Sendo a última etapa do *framework D6*, a etapa de **implementar as ferramentas apropriadas** é realmente acontece a implementação da gamificação. Segundo Kuutti (2013) os elementos de jogos devem ser escolhidos levando em consideração todos os cinco passos anteriores.

Para Cardoso (2015) é necessária uma equipe com uma variedade de habilidades para que de fato tenha uma gamificação eficaz. Habilidades essas que foram definidas por Werbach e Hunter (2012) e explicadas por Cardoso (2015), sendo elas:

- **Habilidade de entender os objetivos de negócio do projeto:** “Mesmo os melhores designers de jogos se não conseguirem entender os objetivos de negócio do sistema haverá sempre o risco de produzirem algo inútil, ou seja, que não condiz com o esperado do sistema”.
- **Habilidade de compreensão do seu grupo-alvo de jogadores e as noções básicas de psicologia:** “Compreender os tipos e grupos de jogadores tem grande relevância para a adesão do jogo por vários tipos de jogadores (Seção 2.2.5.3). Com posse dessas informações de tipos de jogadores aliado com um profissional com conhecimento mínimo de psicologia pode tornar o processo de identificação de perfil de jogadores mais efetivo, já que poderão ser analisadas suas características emocionais”.

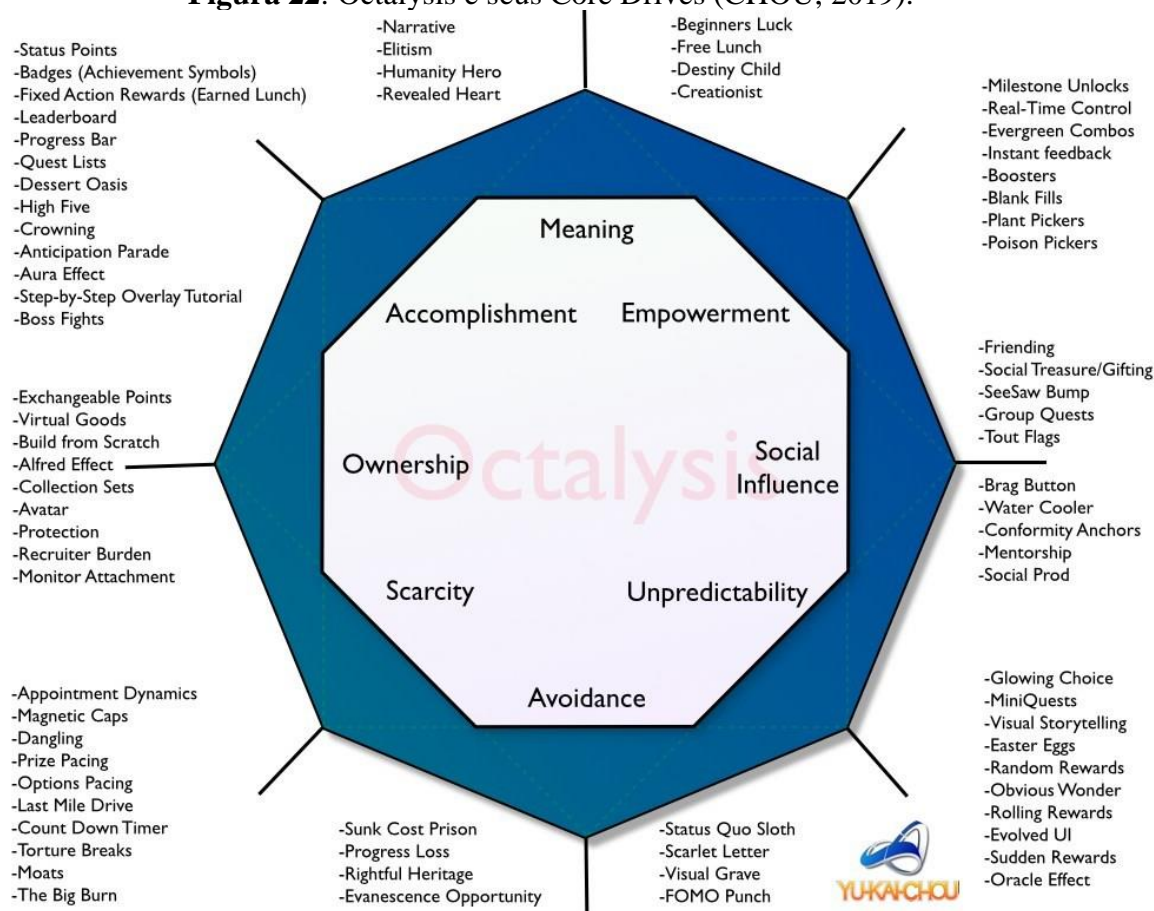
- **Habilidade de criação de jogos:** “Ter um designer de jogos ou alguém com habilidade de criação de jogos é primordial já que um sistema gamificado tem características semelhantes a um jogo”.
- **Habilidade em análise de dados:** “Um especialista em análise de dados pode fazer com que os dados gerados no sistema gamificado façam sentido e se tornem úteis para serem aplicados quando necessários”.
- **Habilidade de implementar a sua visão:** “De nada adiantará ter alguém com a capacidade de ter uma visão se este não é capaz de a pôr em prática. Sem essa habilidade, não faz mais sentido ter as demais, uma vez que sem a efetiva aplicação da visão, as chances de sucesso são baixas”.

2.2.2.2. Octalysis Framework

O Octalysis é um *framework* criado por Yu-kai Chou, durante 10 anos de pesquisa, com o objetivo de criar uma gamificação focada no humano (CHOU, 2019). Ou seja, leva como base a experiência do jogador, do qual pode experimentar sentimentos diferentes em um determinado jogo.

Um jogo é divertido quando atrai determinado ponto emocional de um jogador, logo após o jogador ter esse ponto estimulado, se sente mais motivado e engajado em seu determinado objetivo no jogo, em vista disso existem 8 pontos que atraem o jogador, dos quais são chamados de Core Drives (CHOU, 2019).

Figura 22: Octalysis e seus Core Drives (CHOU, 2019).



A Figura 22 reflete o Octalysis, da qual representa um octógono, onde em cada um dos seus 8 lados, refere a um determinado core drive.

Segundo Chou (2019), os **Core Drives** têm a função de impulsionar o jogador a progredir no jogo, de modo a superar desafios ou missões, que podem ser extremamente difíceis ou não.

No contexto de jogos, isso pode ser observado como uma missão que necessite de requisitos que façam com que os jogadores tenham que superar tudo que fizeram, ao decorrer do jogo, para alcançá-los e assim a finalizar.

Significado épico e chamada (*meaning*)

Chou (2019) aponta que esse *core drive* tem como função mostrar para o jogador que ele está fazendo parte de algo maior, de que ele foi escolhido especialmente para um determinada tarefa. A grosso modo pode ser entendido como uma forma do jogador perceber que aquilo que ele está fazendo é algo significativo para o todo.

Buscando trazer para o contexto de jogo, pode ser percebido que esse *core drive* tem como base mostrar os resultados de suas ações dentro do jogo, trazer sentido e reações para tudo que ele fizer.

Desenvolvimento e Realização (*accomplishment*)

Esse *core drive*, segundo Chou (2019), tem a função de impulsionar o jogador a progredir no jogo, de modo a superar desafios ou missões, que podem ser extremamente difíceis ou não.

No contexto de jogos, isso pode ser observado como uma missão que necessite de requisitos que façam com que os jogadores tenham que superar tudo que fizeram, ao decorrer do jogo, para alcançá-los e assim a finalizar.

Capacitação da criatividade e *Feedback* (*empowerment*)

Esse *core drive* tem a função de incentivar a criatividade do jogador, de modo que o jogador por meio da repetição possa pensar em algo novo para cada vez mais desenvolver sua criatividade, no desenvolvimento de algum problema, mas não só isso, os jogadores também precisam de algo para visualizar o que estão fazendo e receber *feedbacks* constantes a respeito o que estão fazendo (CHOU, 2019).

De modo geral, pode ser visto como uma forma de usar a criatividade para atingir determinado resultado, do qual, por exemplo, em um jogo, da onde conforme for a criatividade do jogador, mais diferente e interessante será o resultado final.

Propriedade e Posse (*ownership*)

Chou (2019) mostra que esse *core drive* é responsável por provocar um desejo de acumular riqueza dentro do jogo, do qual, inicialmente os jogadores são motivados a possuir algo, motivação essa que gera o desejo de posse, do qual gera um desejo de possuir sempre algo melhor.

Em um contexto de jogo, isso pode ser observado no quanto esse jogador gosta de acumular seus itens, e no desejo de sempre querer mais, em jogos de MMO esse sentimento fica evidente, logo que, nesses jogos quanto mais o jogador tiver, melhor.

Influência social e Relacionamento (*social influence*)

Esse *core drive* impulsiona os jogadores, em relação a tudo que tem a ver com a questão social, por exemplo, os amigos, aceitação pela comunidade, resposta sociais, companheirismo e entre outros, tipo isso, pode ser pego o exemplo de um amigo que tem habilidade extraordinárias em determinada área, logo o jogador se sente levado a alcançar o mesmo nível (CHOU, 2019).

Em um contexto geral do jogos, podem ser pegos os *ranking* como exemplo, dos quais o jogador que ocupa o primeiro lugar é sempre o melhor, do jogo em questão, logo, o restante será levado a ultrapassar esse nível, para se tornar o melhor jogador da comunidade do jogo.

Escassez e Impaciência (*scarcity*)

Segundo Chou (2019), esse *core drive* tem o foco de incitar o desejo do jogador por obter algo que ele não possa ter, logo o fato do jogador não ter algo, que ele queira, faz com que o mesmo fique pensando nisso incessantemente.

Buscando trazer o seguinte contexto: “apenas os 10 melhores jogadores do dia, vão conseguir o item X”. Com isso, os jogadores que não conseguiram ficar entre os 10 primeiros, vão ficar motivados a nas próximas vezes a conseguirem entrar na posição que faça com que ganhem o determinado item.

Imprevisibilidade e curiosidade (*unpredictability*)

De modo geral, segundo Chou (2019) esse *core drive* tem a função de trabalhar a curiosidade do jogador, de modo a fazer com que o jogador sinta uma vontade de descobrir o que vem adiante. Ou seja, o que aconteceu quando finalizar determinado evento dentro do jogo.

Em um jogo com uma história bem definida, esse *core drive* fica bastante evidente, logo que além da jogabilidade, outro fator que atrai a atenção dos jogadores é a história, onde descobrir o final da história pode ser um dos motivos do jogador continuar a jogar.

Perda e Evitação (*avoidance*)

Chou (2019) afirma que esse ponto se refere a intenção do jogador de evita perder algo dentro do jogo, do qual em pequena escala pode ser evitar perder o progresso do trabalho anterior, e em grande escala é perder tudo em que o jogador trabalhou no jogo.

Em um jogo de apostas, como o Poker, esse elemento é bastante evidente, logo que o ato do jogador apostar e não perder esse valor apostado, aponta claramente uma intenção de evitar o processo de perda dessa aposta ou de perda no âmbito geral do jogo.

Um bom sistema gamificado não necessariamente precisa ter aplicado todos os *core drives*, porém o ideal é uma pontuação bem definida (CHOU, 2019). De modo geral, essa pontuação define o peso de cada *core drive* dentro do ambiente.

Desta forma, primeiramente deve ser pontuado cada core drive, e em seguida definido os elementos de jogos que compõem a plataforma gamificada, e por fim criar as funcionalidades (CHOU, 2019).

2.3 MINERAÇÃO DE REPOSITÓRIO DE SOFTWARE

Silva (2013) define os repositórios de *software* como “depósitos de informação sobre as atividades de desenvolvimento e de manutenção de um software”. Já a mineração de repositório de software “é uma área que surgiu da necessidade de investigar fatos relevantes sobre projetos, produtos e pessoas no contexto de desenvolvimento de software; esta área ainda está amadurecendo devido ao aumento da quantidade de repositórios de código aberto” (SILVA, 2017).

A mineração de repositório de software segue a premissa de que no desenvolvimento de *software* são criados dados (como, alteração de *software*, criação de uma nova funcionalidade, deletar determina funcionalidade, entre outros) e esses dados podem ser acessados e utilizados para ajudar a resolver problemas presentes no desenvolvimento de *software*.

Para que exista a mineração de repositório de software é necessário que tenha uma base de dados para ser analisada, base de dados essas que se dão através dos repositórios de *software*. Que segundo Hassan (2008) existe cinco tipos de repositórios de *software*, sendo eles: Controle de versionamento de código, repositórios de bug,

arquivos de comunicação, logs de interação e repositórios de código. Explicados por Silva (2017):

- “Controle de versionamento de código, que são responsáveis por armazenar o histórico de desenvolvimento de um sistema ”;
- “Repositórios de bug, que são responsáveis por manter informações de bugs e issues”;
- “Arquivos de comunicação, como listas de e-mails, histórico de mensagens em chat interno, que podem rastrear discussões sobre projetos através do tempo ”;
- “Logs de integração”; e
- “Repositório de código, que é responsável por armazenar os códigos remotamente”.

O GitHub por exemplo é um repositório de software com controle de versionamento de código usando o git. Ou seja, além de ser responsável por armazenar os códigos ele também é responsável por armazenar o histórico de desenvolvimento dos criadores de código-fonte.

No presente trabalho a mineração não vai ser utilizada para obter os dados (como *issues*, *commits*, *milestone*) do repositório de *software* usado pela fábrica de *software* do CEULP/ULBRA, chamado Gitlab, com o propósito de auxiliar na criação de métricas para a gamificação do desenvolvimento de *software*.

No contexto de mineração de repositório de *software*, existe algumas ferramentas que devem ser citadas, ferramentas essas que estarão expostas no tópico seguinte.

2.3.1. Ferramentas de mineração de repositório de software

O trabalho de Roma (2013) por exemplo é sobre uma ferramenta para a mineração de dados de projetos de *softwares* livres e criação de redes sócio técnicas. Além disso Roma também aponta outras ferramentas que trabalham de forma semelhante, ferramentas essas sendo: *xFlow*, *Tesseract*, *Evoltrack-SocialNetwork* e o *BugMaps*. Abaixo um quadro comparativo entre a ferramenta de Roma e as citadas por ele.

Figura 23 - Comparação entre os trabalhos relacionados.

Funcionalidades principais da ferramenta proposta neste trabalho	Ferramentas relacionadas				
	xFlow	Tesseract	Evoltrack-SN	BugMaps	Ferramenta proposta
Mineração de dados técnicos	X	X	X	X	X
Mineração de dados sociais	X		X		X
Geração de redes técnicas	X	X	X		X
Geração de redes sociais		X	X		X
Geração de redes sócio-técnicas		X	X		X
Cálculo de métricas sociais					X
Cálculo de métricas técnicas	X			X	X
Exportação das redes sociais em arquivos CSV ou equivalentes.					X
Exportação das métricas em arquivos CSV ou equivalentes.				X	X
Adição de novas redes e métricas	X		X		X
Plataforma web		X			X

Fonte: Roma (2013)

Santana et al. (2011) afirma que o *xFlow* possibilita o estudo de como os desenvolvedores participam do processo de evolução dos sistemas de software ao longo do tempo. Com por exemplo a coleta de dados de repositórios (*git*), a análise em modificações paralelas de artefatos e métricas para artefatos baseados em linha de código/adicionados/removidos/modificados (ROMA, 2013).

Já o *Tesseract* segundo Sarma et al. (2009) “é uma ferramenta que permite a exploração sócio-técnica a partir da análise dos arquivos de código, registros de comunicação e banco de dados do projeto para capturar as relações entre código, desenvolvedores e defeitos” (apud ROMA, 2013). Roma (2013) ainda aponta que o *Tesseract* tem uma das funções “mostrar simultaneamente as relações sociais, bem como técnica entre entidades de projeto diferentes (por exemplo, os desenvolvedores, a comunicação, os códigos e os defeitos)”.

E o *BugMaps* segundo Hora et al. (2012) “é uma ferramenta para a exploração visual e análise de defeitos que fornece mecanismos para automatizar o processo de análise de dados de repositórios de software” (apud ROMA, 2013).

E a ferramenta proposta por Roma (2013) tem o intuito de “oferecer uma plataforma para pesquisadores implementarem suas próprias redes e métricas; facilitar o

entendimento da real situação de um projeto de desenvolvimento de software; e prover a descoberta de tendências em um projeto de software”

3 MATERIAIS E MÉTODOS

Nessa seção são abordados os materiais e métodos utilizados no presente trabalho.

3.1. MATERIAIS

3.1.1. Octalysis

O Octalysis é um *framework* criado por Yu-Kai Chou, desenvolvido exclusivamente para gamificação. Chou (2019) aponta que são 8 core drives (*meaning, empowerment, accomplishment, ownership, social influence, scarcity, unpredictability e avoidance*) que estimulam os jogadores dentro dos jogos, com isso ele chegou ao Octalysis que organizam a gamificação nos seus 8 core drive.

O Octalysis foi utilizado para definir os core drive predominantes no presente trabalho, para assim poder definir os elementos de jogos que se encaixem melhor na gamificação.

3.1.2. Gitlab

O Gitlab é um gerenciador e versionador de repositórios de código que conta com duas edições: Gitlab CE (*Community Edition*) e o Gitlab EE (*Enterprise Edition*). Podendo explicadas como o Gitlab CE sendo a edição gratuita para a comunidade, e o Gitlab EE a edição paga. A Fábrica de Software do CEULP/ULBRA utiliza o Gitlab CE como repositório de código, e mesmo sendo a versão gratuita conta com funcionalidades que permitem o registro dos dados do desenvolvimento (como *commits, issues e milestones*).

Tanto o Gitlab CE quanto o Gitlab EE contam com uma API (*Application Programming Interface*) que disponibiliza funcionalidades sobre o HTTP (*Hypertext Transfer Protocol*), funcionalidades essas que auxiliam na extração dos dados, a respeito do desenvolvimento, do Gitlab.

As informações fornecidas pelo Gitlab, geradas a partir das atividades dos membros da Fábrica de Software, foram utilizadas como base para a criação da gamificação deste trabalho.

Além disso, o Gitlab também disponibiliza um sistema de *webhooks*. Os *webhooks* que são uma maneira passiva de receber determinadas informações, dos quais

através de um determinado evento, é enviado uma informação para um determinado local.

3.1.3. Angular

O Angular é um *framework* utilizado para construir aplicações web que se baseiam em HTML, CSS e *JavaScript*, assim com sua utilização, ele entrega uma aplicação que executa no *browser* e é capaz de consumir vários serviços dispostos em um servidor. Com o Angular é criado uma aplicação chamada de lado do cliente (do inglês *client side*), também conhecido como *FrontEnd*.

O presente trabalho utilizou o Angular para criar uma aplicação que comunica com a API do *backend* do presente trabalho, e extrai as informações necessárias para a gamificação, e apresenta para os usuários conectados de forma gamificada.

3.1.4. Plataforma de programação

A linguagem de programação foi utilizada Python que é uma linguagem de programação lançada em 1991 por Guido van Rossum. Python possui diversas bibliotecas, e, a partir delas é possível ter uma experiência vasta com várias áreas da computação, como por exemplo: Criação de jogos, *machine learning* e interação com a web (BRASIL, 2018).

Dentre essas bibliotecas a que foram utilizadas no presente trabalho é a *python-gitlab* que por meio de suas funções permite uma interação com a API do Gitlab de maneira simples e eficaz, com isso vai ser possível a obtenção dos dados e assim será feita uma análise dos mesmos, análise essa que é fundamental para a criação das métricas para a gamificação.

Além da biblioteca *python-gitlab*, também foi utilizado um *framework* do python, chamado de Django Rest *Framework*, que foi responsável por criar o *backend* do presente trabalho.

3.2. MÉTODOS

Com o objetivo de organizar a execução do presente trabalho, esta foi dividida em três etapas, sendo elas: compreensão do contexto, projeto e implementação. A etapa de

compreensão do contexto foi responsável por gerar os dados do Octalysis. A etapa de projeto foi responsável por aplicar as técnicas do framework D6. A etapa de implementação foi a responsável por implementar o que foi planejado nas etapas anteriores. As seções a seguir detalham o que foi feito em cada etapa.

3.2.1. Etapa de compreensão do contexto

Na etapa de compreensão do contexto, foram aplicadas as técnicas definidas no Octalysis para a criação do Octacore e definição dos Core Drives que foram mais utilizados no presente trabalho. De modo a ficar mais didático, essa etapa foi dividida em três passos, sendo eles: Criação/aplicação do questionário, análise/pontuação das respostas do questionário e criação do Octacore.

A **criação do questionário** ocorreu juntamente como prof. Orientador Jackson Gomes. Esse questionário contém dezesseis (16) questões, e essas questões foram divididas em oito grupos, esses grupos foram relacionados com os oito Core Drives do Octalysis, onde cada grupo de questões tem perguntas associadas com um determinado Core Drive, de modo que para cada questão haja uma resposta entre 1 a 5.

Após a criação o questionário, o mesmo foi disponibilizado para os integrantes da Fábrica do Software do CEULP/ULBRA responderem.

A **análise/pontuação** foi feita com as respostas do questionário, onde foram somadas as respostas de cada questão, com o resultado da soma foi retirada a média em relação as pessoas que responderam, o resultado disso foi 16 medias, ou seja, a média das respostas das 16 questões do questionário. Para finalizar essa etapa, foi somado as medias de cada core drive. Com a pontuação feita, a influência de cada Core Drive dentro da gamificação ficou evidente, onde, o Core Drive com maior pontuação foi o mais influente.

A **criação do Octacore** foi realizada de modo a colocar a determinada pontuação do Core Drive em suas respectivas locações, assim, foi criado um octógono onde cada lado mostra o determinado Core Drive e, conforme a pontuação, sua influência.

3.2.2. Etapa de projeto

Com a finalização da etapa de Compreensão do contexto foi iniciada a etapa de projeto, desenvolvida seguindo as seis etapas propostas por Werbach e Hunter (2012) no framework D6. Cada etapa foi desenvolvida levando em consideração os dados obtidos com o Octalysis, na etapa de compreensão de contexto (3.2.1.). Os passos da etapa de Projeto são:

1. **Definir Objetivos de negócios:** Nesta etapa foi decidido, juntamente com o prof. Jackson Gomes, foi efetuado um acompanhamento das atividades realizadas entre o time de desenvolvimento da Fábrica de Software do CEULP/ULBRA. Após esse contato e analisado os resultados da aplicação do octalysis os objetivos de negócios da gamificação tomaram forma.
2. **Delimitar comportamento alvo:** Levando em consideração os objetivos de negócios essa etapa teve como função delimitar como esses objetivos se comportaram dentro do ambiente. Desta forma foi definido como os jogadores interagiram com o ambiente, e o que isso causou;
3. **Descrever jogadores:** Nessa etapa foram estabelecidos os perfis e as características dos jogadores. Por causa do contexto de desenvolvimento de software foram adotados nomes de níveis de experiência desse ambiente, como Desenvolvedor Junior e Desenvolvedor Sênior;
4. **Planejar ciclo de atividades:** Levando em consideração esse contexto, o presente trabalho considerou as escadas de progressão, pelo fato dos níveis mostrar exatamente como está o progresso do jogador;
5. **Garantir diversão:** Nessa etapa deve ser observado se o que está sendo desenvolvido realmente pode trazer diversão para os jogadores. Com isso foi analisado o desenvolvimento de *software*, que por sua vez pode ser dito como algo rígido, até mesmo burocrático. Logo após essa análise deve ser encontrado os elementos de jogos que se encaixam nesse contexto, e com isso, verificar se esses elementos podem realmente representar algo divertido para os jogadores; e
6. **Implementar ferramentas apropriadas:** Sendo a etapa final do *Framework D6*, foi trabalhado tudo aquilo que foi passado nas etapas anteriores, e trazer como resultado um *Game Design*.

3.3.1. Etapa de implementação

Esta etapa foi dividida em duas, sendo elas *Backend* e *Frontend*. Onde o *backend* foi construído utilizando Django e é responsável por guardar os dados dos jogadores, e o *frontend* foi construído utilizando Angular e é responsável por prover uma interface para o que o jogador interaja.

4 RESULTADO E DISCUSSÃO

A presente seção consiste na criação da gamificação e da plataforma do DevMaster, e para melhor organização ela foi dividida em 3 partes, sendo elas: Compreensão do contexto, projeto e plataforma desenvolvida.

4.1. COMPREENSÃO DO CONTEXTO

Como foi descrito na seção 3.2.1 esta etapa foi a responsável por gerar os dados do Octalysis. Assim, foi dividida em 3 seções, sendo elas:

4.1.1. Questionário

Cada pergunta do questionário teve como base uma relação com um determinado core drive, logo, foi feito uma análise dos core drives e após essa análise, o questionário foi elaborado. O questionário do presente trabalho tem dezesseis perguntas, sendo duas para cada core drive. A escolha dessa quantidade de perguntas por core drive se deu pelo fato de melhor quantificar a pontuação de cada um, ou seja, apenas para melhor resolução da pontuação de cada core drive.

Tabela 1 – Questionário

Pergunta	Core Drive
Com que frequência disponibiliza no github código feito por você para ajudar a comunidade?	<i>Meaning</i>
O quanto você acredita que suas skills podem ajudar no desenvolvimento de uma nova funcionalidade desafiadora no seu ambiente de trabalho?	<i>Meaning</i>
Ao aprender uma nova linguagem de programação, geralmente você encontra dificuldade na execução de determinados comandos, o quão motivado a continuar você se sente quando se depara com essas dificuldades?	<i>Accomplishment</i>
Se o seu código não compila, e você já tentou de tudo, qual a chance de você passar essa demanda para outra pessoa?	<i>Accomplishment</i>
O quão você gosta de projetos que são mais necessários a sua criatividade do que habilidades de desenvolvimento?	<i>Empowerment</i>
O quanto você gosta de usar suas skills de desenvolvimento juntamente com sua criatividade para resolver problemas presentes no determinado projeto?	<i>Empowerment</i>
O quanto você gosta de recompensas por ter desenvolvido algo?	<i>Ownership</i>
O quanto você gostaria de acumular essas recompensas?	<i>Ownership</i>
Quanto empenho você coloca em ajudar um companheiro de trabalho com alguma dificuldade que surge no desenvolvimento?	<i>Social influence</i>
O quanto você gosta de trabalhar em equipe?	<i>Social influence</i>
O quanto empenhado você fica em desenvolver uma skill nova?	<i>Scarcity</i>
O quanto você gosta de aprender algo novo, apenas para se manter atualizado com o que é lançado na área da tecnologia?	<i>Scarcity</i>

Qual a chance de você continuar em um projeto com um futuro imprevisível, após alguns erros?	<i>Unpredictability</i>
O quanto você prefere participar de um projeto já consolidado no mercado do que em um ainda não consolidado, mas com potencial, com um futuro que pode dar muito certo ou não?	<i>Unpredictability</i>
Em uma determinada vaga de emprego tem como pré-requisito saber pelo menos o básico (CRUD e sistema de login) no framework Django, você tem dois dias para aprender o básico e ganhar a vaga. Qual a chance de você concorrer a essa vaga sabendo que se não conseguir aprender o básico nesses dias você não conseguirá mais emprego nessa cidade durante algum tempo?	<i>Avoidance</i>
O quão disposto você está a correr riscos?	<i>Avoidance</i>

A tabela 1 mostrou as perguntas do questionário e os core drives que representam essas perguntas.

4.1.2. Pontuação das respostas do questionário

Após a criação do questionário, o mesmo foi disponibilizado para que a Fábrica de Software do CEULP/ULBRA o respondesse.

Para cada questão dentro do questionário foi dada uma resposta de 1 a 5, o que representava a afinidade da pessoa com o tema da questão, questão essa que remetia a um determinado core drive, logo a resposta representava esse core drive.

O Octalysis orienta que a pontuação de cada core drive de ser medida com bastante cuidado, onde a pontuação máxima para cada core drive deve ser apenas 10, com isso em mente, tanto a criação do questionário como o meio de o responder, foram pensados de uma forma a melhor quantificar esse resultado.

Com as respostas do questionário, primeiramente foi realizado a soma das respostas de cada questão, logo depois foi retirada a média desta soma e por fim foi somado as médias. A soma dessas médias mostra a pontuação do *Core Drive*.

Tabela 2 – Pontuação dos Core Drive

Core Drive	Soma das Questões	Pontuação
<i>Social influence</i>	46	9.2
<i>Ownership</i>	45	9.0
<i>Empowerment</i>	44	8.8
<i>Scarcity</i>	40	8.0
<i>Accomplishment</i>	33	6.6
<i>Unpredictability</i>	31	6.2

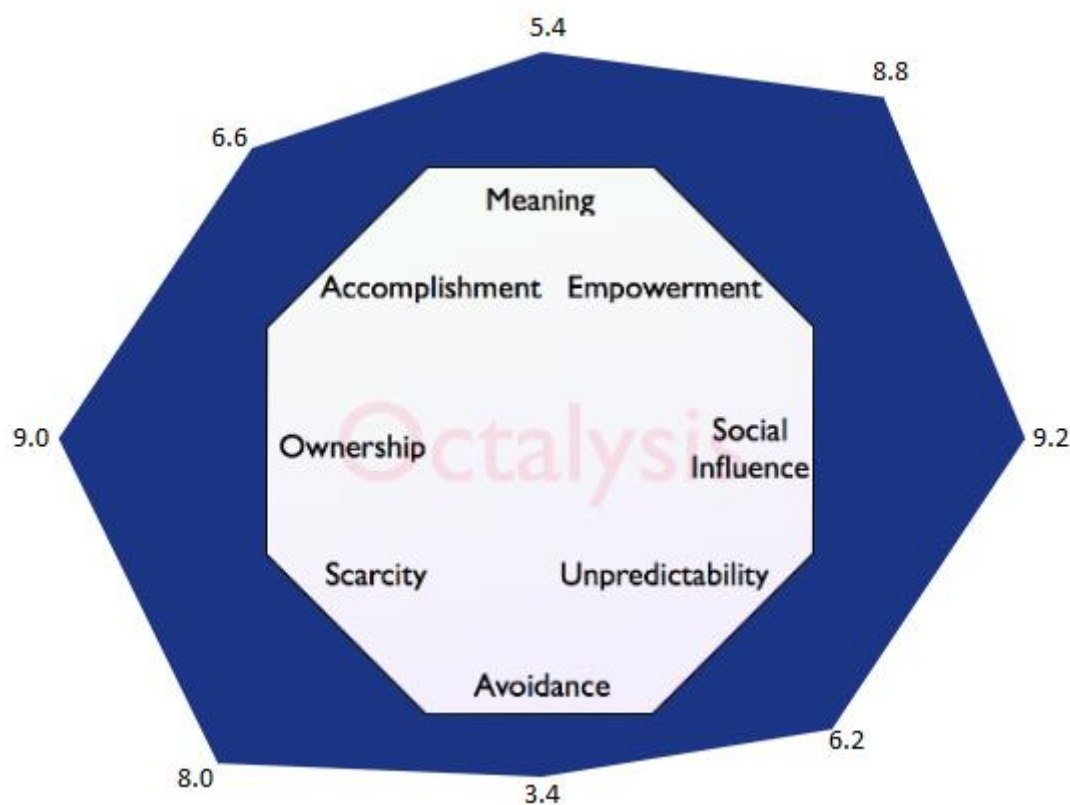
<i>Meaning</i>	27	5.4
<i>Avoidance</i>	17	3.4

A tabela 2 mostra a ordem de como ficou cada core drive. Para o presente trabalho foi adotado *social influence*, *ownership*, *empowerment* e *scarcity* como os core drives mais relevantes dentro do DevMaster, ou seja, o desenvolvimento da gamificação levou em conta esses quatro pontos, é claro que o restante não foi esquecido, porem esses 4 core drive executaram um papel mais importante.

4.1.3. Octacore

Para a criação do octacore foi necessário colocar a pontuação de cada core drive em seu respectivo ponto.

Figura 24: DevMaster Octacore



A Figura 24 apresenta o Octacore do presente trabalho, onde pode-se perceber que os core drive: *Social influence*, *Ownership*, *Scarcity* e *Empowerment*, tiveram as pontuações mais relevante dentro do DevMaster. Logo cada tomada de decisão, em relação a elementos de jogos para compor o DevMaster, foram finalizadas levando em consideração principalmente esses 4 core drives.

4.2. GAME DESIGN

Como foi descrito na seção 3.2.2 esta etapa foi a responsável por aplicar as técnicas do Framework D6.

4.2.1. Objetivo de Negócios

Como resultado da definição dos objetivos de negócios, já pensando em uma forma de unir esses objetivos com a gamificação, foram definidos os seguintes objetivos:

1. Apenas os desenvolvedores da Fábrica de Software do CEULP/ULBRA, que tem acesso ao Gitlab podem criar uma conta, do tipo jogador;
2. O ambiente deve fornecer meios para explorar a convivência social dentro da Fábrica de Software, com o foco em aumentar o companheirismo e a competitividade;
3. O ambiente deve fornecer meios para que os jogadores possam acompanhar seu desenvolvimento, dentro da plataforma, para o restante da equipe;
4. O ambiente deve conseguir manter o jogador empenhado a alcançar um objetivo novo;
5. Apenas o *Scrum Master* da Fábrica de Software deve poder ter uma conta do tipo moderador;

Para medir qual objetivo é mais relevante para o ambiente. É necessário atribuir uma prioridade para cada um, com isso em mente, os objetivos de negócios foram classificados em relação a sua prioridade, da mais baixa até a mais alta.

- Objetivo 1 - Prioridade baixa;
- Objetivo 2 - Prioridade alta;
- Objetivo 3 - Prioridade alta;
- Objetivo 4 - Prioridade alta;
- Objetivo 5 - Prioridade intermediária.

Como os dados usados para a análise são todos providos da Fábrica de Software do CEULP/ULBRA, do uso diário do Gitlab durante o desenvolvimento de software, não teria sentido abrir o jogo para demais usuários, tanto de outras áreas do CEULP/ULBRA quanto do Gitlab. Assim o objetivo 1, apesar da baixa prioridade, não

deixa de ser importante, pois, evita uma falha na medição dos dados referentes ao desenvolvimento.

O objetivo 2 tem como foco gerar uma competitividade amigável entre os jogadores, desse modo, os usuários serão instigados a serem melhores que os companheiros com colocações mais altas no quadro de líderes ou com mais itens do que os outros, assim melhorando o desempenho dos jogadores, e, consequentemente melhorando o desempenho deles dentro do ambiente de trabalho.

A função do objetivo 3 é mostrar para os jogadores seu desenvolvimento dentro da plataforma. Essa função tem um peso muito grande na questão da competitividade (citada no objetivo 2), pois é a partir dela que o jogador consegue acompanhar seu progresso e o progresso dos companheiros.

A função do objetivo 4 é manter o jogador engajado no jogo. Ao analisar alguns jogos, o autor do presente trabalho notou que um dos pontos principais que mantêm um jogador engajado em um jogo *multiplayer* (jogos com multijogadores, ou seja, jogos que permitem mais de um jogador simultaneamente) é a competitividade, ou seja, enquanto tiver competitividade o jogador vai continuar jogando aquele determinado jogo. Deste modo esse objetivo tem como base colocar um propósito para o jogador, e logo após esse propósito ser alcançando, um novo deve ser colocado no lugar, o qual deve levar em consideração a competitividade entre os jogadores.

A intenção do DevMaster é gamificar o processo de desenvolvimento, com isso, o administrador da equipe de desenvolvimento não pode ficar de fora. Com prioridade intermediária o objetivo 5 teve função de monitorar todo o ambiente e configurar determinadas variáveis de ambientes, descritas na seção 4.2.2.

Pode-se afirmar que o objetivo 2 é o objetivo principal do presente trabalho, teve como foco explorar o core drive de *social influence*, que é o core drive com a maior pontuação. Apesar do objetivo 3 e 4 serem formas de desenvolver o objetivo 2, não deixam de ser extremamente importantes, sendo o objetivo 3 para focar no core drive de *empowerment* e o objetivo 4 para focar no core drive de *scarcity*.

4.2.2. Delimitar Comportamento Alvo

Com a conclusão dos objetivos de negócios, foi realizado a definição do comportamento dentro da plataforma, para atingir esses objetivos. De forma a ficar mais didático, o comportamento alvo foi dividido em cinco, sendo eles: Jogador, desafio e *party*, *dashboard* e moderação.

4.2.2.1. Jogador

Apenas os desenvolvedores da fábrica de software do CEULP/ULBRA podem realizar o cadastro, logo, pessoas que não estão cadastradas no grupo de desenvolvimento da fábrica no Gitlab, não terão acesso ao DevMaster.

4.2.2.2. Desafio e *party*

Os jogadores poderão desafiar outros jogadores. Um desafio contém três requisitos, sendo eles: Nome do desafio, jogadores participantes e itens (do DevMaster). Onde cada jogador entrega um item de sua escolha, e o vencedor ganha tudo.

As *party* são grupos de jogadores designados a realizar determinada missão.

4.2.2.3. *Dashboard*

O *dashboard* é o local onde o jogador verifica seu progresso e o progresso dos outros jogadores da equipe. Do mesmo modo verificando os *rankings* dentre eles.

4.2.2.4. Moderação

A área de moderação no DevMaster é guardada para o *scrum master* da fábrica de software do CEULP/ULBRA. Como moderador é possível realizar eventos dentro da plataforma, eventos como por exemplo: 2 vezes mais pontos de experiência ou torneios (com recompensas).

4.2.3. Descrever Jogador

Para o desenvolvimento da gamificação é necessário criar um perfil para os jogadores, e como o DevMaster está focado no desenvolvimento de *software*, e, na metodologia de desenvolvimento *Scrum*, nada mais justo do que eles terem dois perfis, que são desenvolvedores e moderador.

Os desenvolvedores foram o time *scrum*, que nada mais é do que os integrantes da fábrica de software do CEULP/ULBRA que tenham perfil no Gitlab. O moderador foi o *scrum master*, ele é responsável por adicionar variáveis de ambiente, como foi descrito da seção 4.2.2.4.

4.2.4. Ciclo de Atividades

Para o presente trabalho foram planejados três ciclos de atividades. Os ciclos são: Desafios, missões e eventos. Dentre esses ciclos não existe um principal, pois cada um completa o outro, e os três completa os objetivos de negócios (seção 4.2.1.).

4.2.4.1. Desafios

Os desafios são realizados entre os jogadores. Um desafio tem quatro etapas: proposta, aposta, andamento e conclusão.

A proposta é onde um jogador desafia outro, é claro que os dois jogadores devem concordar para que o desafio ocorra, e é necessário que os jogadores participem do ambiente de jogo.

A aposta é logo após a “proposta”, assim que o jogador aceita o desafio é necessário colocar algo em jogo, o DevMaster vai disponibilizar (após a conclusão de alguns eventos, seção 4.2.4.3.) para os jogadores determinados itens, e são esses itens que vão ser colocados em jogo, logo o ganhador do desafio, ganha o item apostado. Não é obrigatório que seja apostado algo, dessa forma os jogadores podem realizar um desafio apenas para contabilizar no seus status.

A fase de andamento é onde os jogadores devem finalizar suas missões para ganhar XP. E a fase de conclusão é onde ocorre a conclusão do desafio.

4.2.4.2. Missões

O Gitlab tem várias características, dentre elas, foram escolhidas as *issues* e as *milestones* para retratar esse ciclo.

As *issues* são as atividades que um desenvolvedor tem que realizar no ambiente de trabalho, logo pode ser dito que essa *issue* é uma missão de um determinado desenvolvedor. É claro que as *issues* tem atributos como o dono, o tempo de conclusão, e a complexidade, deste modo apenas um único jogador pode finalizar determinada *issue*, e apenas esse mesmo jogador recebera a experiencia.

As *milestones* são um agrupamento de *issues*, no presente trabalho elas são tratadas como missões em grupo. Assim dando aos jogadores a experiencia de jogar em grupo, deste modo, ajudando um ao outro na conquista de um objetivo em comum.

4.2.4.3. Eventos

Os eventos são realizados pelo moderador do jogo. O objetivo dos eventos é acrescentar uma variável de imprevisibilidade juntamente como competitividade ao jogo. Imprevisibilidade no sentido que os jogadores não vão saber quando ocorrerá um evento, ou, quando esse evento vai terminar. No caso da competitividade é que, os jogadores que tiverem, mais pontos em um evento, vão ganhar as melhores recompensas, logo eles vão ser instigados a serem mais competitivos.

4.2.5. Garantir Diversão

Graças ao desenvolvimento do Octalysis e a criação do *Octacore*, todos os elementos existentes no presente trabalho foram pensados levando em consideração o *Octacore*, logo, foi levado em consideração as preferencias do time de desenvolvimento da fábrica de *software* do CEULP/ULBRA. Dentre eles pode ser colocado como foco os seguintes elementos:

- Competitividade
- Níveis
- Acumulo de riqueza

A competitividade serviu como uma forma de não deixar o jogo monótono. Por meio dela, os jogadores vão ser instigados a continuar jogando e a melhorar sempre, assim os mantendo sempre focando no jogo.

Os níveis, além das próprias recompensas (de missões ou eventos), vão ser responsáveis por premiar os jogadores por seus desempenhos conforme o andamento de suas atividades, além de mostrar o esforço do jogador em forma de níveis, em comparação aos outros jogadores.

O core drive de *ownership*, que retrata a vontade da pessoa de acumular riqueza, foi o que teve a segunda maior nota no *Octacore*, logo é algo essencial no DevMaster, pois retrata exatamente o que os integrantes da frabrica de software querem.

4.2.6. Implementar Ferramentas Apropriadas

A fábrica de software do CEULP/ULBRA tem uma metodologia de desenvolvimento muito bem definida, com reuniões, análise de requisitos, *product backlog*, *sprint*, e vários outros elementos. Porém a premissa do DevMaster é gamificar um ambiente de desenvolvimento de software que use o *Scrum* e adote o Gitlab como repositório de código fonte. Logo os elementos utilizados para gamificar esse ambiente, deve ser uma junção desses dois requisitos.

Levando isso em consideração temos as *issues* e as *milestones* do Gitlab. As *issues* são tarefas, que podem ser atribuídas a um determinado desenvolvedor (ou integrante do Gitlab da fábrica de software), além disso, também é possível atribuir metas a uma *issue*, dando o tempo que vai ser gasto com ela e até que dia que essa *issue* será finalizada. As *milestones* são agrupamento de *issues*, onde pode ser atribuída várias *issues* a uma *milestone*. Lembrando que uma *issue* não necessariamente vai pertencer a uma *milestone*.

Relacionando essa informação com o *Scrum*, a fábrica de software considera que as *issues* são parte do *product backlog*, e as *milestones* são as *sprints backlog*. Onde uma *sprint* deve ter duração de 1 semana, logo se esse prazo não for atendido, é tido que ocorreu uma falha na organização do projeto.

Em vista disso as *issues* e as *milestones* foram os elementos de desenvolvimento de software escolhidos para a gamificação no presente trabalho. Na parte do *Scrum* o que foi escolhido foi o *burndown chart* (além do *product backlog* e da *sprint backlog*, que foram representadas pelas *issues* e as *milestones*).

4.3.6.1. Elementos de jogos

A escolha dos elementos de jogos do presente trabalho, se deu através da análise dos dados do Octalysis e das técnicas adotadas pela fábrica de software no desenvolvimento. Os elementos de jogos foram organizados da forma como Werbach e Hunter (2012) definiram, em forma de pirâmide:

- Dinâmicas
 - Relacionamento;
 - Narrativa;
 - Emoções;
 - Progressão.

- Mecânica
 - Desafio
 - Competição e cooperação;
 - Feedback;
 - Recompensa;
 - Turno;
 - Estado de vitória.
- Componentes
 - Coleções;
 - Combate;
 - Missões;
 - Quadro de líderes;
 - Níveis;
 - Pontos;
 - Gráfico social;
 - Equipes;
 - Bens virtuais.

As seções seguintes foram criadas com o objetivo de melhor explicar como os elementos de jogos, escolhidos para compor o presente trabalho, foram utilizados. Porém os elementos referentes às dinâmicas, por serem mais abstratos, foram citados em cada seção que fez menção a ele.

Perfil (Progressão, Feedback, Níveis e Pontos)

O perfil é a junção dos elementos de progressão, feedback, níveis e pontos. Onde os pontos foram definidos como pontos de experiência (XP), e o níveis foram expostos como uma forma de medir o progresso do jogador em relação aos pontos. Como mostra a tabela 3.

Tabela 3 – Pontuação dos *Core Drive*

Nível	XP Necessária
0	0 xp
1	30 xp
2	60 xp
3	120 xp

4	240 xp
5	480 xp
6	960 xp
7	1920 xp
8	3840 xp
9	7680 xp
10	15360 xp

De modo geral os pontos foram vistos como o progresso do jogador. Já os níveis foram criados para referenciar em que estágio está o jogador, ou seja, em que determinado nível os pontos desse jogador está, no qual quanto maior o nível/pontos, maior o progresso do jogador. Os níveis foram criados usando uma fórmula de PG (progressão geométrica) com razão 2, onde o número inicial é 30.

Todos esses dados foram disponibilizados para que o jogador pudesse acompanhar, assim, tornando possível a utilização do elemento feedback.

Missões solo e em Grupo (Relacionamento, cooperação, Recompensa, Missões e Equipes)

As missões, em geral, foram criadas com base nas *issues*. Na qual as missões solos são as *issues* que não pertencem a nenhuma *milestone*, já as missões em grupo, são a *issues* que pertencem a uma *milestone*.

No presente trabalho, a única forma de um jogador conseguir XP é na finalização de uma *issue* ou *milestone*. Para que uma *issue* seja qualificada como uma missão no DevMaster é necessário que a mesma tenha 8 atributos, sendo eles:

- **Nome:** Nome da *issue*, que também é o nome da missão;
- **Responsável:** É o usuário do Gitlab responsável por finalizar a *issue*. Este usuário é o mesmo jogador do DevMaster, logo quando for criado a missão do DevMaster, essa, será atribuída a esse jogador;
- **Estado:** É o estado da *issue*, no Gitlab ela tem dois estados, sendo eles “opened” e “closed”. Apenas as *issues* que tem o estado “opened” vão virar missões.
- **Milestone:** É a *milestone* dessa *issue*, esse campo que vai definir se é uma missão em grupo ou não. Do modo que apenas as *issues* que tiver algo nesse campo vão virar missões em grupo.

- **Data de vencimento:** Essa é a data que foi dada para que essa *issue* dure. Ou seja, essa *issue* deve ser finalizada até o final desta data.
- **Data de finalização:** Essa é data que a *issue* foi finalizada, inicialmente esse campo recebe um valor nulo, apenas quando a *issue* é finalizada esse campo é atualizado.
- **Tempo estimado:** Este é o tempo que foi estimado para realizar essa determinada *issue*.
- **Tempo gasto:** Este é o tempo que foi gasto para realizar a *issue*. Esse campo recebe nulo inicialmente, apenas quando a *issue* fecha que ele é alterado.

Após a criação da missão, é necessário quantificar a XP que o jogador pode ganhar com essa determinada missão, sendo que a missão só será quantificada quando for finalizada no Gitlab, como mostra na tabela 4.

Tabela 4 – Quantificação em uma Missão.

Finalizar <i>Issue</i>	XP ganha
Valor da Missão	+120 xp
Tempo gasto > Tempo estimado	-50 xp
Data de finalização > Data de Vencimento	-20 xp

Após a finalização da *issue* o jogador responsável por ela ganha a XP como recompensa, independente se essa missão é uma missão solo ou em grupo.

A verificação do tempo estimado e da data de finalização é um ponto extremamente importante, tanto para o jogador quanto para o desenvolvimento de software, pois uma boa estimativa desses pontos faz com que a etapa de planejamento do desenvolvimento de software se torne mais precisa, deste modo ao planejar uma *sprint* o tempo que vai gasto com ela, será mais acurado, logo o tempo, que é um fator muito importante no desenvolvimento de software, não é desperdiçado. Já o jogador tentará estimar mais precisamente para ganhar mais XP.

Tabela 5 – Quantificação das Missões em Grupo.

XP Ganha
(Missões x 10) xp

Uma missão em grupo só é fechada quando todas as missões que pertencem a esse grupo forem fechadas. A tabela 3 mostra exatamente o que é entregue ao jogador

como recompensa, de modo geral a recompensa por uma missão em grupo é a quantidade de missões, que pertence a esse grupo, multiplicada por 10.

De modo a haver diversas missões, cada uma com um jogador responsável, uma missão em grupo tem mais de um jogador encarregado por ela, dessa forma criando a ideia de grupos cooperativos para a realização da mesma.

Evento (Competição, Recompensa, Estado de vitória, Bens virtuais e Combate)

De maneira geral, um evento pode ser um acontecimento pré-definido com ações determinadas para pessoas específicas. No DevMaster, um evento é algo criado para os jogadores, com o objetivo não só de incentivar a produtividade, mas também os core drives que representam a presente gamificação.

Um evento só pode ser criado pelo moderador da plataforma, onde foi definido que seria o *Scrum Master* da fábrica de software. Dessa forma tanto jogador como moderador interagem um com o outro. Um evento é caracterizado por 5 atributos, sendo elas:

- **Nome:** Aqui é onde o moderador dá um nome para o evento em questão;
- **Multiplicador de XP:** Esse atributo é responsável por mostrar o quanto XP a mais esse evento vai gerar, na qual, o valor apontado por esse atributo multiplica toda a XP que o jogador ganha, durante o tempo em que o evento esteja ativo.
- **Data de início:** A data de início é o que mostra o dia que o evento iniciou.
- **Data de término:** A data de termino mostra o dia que o evento finalizou.
- **Premiação:** Um evento pode ter até 6 premiados. Com o decorrer do evento, os jogadores acumulam pontos, em forma de XP, no qual ao final do evento os jogadores que estão melhor colocados, no ranking do evento, ganham as premiações. Um moderador poder definir uma premiação tanto para 1 jogador quanto para 6 jogadores, ou até mesmo pode não haver premiação.

Os prêmios de um evento são tratados como itens, de modo que esses itens podem ser qualquer coisa fora da plataforma, é claro que tudo vai depender da criatividade do moderador ao criar os itens. Em vista disso, juntamente com o professor Jackson Gomes, foram definidos alguns itens, como aponta a tabela 6.

Tabela 6 – Itens do DevMaster.

Nome	Descrição
------	-----------

Hora do café!	O jogador que usar esse item tem o poder de escolher um dos companheiros do time para fazer um café para equipe.
Estou cansado	O jogador que usar esse item tem o poder de descansar por 15 minutos.
Estou bastante cansado	O jogador que usar esse item tem o poder de descansar por 30 minutos.
Sem tempo irmão	O jogador que usar esse item tem o poder de passar a sua vez do atendimento da fábrica de software para o próximo companheiro da lista.
<i>Leon Help-me</i>	O jogador que usar esse item tem o poder de convocar a ajuda de até 2 companheiros da equipe para auxiliá-lo em algum problema decorrente do desenvolvimento de software.
Dibrei	O jogador que usar esse item tem o poder de anular qualquer efeito de outro item que possa atingi-lo.
NO SPOILER	O jogador que usar esse item tem o poder de ficar imune por até 2 meses sem receber <i>spoiler</i> de alguma série/filme/anime/mangá que outro integrante acompanha.
Vale coxinha	O jogador que usar esse item tem o direito de ganhar um lanche de todos os integrantes da equipe de desenvolvimento.
Joker	O uso desse item faz com que o jogador possa usar qualquer feito dos itens descritos acima.

O ganho desses itens depende primeiramente do moderador, pois ele que decidirá a premiação do evento. Os itens estão presentes na gamificação como forma de interação entre os jogadores, de modo a instigar a competitividade e o acúmulo de riqueza, da forma que um item tem vantagens que podem influenciar a vida real, logo, o ganho deles pode trazer benefícios para o jogador.

Após o moderador decidir as premiações do evento, o ganho desses itens depende apenas dos próprios jogadores, onde eles precisam ganhar o máximo de XP, durante o evento, para ao final ganhar determinado item. Como foi descrito anteriormente, a premiação ocorrerá por meio do ranking do evento, logo, os jogadores com mais pontos, serão os vitoriosos.

Para que um jogador “use” um item, é necessário que ele informe ao moderador, dessa forma o moderador irá remover o item do jogador e o jogador ganha os benefícios do determinado item.

Desafio (Emoções, Desafio, Competição, Turno, Estado de vitória, Gráfico social e combate)

O desafio é uma disputa entre dois jogadores dentro da plataforma. O foco desse elemento é aumentar a competitividade. Um desafio só pode ser criado entre jogadores, ou seja, o moderador não pode intervir em nenhum desafio, ou desafiar algum jogador. O que caracteriza um desafio são seus 8 atributos:

- **Nome:** Esse atributo é o que leva o nome do desafio, e ele é escolhido pelo jogador que cria o desafio.
- **Desafiante:** O jogador que cria o desafio automaticamente vira o desafiante, logo esse atributo refere ao jogador que está desafiando outro jogador.
- **Desafiado:** Para que seja um desafio, o desafiante deve escolher outro jogador para ser desafiado por ele, logo esse atributo refere ao jogador desafiado.
- **Status:** Esse atributo é o que determina em que etapa está o desafio, inicialmente é dado a etapa de “proposta”, porém existe mais 3 etapas (aposta, andamento e conclusão) que serão explicadas posteriormente.
- **Item do Desafiante:** Esse atributo caracteriza o item que o desafiante apostou no desafio. É claro que não é obrigatório que seja apostado um determinado item, esse campo pode receber um item ou não receber nada.
- **Item do Desafiado:** Esse atributo caracteriza o item que o desafiado apostou no desafio. Assim como no item do desafiante, esse campo pode receber um item ou não independentemente se o desafiante apostar um item não apostar nada.
- **Missões:** Tanto o desafiante quanto o desafiado precisa adicionar as suas missões ao desafio, sendo a missão o único meio de medir a pontuação (XP) do jogador nada mais lógico do que também será utilizada para medir a pontuação do jogador referente ao desafio.
- **Vencedor:** Após a conclusão das missões referentes ao desafio, os jogadores devem finalizar o desafio, logo aquele que recebeu a maior pontuação será o vencedor e vai receber os itens apostado pelo outro jogador.

Para melhor entendimento do fluxo de um desafio essa seção foi dividida em suas 4 etapas, onde:

Proposta

É o ato de um jogador poder visualizar outro e o desafiar, mas, para isso ele deve fornecer o nome do desafio, o jogador desafiado e se ele deseja apostar um item ou não. Após criado o desafio, o “desafiado” deve aceitar ou recusá-lo, de modo que para

aceitar ele não precisa adicionar um item, é claro que se o desafio vai ter um item apostado ou não, cabe aos dois jogadores decidirem.

Aposta

Logo após o desafiado aceitar o desafio, chega a etapa de aposta, onde os jogadores devem adicionar suas respectivas missões. Assim como na etapa de proposta, os jogadores devem entrar em um consenso para decidirem quais missões vão adicionar ao desafio, assim para não haver queixas em nenhum dos lados. Apenas as missões abertas podem ser colocadas na aposta.

Vale lembrar, que se os jogadores apostaram algum item (na etapa de proposta), quando entram nessa etapa os jogadores não podem mais remover os itens.

Andamento

Após a escolha das missões, os jogadores devem trabalhar e concluir suas missões nos tempos previsto, para adquirirem a pontuação máxima da missão e assim conseguir vencer o desafio.

Conclusão

Após finalizada as missões, o jogador que obteve a maior pontuação com elas será o vencedor do desafio, e assim ganhará o item apostado pelo o outro jogador (se houver item apostado). Também existe a possibilidade do empate, logo se houver empate, nenhum dos jogadores perdem seus itens.

***Ranking* (Relacionamento, Emoções, Competição, Feedback, Estado de vitória, Combate e Quadro de líderes)**

O *ranking* no presente trabalho é uma forma de relacionar todos os jogadores, onde é possível ver a colocação de um determinado jogador, de modo a incentivá-los a buscar um resultado melhor dentro do ambiente. Para isso foi decidido a criação de 4 rankings, sendo eles:

- ***Ranking de XP:*** Esse pode ser tido como o *ranking* principal, que mostra toda a experiência do jogador dentro da plataforma, independente do jogador ter perdido um desafio, ou não participado de um evento por exemplo.
- ***Ranking de Missões Concluídas:*** Esse *ranking* aponta quantas missões determinado jogador conseguiu finalizar, que também é uma forma de medir quantos itens do *product backlog* o jogador conseguiu fazer.

- **Ranking de Desafios ganhos:** Esse *ranking* aponta quantos desafios os jogadores ganharam.
- **Ranking de Eventos ganhos:** Esse *ranking* aponta quantos eventos os jogadores conseguiram ficar em primeiro.

Os *rankings* são ordenados do maior para o menor, logo o primeiro do ranking é o que tem os status com maior quantidade de pontos (XP, missões fechadas, desafios e eventos ganhos). O objetivo central dos *rankings* é promover a competição em busca do primeiro lugar, eles não zeram, ou seja, continuam conforme o jogador vai crescendo no jogo.

Mochila (Feedback, Recompensa, Coleções e Bens virtuais)

Fora do mundo digital, uma mochila é um acessório para transportar determinados itens. No presente trabalho a mochila é exatamente a mesma coisa, cada jogador possui uma, e dentro dela estão os itens do jogador, que apenas o dono pode ver, entretanto como diferencial, a capacidade de armazenamento de uma mochila é ilimitada, ou seja, pode armazenar quantos itens o jogador conseguir obter.

Dessa forma o jogador consegue acumular seus bens, e assim formar sua coleção de itens.

Burndown Chart (Narrativa)

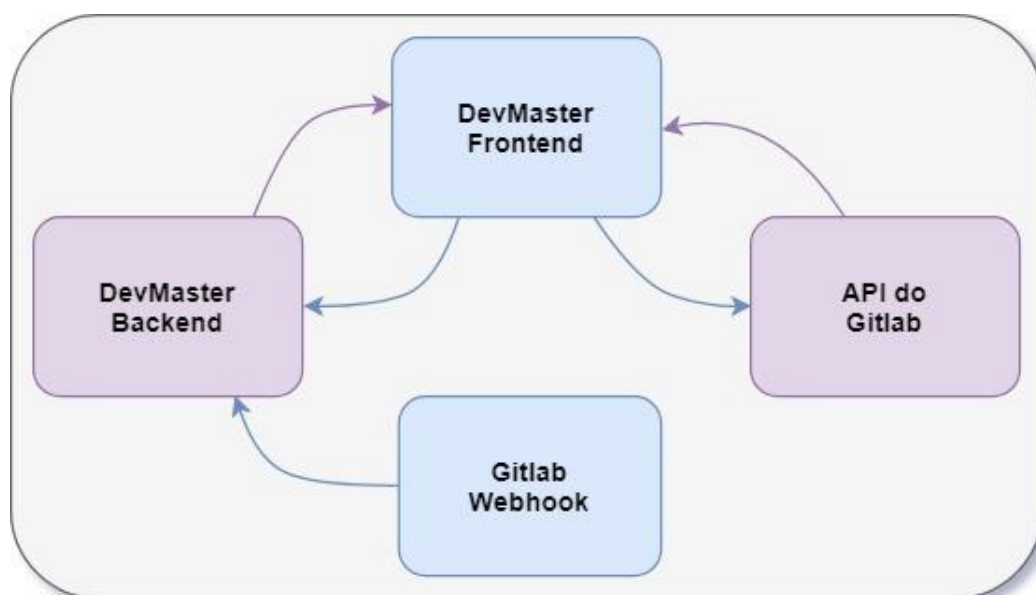
O Burndown chart é um artefato do *scrum*, que mostra o progresso da *sprint* no decorrer da semana. Onde é apresentado a linha ideal do gráfico e o progresso do que está sendo feito, logo, apontando se a equipe estimou bem as tarefas e se não vai ocorrer atraso no que foi planejado.

No Devmaster os jogadores podem ver esse gráfico, divididos por semanas, dessa forma eles podem verificar o progresso da semana atual e o das semanas passadas. O intuito desses gráficos é mostrar para os jogadores como anda o progresso da equipe, dessa forma mostrando que tudo que eles fazem tem um efeito evidente dentro da plataforma e no desenvolvimento de software em si.

4.3. PLATAFORMA DESENVOLVIDA

A plataforma desenvolvida no presente trabalho teve como resultado duas partes, sendo elas o *backend* e o *frontend*. Deve-se deixar claro que o *backend* do presente trabalho foi construído em forma de uma API HTTP REST, que trabalha através de métodos HTTP (POST, GET, PUT E DELETE), onde ficou responsável por receber requisições, das quais serão retornas respostas referente ao que foi requisitado. O *frontend* ficou responsável por consumir os serviços tanto do *backend* quanto da API do Gitlab, além de prover uma interface de interação para o usuário. A Figura 25 mostra exatamente como ocorre essas ligações e como foi definida a estrutura do sistema.

Figura 25: Estrutura do Sistema.



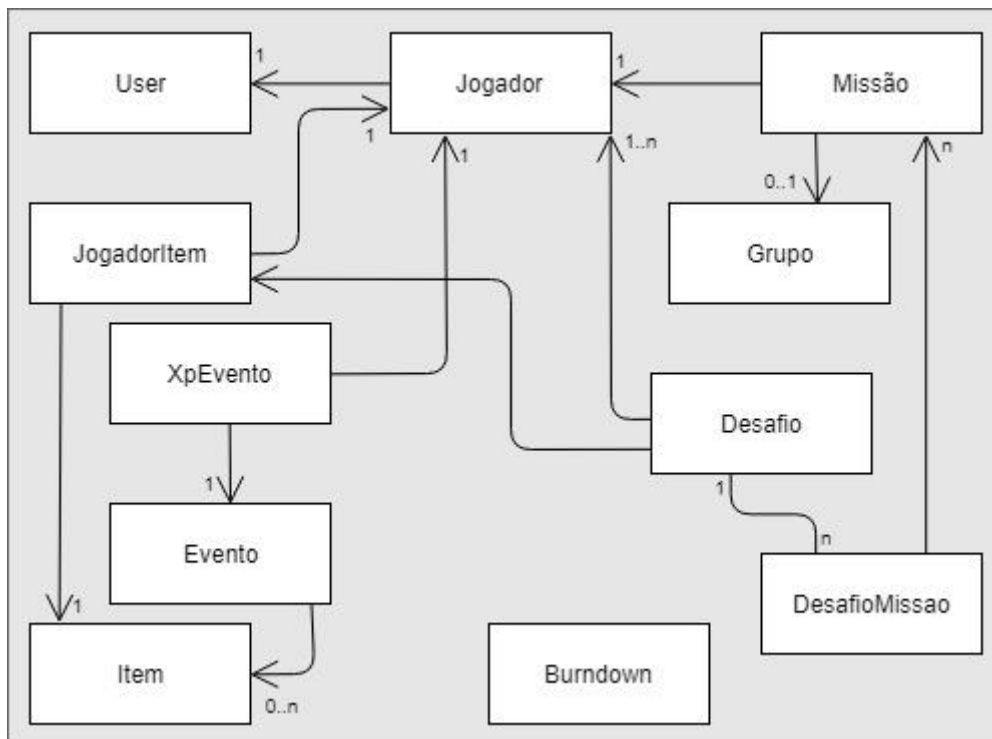
Para ficar mais claro, as setas azuis da Figura 25 referem as requisições realizadas as determinadas API's, e as setas roxas referem as respostas dessas requisições. O Gitlab também conta com um sistema de *Webhooks*, que de modo geral, é um gatilho, explicado posteriormente (seção 4.3.1.2.), onde se houver determina alteração no Gitlab, é realizada uma requisição POST, com os dados dessa alteração para uma determina url.

4.3.1. Backend

Como forma de estruturação do *backend*, foi criado um diagrama de classes (Figura 26), vale lembrar que esse diagrama de classes foi adaptado para mostrar apenas

a ideia geral dos relacionamentos de cada classe dentro do presente trabalho, de modo que ao decorrer do trabalho foi mostrado as classes mais importantes, e seus atributos.

Figura 26: Diagrama de classes do DevMaster.



A classe User é referente ao usuário do Django, a função dela é prover ao usuário campos para que ele se autentique, ou seja, essa é classe é usada para verificar quem é o usuário, ela é composta vários atributos, porém os apenas 5 são relevantes no DevMaster. Sendo eles:

- **Username:** Esse atributo refere ao nome que o usuário utiliza para logar. Foi definido que esse *username* deve ser o mesmo que os desenvolvedores usam no Gitlab, assim facilitando a sua busca no API do Gitlab.
- **First_name:** Esse atributo é o primeiro nome do usuário.
- **Last_name:** Esse atributo é o sobrenome do usuário.
- **Password:** Esse atributo é a senha que o usuário usa para o login. O django salve essa senha já encriptada, de modo que nem o administrador do sistema consiga saber qual senha é.
- **User_permissions:** Esse atributo é o que aponta quais permissões, no Django admin (seção 4.3.2.8.), determinado usuário tem.

A Figura 27 apresenta a classe do jogador, que é uma das classes mais importante de todo o trabalho. De modo geral a classe jogador tem o objetivo de guardar todos os dados gerados pelo jogador dentro da plataforma.

Figura 27: Classe Jogador.



- **User:** Esse campo refere ao User do Django.
- **Tipo:** Esse campo refere ao tipo de jogador. Esse campo é essencial para referenciar ao moderador da plataforma. Logo que um jogador normal vai ter um tipo “jogador” e o moderador um tipo “moderador”, porém, o moderador conseguira ver seu status dentro da plataforma,
- **Xp_total:** Esse campo refere a toda XP que o usuário ganha na plataforma.
- **Desafios_v:** Esse campo refere a todos os desafios que o usuário ganha na plataforma, não vai ser guardado os desafios que o usuário perde, pelo fato de mostrar algo negativo, que pode ser apontado como um fator.
- **Eventos_v:** Esse campo refere a todos os eventos que o usuário ganha na plataforma, da mesma forma com os desafios, também não vai ser guardado os dados que refere a eventos que o jogador não ganhou.
- **M_realizadas:** Esse campo refere a todas as missões realizadas pelo jogador, ou seja, todas as missões que o jogador adquiriu e depois finalizou no Gitlab.

- **M_adquiridas:** Esse campo refere a todas as missões adquiridas pelo jogador, ou seja, todas as missões que ele criou no Gitlab.
- **Mr_nadata:** Esse campo refere a todas as missões que o jogador conseguiu concluir na data correta.
- **Mr_notempo:** Esse campo refere a todas as missões que o jogador conseguiu concluir no tempo correto.
- **Private_token:** Esse campo refere ao *token* que o Gitlab disponibiliza ao jogador, para que possa ser realizada ações dentro de sua API. Esse *token* é uma das formas que possibilita a comunicação com a API do Gitlab.
- **url_image:** Esse campo refere a url da imagem do jogador. A escolha da url da imagem e não da imagem em si, se deu pelo fato de que o usuário ao se cadastrar no DevMaster, é realizada uma requisição a API do Gitlab e é copiada a imagem dele no Gitlab, em forma de url, para esse campo.

Já a classe missão tem o objetivo de referenciar as *issues* do Gitlab. Além de ser classe que possibilita a criação de todo o restante das classes. A Figura 28 apresenta todos os atributos da classe missão.

Figura 28: Classe Missão.

Missão
+ jogador: Jogador
+ nome_missao: string
+ xp_ganha: float
+ xp_missao: float
+ data: date
+ nice_tempo: boolean
+ nice_data: boolean
+ status: boolean
+ id_issue: int
+ id_projeto: int
+ id_milestone: int

Os tópicos a seguir tem o objetivo de relacionar os atributos de uma missão (seção 4.3.6.1.2.) com os dados referente a classe de Missão:

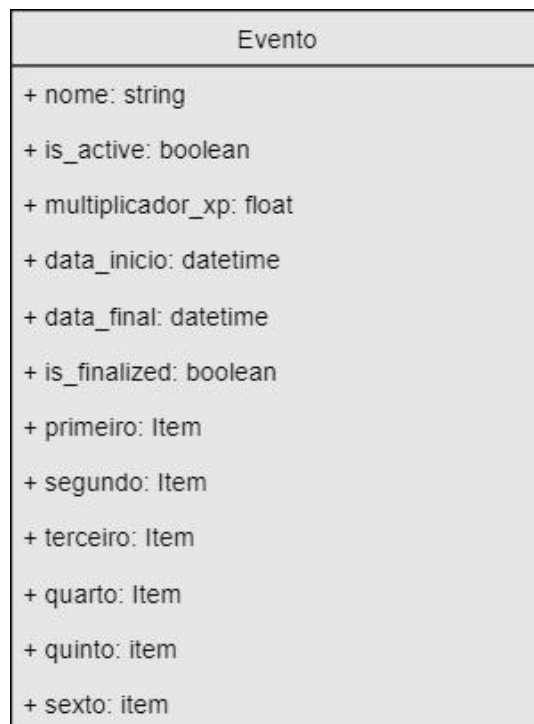
- **Jogador:** Esse campo refere a classe jogador, que é mesmo responsável pela *issue* que representa essa missão no Gitlab.
- **Nome_missao:** Esse campo refere ao nome da missão. Que é o mesmo nome atribuído na hora da criação da *issue*.
- **Xp_ganha:** Quando a missão é finalizada esse campo refere a XP que o jogador ganhou com essa missão.
- **Xp_missao:** Esse campo refere a XP que o jogador pode ganhar com a missão.
- **Data:** Esse campo refere a data em que a missão foi fechada.
- **Nice_tempo:** Esse campo aponta se a missão foi fechada no tempo proposto ou não. Logo se esse campo foi positivo, a XP ganha será maior.
- **Nice_data:** Esse campo aponta se a missão foi fechada na data proposta ou não. Assim como no campo anterior, se esse atributo foi positivo, a XP ganha será maior.
- **Status:** Esse campo é o que representa o estado da *issue*, onde mostra se ela foi fechada ou não.
- **Id_issue:** Esse campo refere ao identificador da *issue* no Gitlab.
- **Id_projeto:** Esse campo refere ao identificador do projeto dessa *issue* no Gitlab.
- **Id_milestone:** Esse campo refere ao identificador da *milestone* que essa missão/*issue* possa ter. Logo esse campo vai indicar a qual grupo essa missão pertence. Lembrando que esse campo pode receber um valor nulo, que vai representar que essa *issue* não tem nenhuma *milestone*, que por sua vez, representa que a missão referente a essa *issue* não tem nenhum grupo.

A classe Grupo foi criada com 4 atributos, sendo eles: nome, xp_grupo, status, id_milestone e participantes. Onde o campo nome refere ao nome do grupo, o campo status aponta se esse grupo teve as suas missões finalizadas ou não, o campo id_milestone refere as missões que esse grupo possui, e o campo participantes, refere aos participantes desse grupo.

A classe Item tem objetivo de descrever os itens do sistema. Ela tem 3 atributos, sendo eles o nome, a descrição e a url_image. Onde a descrição é o efeito desse item e a url_image é uma imagem que representa o item.

O evento é classe que é responsável pela criação dos eventos na plataforma, do qual é constituído por 12 atributos, como aponta a Figura 29.

Figura 29: Classe Evento.



Os tópicos a seguir tem o objetivo de relacionar os atributos de um evento (seção 4.3.6.1.3.) com os dados referente a classe de Evento:

- **Nome:** Esse é o nome do evento, escolhido na hora em que o moderador o cria.
- **Is_active:** Esse campo aponta se o evento está ativo ou não.
- **Multiplicador_xp:** Esse campo tem o objetivo de adicionar uma variável de ambiente ao jogo, de modo que sempre que um evento esteja ativo, toda a XP que um jogador ganhe, ao realizar uma missão, é multiplicada por esse valor.
- **Data_inicio:** É a data em que o moderador criou o evento.
- **Data_final:** É a data em que o evento é encerrado.
- **Is_finalized:** Esse campo refere as premiações, de modo que se esse campo estiver negativo ainda não foi realizada as premiações do evento. Para que seja realizada as premiações, é necessário que esse campo seja negativo e o campo “is_active” seja positivo, ou seja, apenas quando um evento não está ativo e não for finalizado que é realizada as premiações.

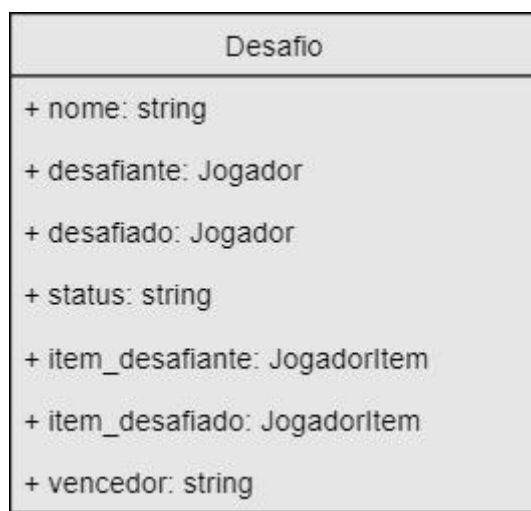
- **Premiação:** Esse campo refere aos atributos do “primeiro” ao “sexto” da Figura 29. Todos esses 6 campos referem a um item que um determinado jogador pode ganhar, vale lembrar que esses campos tanto podem receber um item como também podem não receber nada.

Já a classe de XpEvento é a responsável por guardar a XP que o um determinado jogador ganha durante um determinado evento. Ela possui 3 atributos, sendo eles: jogador, evento e xp_evento. Onde o atributo jogador é a referência ao jogador que é dono dessa XP, e o evento é a referência para qual evento o jogador conseguiu esses pontos. O atributo xp_evento é o responsável por guardar a XP que o jogador ganha quando o evento estava ativo. Logo ao realizar a premiação de um determinado evento será analisado essa classe, onde, os jogadores obtêm os itens referente a sua posição no evento em relação a XP ganha por eles.

A classe JogadorItem tem o objetivo de relacionar um determinado item ao um jogador. De modo que essa classe possui 4 atributos, sendo eles: jogador, item, quantidade, quantidade_bloqueada. Onde jogador é referência a um determinado jogador da plataforma, assim como item é referência a determinado item. A quantidade é o número desse determinado item que esse jogador possui, já a quantidade bloqueada é o número de desse item, que o jogador tem bloqueado, ou seja, que ele não pode usar.

A classe de desafio é o que disponibiliza que um jogador possa desafiar outro. Da qual tem 7 atributos, com mostra Figura 30.

Figura 30: Classe Desafio.

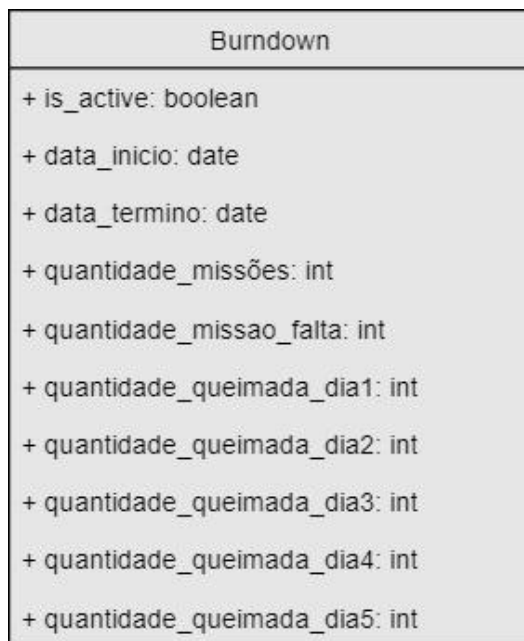


Os tópicos a seguir tem o objetivo de relacionar os atributos de um desafio (seção 4.3.6.1.4.) com os dados referente a classe de Desafio:

- **Nome:** Esse campo refere ao nome dado para o desafio pelo jogador, na hora de sua criação;
- **Desafiante:** Esse campo refere ao jogador que criou o desafio;
- **Desafiado:** Esse campo refere ao jogador que foi desafiado;
- **Status:** Esse campo aponta em que status está o desafio.
- **Item_desafiante:** Esse campo faz referência ao item que o jogador desafiante apostou.
- **Item_desafiado:** Esse campo faz referência ao item que o jogador desafiado apostou. Lembrando tanto esse campo quanto o de “item_desafiante” pode receber um valor nulo.
- **Vencedor:** Esse campo é o que aponta qual jogador venceu o desafio.

Já a classe de DesafioMissão é a que faz referência ao atributo de “missões” da seção 4.3.6.1.4. Essa classe tem 3 atributos, sendo eles: desafio, missão e xp_ganha. Onde o campo missão é o responsável por conter 1 missão, já o campo de desafio é o responsável por conter o desafio escolhido pelo jogador para aquela missão, dessa forma que para cada missão que um jogador adiciona ao DesafioMissão, é criado um objeto da classe que a referência, deste modo um desafio pode ter quantas missões os jogadores decidirem, é claro que não é possível missões repetidas, mas, é possível que dois desafios diferentes tenha as mesma missão. O atributo xp_ganha é responsável por guardar a XP que o jogador dono da missão conseguiu com a determinada missão, logo, quando um desafio é concluído, é somado todos os atributos de xp_ganha, que referente ao desafio concluído e aos jogadores (desafiante e desafiado) participantes do desafio, deste modo o jogador que conseguiu mais XP é o vencedor do desafio e conquistara os prêmios desafiados, se houver.

A classe *Burndown* é a responsável por apresentar os dados referente ao *burndown chart* do *scrum* e possui 10 atributos, como mostra a Figura 31.

Figura 31: Classe de *Burndown*.

- **Is_active:** Esse campo aponta se esse *burndown* está ativo ou não, de modo que apenas os que estiverem desativados, estão com os gráficos completos;
- **Data_inicio:** Esse campo aponta em que data esse *burndown* foi iniciado;
- **Data_termino:** Esse campo aponta em que data esse *burndown* foi encerrado;
- **Quantidade_missoes:** Esse campo aponta a quantidade de missões que com o decorrer da semana devem ser realizadas;
- **Quantidade_missoes_falta:** Esse campo referência a quantidade de missões que não foram realizadas na semana anterior, que deveram ser finalizadas essa semana;
- **Quantidade_queimada (dia1 ao dia5):** Os campos referentes a quantidade queimada, aponta quantas missões foram queimadas no decorrer da semana. Ou seja, quantas missões foram finalizadas pela equipe no decorrer da semana.

4.3.1.1. API HTTP REST

A presente seção, destina-se a apresentação da API REST do DevMaster, com objetivo de exibir seus *endpoints*, tais como suas rotas, seus métodos, as requisições e as respostas. Antes de iniciar essa apresentação, existe dois pontos que devem ser adiantados, que são referentes a utilização de duas bibliotecas, sendo elas o *corshearders* e o *auth_token*.

O *corsheaders* é uma biblioteca que permite que apenas determinados domínios, acessem uma determinada API. Em vista disso, o presente trabalho a utiliza para que apenas dois domínios tenham acesso a API do DevMaster, esses domínios são referentes ao *Frontend* do DevMaster e ao Gitlab da fábrica de *software*, logo qualquer outro domínio é impedido de acessar as informações da API em questão.

A biblioteca *auth_token* foi utilizada para prover o sistema de login, de forma automática, assim, não precisou de uma implementação do *endpoint* de login, pois essa biblioteca disponibiliza esse *endpoint* pronto.

Com isso em mente, as funcionalidades da API do DevMaster foram divididas em 3 grupos, sendo eles: Sem autenticação, *webhook* do Gitlab e com autenticação.

Sem autenticação

Essa seção é referente às funcionalidades que o usuário pode utilizar na API, sem a necessidade de autenticação.

As funcionalidades dessa seção, são:

- Login;
- Cadastro de Jogador.

Essas funcionalidades são descritas em detalhes nas seções a seguir:

Login

Realiza a autenticação do usuário no sistema.

POST /login

Os parâmetros são (corpo da requisição):

Nome	Tipo	Descrição
username	string	O nome de usuário do jogador
password	string	A senha de usuário do jogador

Em caso de sucesso (código 200) a resposta indica que o login foi realizado com sucesso e retorna um objeto com o seguinte atributo:

- *Key*: O *token* de acesso do usuário na plataforma.

Em caso de erro com código 400 a resposta indica que não foi possível encontrar o usuário indicado, ou as credenciais estão incorretas.

Em caso de erro com código 500 a resposta indica que ocorreu um erro de execução no servidor.

Cadastro de jogador

Cadastra um novo jogador no sistema.

POST /criarjogador

Os parâmetros são (corpo da requisição):

Nome	Tipo	Descrição
username	string	O nome de usuário do jogador
password	string	A senha de usuário do jogador
first_name	string	Primeiro nome do jogador
last_name	string	Sobrenome do jogador
email	string	Email do jogador
private_token	string	Token de acesso a API do Gitlab
url_image	string	url da imagem do jogador

Em caso de sucesso (código 200) a resposta indica que o jogador foi cadastrado com sucesso e retorna um objeto com o seguinte atributo:

- *Message*: Sucesso do cadastro.

Em caso de erro com código 500 a resposta indica que ocorreu um erro de execução no servidor.

Webhook do Gilab

Antes de abordar quais requisições são realizadas pelo usuário, deve ser explicado como o sistema de *Webhook* do Gitlab se comunicou com a API do DevMaster. Primeiramente deve ficar explícito que os passos descritos para a realização dessa comunicação foram feitos para cada projeto do Gitlab da fábrica de *software* do CEULP/ULBRA. Em vista disso, foi dividido em 4 passos, sendo eles:

1. Integração: Cada projeto possui suas próprias preferencias, é para configurar o *webhook*, primeiro precisa entrar nas configurações de um projeto, mais

precisamente na opção de integração (url: www.gitlab.com/"nome do grupo"/"projeto"/settings/integrations).

2. URL do *endpoint* e *token*: Após acessar o painel de integração, deve ser definida a rota em que receberá a requisição do Gitlab e o *token* de acesso para essa funcionalidade. Como mostra Figura 32.

Figura 32: URL do *endpoint* e *token*.

URL

{{api_dominio}}/gitlabissue

Secret Token

token para acesso da api

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

3. Gatilho: Agora deve ser escolhido o evento que encadeará uma requisição para a determina API. Como o presente trabalho se trata da utilização das *issues*, apenas elas devem ser colocadas nesse gatilho, logo qualquer alteração que houver em uma *issue* dentro desse projeto, o Gitlab mandará uma requisição. Como aponta a Figura 33

Figura 33: Seleção dos eventos para o gatilho.

- ☒ **Issues events**
This URL will be triggered when an issue is created/updated/merged
- ☒ **Confidential Issues events**
This URL will be triggered when a confidential issue is created/updated/merged

4. Desativar o SSL: Com a ideia de evitar problemas com o protocolo de acesso, deve ser desativado o SSL desse *webhook*, após isso basta apenas clicar em “Add *webhook*” no final da página. Como mostra a Figura 34.

Figura 34: Desativando o SSL.

SSL verification

☐ **Enable SSL verification**

Add webhook

Esse *webhook* realiza uma requisição com o método POST para o determinado *endpoint*. Junto dessa requisição, é mandando um corpo, com os dados referentes ao evento configurado. Em vista disso, é claro que o *endpoint* que deve ser configurado no Gitlab é o do DevMaster. Como a requisição que o Gitlab faz é padrão, esse *endpoint* foi feito para se adaptar aos dados recebidos.

POST /gitlabissue

Os parâmetros são (corpo da requisição):

Nome	Tipo	Descrição
project.id	number	Identificador do projeto daquela <i>issue</i>
object_attributes.id	number	Identificador da <i>issue</i>
object_attributes.assignee_id	number	Identificador do responsável pela <i>issue</i>
assignee_id.milestone_id	number	Identificador da <i>milestone</i> dessa <i>issue</i>
assignee_id.state	string	Indica o estado da <i>issue</i> (<i>opened</i> , <i>cloued</i>)
assignee_id.action	string	Ação realizada no gatilho (criação, edição ou exclusão)

Por se tratar do *webhook* do Gitlab, não é retornado nenhuma mensagem para ele.

Com nesses dados é possível verificar se foi um gatilho para criação, atualização ou exclusão de uma *issue*. Com isso esse *endpoint* deve pegar essa informação e realizar uma dessas 3 ações.

- **Criar Missão:** Se o gatilho foi do tipo criação, uma nova missão é criada e atrelada ao jogador que faz referência ao desenvolvedor da fábrica de software que é responsável por essa *issue*.
- **Atualizar Missão:** Se o gatilho foi do tipo atualização, posse ser realizada apenas 1 ação, sendo ela:
 - **Finalizar Missão:** Se a atualização foi feita para fechar a *issue*, a missão que refere a essa *issue* é fechada, e os pontos de XP dela é computado. Ainda nessa ação deve ser verificado se a missão faz parte de um grupo,

de uma `MissaoDesafio` ou se tem um evento ativo, logo, com essas informações, deve ser feito as atualizações nesses itens.

- **Excluir missão:** Se o gatilho foi do tipo exclusão, a missão referente a essa *issue* deve ser excluída, com a condição de não a ter finalizado ainda. É claro que se a *issue* não existir como missão, não será realizada nenhuma ação.

Com Autenticação

Os *endpoints* das seções anteriores (4.3.1.1.1. e 4.3.1.1.2.) foram utilizados para criar a base em que o jogador possa utilizar as funcionalidades do DevMaster.

Antes de apresentar o restante dos *endpoints* vale lembrar que a partir de agora todos levam no header da requisição, duas informações, assim não será necessário apresentar as mesmas informações em toda requisição. As informações são:

- **Authorization: token token_jogador:** O `token_jogador` referência a “key” que retorna do *endpoint* de login, apenas com ele o usuário consegue ter acesso as funcionalidades, pois ele representa o próprio jogador dentro da API.
- **Content-Type: application/json:** Essa informação referência que todos os dados passados como requisição são do tipo json (*JavaScript Object Notation*).

As funcionalidades da API, que necessitam de autenticação, são:

- User
 - Consultar usuário logado.
- Jogador
 - Listar jogadores;
 - Consultar um jogador;
 - Consultar itens de um jogador;
- Evento
 - Listar eventos;
 - Consultar um evento;
 - Consultar premiação do evento;
 - Gerar premiação do evento.
- Missões
 - Listar todas as missões.

- Grupos
 - Listar grupo;
 - Consultar um grupo;
 - Listar missões de um grupo.
- Desafios
 - Listar desafios;
 - Consultar missões de um desafio;
 - Criar um desafio;
 - Adicionar missão a um desafio;
 - Trocar item do desafio;
 - Alterar status do desafio;
 - Deletar desafio;
 - Remover uma missão de um desafio;
- Burndown
 - Listar *burndowns*;
 - Consultar um *burndown*.

User

Consultar usuário logado

Consulta o usuário referente ao *token* passado.

GET /user

Em caso de sucesso (código 200) retorna um objeto com os atributos:

- `id (string)`: identificador do User;
- `username (string)`: nome do usuário;
- `password (string)`: senha do usuário;
- `first_name (string)`: primeiro nome;
- `last_name (string)`: sobrenome;
- `email (string)`: email do usuário;

Em caso de erro com código 404 a resposta indica que não foi possível encontrar o usuário indicado.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Jogador

Listar jogadores

Lista todos os jogadores da plataforma.

```
GET /jogador
```

Em caso de sucesso (código 200) a resposta é um array de objetos com os mesmos atributos da funcionalidade de cadastro de jogador.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Consultar um jogador

Consulta um jogador específico.

```
GET /jogador/:id
```

O parâmetro de rota `:id` representa o identificador do jogador específico.

Em caso de sucesso (código 200) a resposta é um objeto com os mesmos atributos da funcionalidade de cadastro de jogador.

Em caso do erro com código 500 a resposta indica um erro de execução do servidor.

Consultar itens de um jogador

Lista todos os itens do jogador com o *token* específico.

```
GET /jogadorItens/itens
```

Em caso de sucesso (código 200) a resposta é um *array* de objetos com os atributos:

- Item (object): um objeto de item, com os seguintes atributos:
 - nome (string): nome do item;
 - descricao (string): descrição do item;
 - url_image (string): url da imagem do item.

- `jogador (object)`: é um objeto de jogador, com os mesmos atributos, do corpo da requisição, de cadastro de jogador;
- `quantidade (number)`: quantidade de itens que o jogador tem;
- `quantidade_bloqueada (number)`: quantidade bloqueada desse item.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Evento

Listar eventos

Lista todos os eventos da plataforma.

GET /evento/consultarEventos

Em caso de sucesso (código 200) a resposta é um array de objetos com os mesmos atributos da funcionalidade de consultar um evento.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Consultar um evento

Consultar um determinado evento.

GET /evento/consultarEvento/:id

O parâmetro de rota `:id` representa o identificador do evento, do qual deve ser consultado.

Em caso de sucesso (código 200) a resposta é um objeto com os atributo:

- `nome (string)`: Nome do evento;
- `is_active (boolean)`: indicador se o evento está ativo ou não;
- `multiplicador_xp (number)`: multiplicado da xp ganha pelos usuários no evento;
- `data_inicio (datetime)`: data e hora que o evento iniciou;
- `is_finalized (boolean)`: indicador se a premiação do evento foi entregue ou não;
- `primeiro (object)`: objeto de item, que representa a premiação do primeiro colocado.
- `segundo (object)`: objeto de item, que representa a premiação do segundo colocado.

- terceiro (object): objeto de item, que representa a premiação do terceiro colocado.
- quarto (object): objeto de item, que representa a premiação do quarto colocado.
- quinto (object): objeto de item, que representa a premiação do quinto colocado.
- sexto (object): objeto de item, que representa a premiação do sexto colocado.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Consultar premiação do evento

Lista a XP ganha pelos jogadores em um determinado evento.

```
GET /evento/premiacaoEvento/:id
```

O parâmetro de rota `:id` representa o identificador do evento, do qual deve ser consultado.

Em caso de sucesso (código 200) a resposta é um objeto com os atributos:

- evento (object): objeto referente ao evento;
- jogador (object): objeto referente ao jogador;
- xp_evento (float): Xp ganha pelo jogador nesse evento.

Em caso de erro com código 404 a resposta indica que não foi possível encontrar as XPs do evento indicado.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Gerar premiação do evento

Finaliza um evento e computa as premiações para os jogadores, conforme a XP ganha por cada um dentro do evento.

```
POST /evento/gerarPremiacao/:id
```

O parâmetro de rota `:id` representa o identificador do evento, do qual deve ser premiado.

Essa requisição não precisa de parâmetros.

Em caso de sucesso (código 200) a resposta indica que o evento foi finalizado e retorna um objeto com o atributo:

- `message`: Evento finalizado e premiação gerada!: o identificador da operação

Em caso de erro com código 400, tem a possibilidade de haver uma dessas 4 mensagens:

- Evento não existe;
- Evento ainda está ativo;
- Evento já teve a premiação gerada.

Em caso de erro com código 500 a resposta indica que ocorreu um erro de execução no servidor.

Missões

Listar todas as missões

Lista todas as missões da plataforma.

GET /missao

Em caso de sucesso (código 200) a resposta é uma *array* de objetos com os atributos:

- `id` (number): o identificador da missão;
- `nome_missao` (string): o nome da missão;
- `xp_ganha` (float): XP ganha com a missão, ao ser finalizada;
- `xp_missao` (float): XP que pode ser ganha com a missão;
- `data` (date): Data que a missão deve ser finalizada;
- `nice_tempo` (boolean): indicador que mostra se o tempo que a missão foi concluída foi um bom tempo ou não;
- `nice_data` (boolean): indicador que mostra se a data que a missão foi concluída foi uma boa data ou não;
- `status` (boolean): indicador se a missão foi concluída ou não;
- `id_issue` (number): indicador da *issue* do Gitlab;
- `id_projeto` (number): indicador do projeto do qual essa *issue* pertence;

- `id_milestone (number)`: indicador de qual *milestone* essa *issue* pertence.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Grupos

Listar grupo

Lista todos os grupos da plataforma:

```
GET /grupo/listarGrupos
```

Em caso de sucesso (código 200) retorna um array de objetos com os atributos:

- `name (string)`: nome do grupo em questão;
- `xp_grupo (float)`: XP ganha ao finalizar o grupo;
- `status (boolean)`: Indicador se o grupo está fechado ou não;
- `id_milestone (number)`: Identificador da *milestone* do grupo;

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Consultar um grupo

Consulta um determinado grupo no sistema:

```
GET /grupo/consultarGrupo/:id
```

Em caso de sucesso (código 200) retorna um objeto com os atributos:

- `name (string)`: nome do grupo em questão;
- `xp_grupo (float)`: XP ganha ao finalizar o grupo;
- `status (boolean)`: Indicador se o grupo está fechado ou não;
- `id_milestone (number)`: Identificador da *milestone* do grupo;

Em caso de erro com código 404 a resposta indica que não foi possível encontrar o grupo indicado.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Listar missões de um grupo

Lista todas as missões de um grupo específico:

```
GET /grupo/missoes/:id
```

Em caso de sucesso (código 200) retorna um objeto com os atributos descritos na funcionalidade de listar todas as missões.

Em caso de erro com código 404 a resposta indica que não foi possível encontrar o grupo indicado.

Em caso de erro com código 500 a resposta indica um erro de execução do servidor.

Desafios

Listar desafios

Lista todos os grupos da plataforma:

```
GET /desafio/listarDesafios
```

Em caso de sucesso (código 200) retorna um array de objetos com os atributos:

- **name (string):** nome do desafio;
- **desafiante (object):** jogador que cria o desafio;
- **desafiado (object):** jogador desafiado;
- **status (string):** indicador do estágio do desafio;
- **item_desafiante (object):** item apostado, do jogador que criou o desafio.
- **item_desafiado (object):** item apostado, do jogador que foi desafiado.
- **vencedor (string):** indicador do vencedor do desafio, pode ser: “desafiante” ou “desafiado”

Em caso de erro com código 500 a resposta indica um erro de execução do servidor.

Consultar missões de um desafio

Lista todas as missões de um desafio específico:

```
GET /missoes/consultarMissoesDesafio/:id
```

O parâmetro de rota **:id** representa o identificador do desafio específico, do qual deve ser listado as missões.

Em caso de sucesso (código 200) retorna um array de objetos com os atributos descritos na funcionalidade de listar todas as missões.

Em caso de erro com código 404 a resposta indica que não foi possível encontrar o desafio indicado.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Criar um desafio

Cadastra um desafio para o usuário. É o mesmo que afirmar que um usuário desafiou outro.

```
POST /desafio/criarDesafio
```

Os parâmetros são (corpo da requisição):

Nome	Tipo	Descrição
nome	string	Nome do desafio, escolhido pelo usuário
desafiado	number	identificador do desafiado
is_item	boolean	indicador de existe item ou não
item_desafiante	number	identificador de item do jogador, se for mandando
desafiante	token	token mandando no header da requisição

Em caso de sucesso (código 200) retorna um objeto com os seguintes atributos.

- **Message: Desafio criado com sucesso!:** Mensagem de sucesso na criação do desafio.

Em caso de erro com código 404 a resposta indica que não foi possível encontrar o desafiado, o item do desafiante indicado ou se o nome dado para o desafio já existe.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Adicionar missão a um desafio

Cadastra uma missão ao desafio em questão. É o mesmo que afirmar que um usuário desafiou outro.

```
POST /missaodesafio/addMissoesDesafio
```

Os parâmetros são (corpo da requisição):

Nome	Tipo	Descrição
id_missao	number	identificador da missão
id_desafio	number	identificador do desafio

Em caso de sucesso (código 200) retorna um objeto com os seguintes atributos.

- **Message:** Missão adicionada ao desafio com sucesso!: Mensagem de sucesso na adição da missão ao desafio.

Em caso de erro com código 404 a resposta indica que:

- Não foi possível encontrar o desafio;
- Não foi possível encontrar a missão;
- Missão já está fechada;
- Missão não pertence ao jogador logado.

Em caso de erro com código 500 a resposta indica um erro de execução do servidor.

Trocar item do desafio

Troca o item apostado em um desafio por ou item, ou retira o item aposta.

```
PUT /desafio/trocarItemDesafio
```

Os parâmetros são (corpo da requisição):

Nome	Tipo	Descrição
id	number	identificador do desafio
item	number	identificador do item do jogador

Em caso de sucesso (código 200) retorna um objeto com os seguintes atributos.

- **Message:** Item do desafio atualizado com sucesso!: Mensagem de sucesso na atualização de um item do desafio.

Em caso de erro com código 404 a resposta indica:

- Não foi possível encontrar o desafio;
- Não foi possível encontrar o item do jogador;
- O desafio não está mais nesse estágio;

- Item do jogador está bloqueado;
- Esse item não pertence ao jogador em questão.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Alterar status do desafio

Troca o status de um determinado desafio.

```
PUT /desafio/trocarItemDesafio
```

Os parâmetros são (corpo da requisição):

Nome	Tipo	Descrição
id	number	identificador do desafio
status	string	status para qual o desafio vai ser atualizado. “C” para concluído e “A” para andamento.

Em caso de sucesso (código 200) retorna um objeto com os seguintes atributos.

- **Message: Desafio atualizado com sucesso!:** Mensagem de sucesso na atualização do desafio.

Em caso de erro com código 404 as respostas podem ser:

- Não foi possível encontrar o desafio;
- Você não é desafiante e nem o desafiado;
- O desafio não está mais nesse estágio.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Deletar desafio

Deleta um desafio do sistema. Se o desafiado deleta, é o mesmo que ele recusar.

```
DELETE /desafio/deletarDesafio/:id
```

Os parâmetros são de rota :id representa o identificador do desafio que deve ser deletado.

Em caso de sucesso (código 200) retorna um objeto com os seguintes atributos.

- **Message:** Desafio deletado com sucesso!: Mensagem de sucesso na deleção do desafio.

Em caso de erro com código 404 as respostas podem ser:

- Não foi possível encontrar o desafio;
- Você não é desafiante e nem o desafiado;
- O desafio não está mais nesse estágio.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Só é possível deletar um desafio, se ele estiver na fase de “proposta”.

Remover uma missão de um desafio

Deleta um desafio do sistema. Se o desafiado deleta, é o mesmo que ele recusar.

DELETE /missaodesafio/deletarMissaoDesafio/:id
--

Os parâmetros são de rota :id representa o identificador da missão do desafio que deve ser deletado.

Em caso de sucesso (código 200) retorna um objeto com os seguintes atributos.

- **Message:** Missão deletada do desafio com sucesso!: Mensagem de sucesso na deleção da missão do desafio.

Em caso de erro com código 404 as respostas podem ser:

- Não foi possível encontrar a missão do desafio;
- Você não é desafiante e nem o desafiado desse desafio;
- O desafio não está mais nesse estágio.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Só é possível deletar uma missão de desafio se o desafio estiver na fase de “aposta”.

Burndown

Listar burndowns

Lista todos os burndowns da plataforma.

```
GET /burndown/listarBurndown
```

Em caso de sucesso (código 200) retorna um array de objetos com os atributos:

- `is_active` (boolean): Indicador se o burndowns está ativo ou não;
- `data_inicio` (date): data de início desse burndowns;
- `data_termino` (date): data de termino desse burndowns;
- `quantidade_missao` (number): quantidade de missões que deve ser queimada ao decorrer da semana;
- `quantidade_missao_falta` (number): quantidade de missões que acumularam da semana passada;
- `quantidade_queimada_dia1` (number): quantidade de missões queimadas no dia 1;
- `quantidade_queimada_dia2` (number): quantidade de missões queimadas no dia 2;
- `quantidade_queimada_dia3` (number): quantidade de missões queimadas no dia 3;
- `quantidade_queimada_dia4` (number): quantidade de missões queimadas no dia 4;
- `quantidade_queimada_dia5` (number): quantidade de missões queimadas no dia 5.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

Consultar um *burndown*.

Consultar um burndowns específico:

```
GET /burndown/consultarBurndown/:id
```

O parâmetro de rota `:id` representa o identificador do *burndown* específico, do qual deve ser consultado.

Em caso de sucesso (código 200) retorna um objeto com os atributos descritos na funcionalidade de listar todos os burndowns.

Em caso de erro com código 404 a resposta indica que não foi possível encontrar o *burndown* indicado.

Em case do erro com código 500 a resposta indica um erro de execução do servidor.

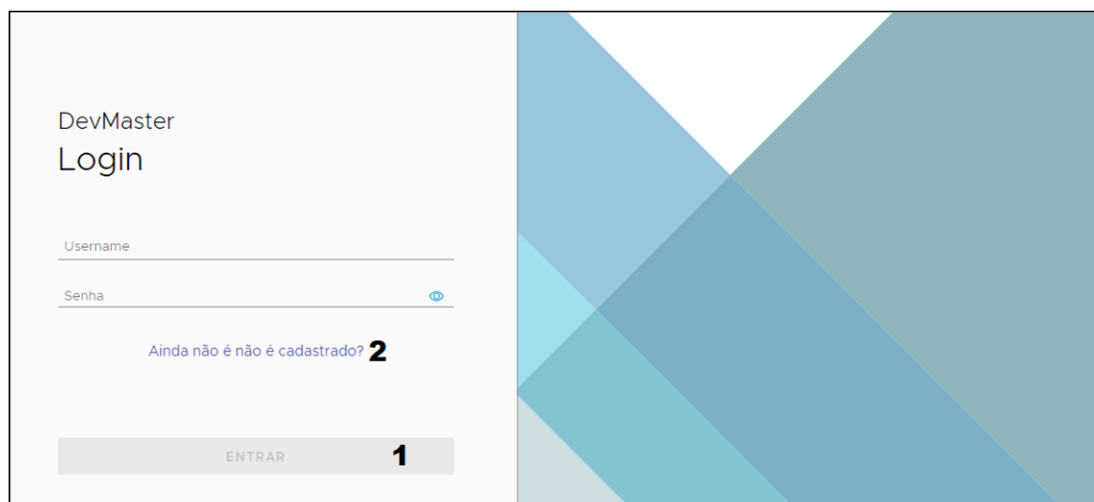
4.3.2. Frontend

A presente seção tem como objetivo mostrar a interface de interação do jogador e como essa interface comunica com o *backend* do DevMaster e a API do Gitlab. Em vista disso, para ficar mais didático, essa seção foi dividida em 8 subseções, sendo elas: login, header, *dashboard*, missões, desafio, evento, *burndown* e área administrativa.

4.3.2.1. Login

A tela inicial do *frontend* do DevMaster vai depender se o usuário está logado ou não. Onde se ele estiver logado, será redirecionado para a tela de dashboard (4.3.2.2.), e se não tiver, será redirecionado para a tela de login. O que define se o usuário está logado ou não é o dados salvos no *local storage*, ou armazenamento local, que uma funcionalidade do *javascript* para armazenar dados no navegador, do lado do cliente.

Figura 35: Tela de Login.



A tela de login é composta por dois campos para que o usuário se autentique, os campos são “*username*” e “*senha*”. Após o usuário ter preenchido os dados, o botão de entrar (ponto 1 da Figura 35) é liberado, e ao clicar nele é feito uma requisição para a rota de login da API do DevMaster, e se o usuário existir e os dados que ele passou tiverem corretos, ele consegue realizar o login e é redirecionado para a tela de dashboard.

Mas se o usuário ainda não tiver uma conta cadastrada, basta ele clicar em “Ainda não é cadastrado?” (ponto 2 da Figura 35), que é redirecionado para a tela de cadastro, tela essa apresentada na Figura 36.

Figura 36: Tela de Cadastro.

Nome	Nome
Sobrenome	Sobrenome
Email	Email
Username	username Mesmo usuario do Gitlab.
Senha	Senha
Private Token	Private Token Private token obtido no gitlab
<div>CADASTRAR! 1</div>	

Essa tela possui 6 campos a serem preenchidos pelo usuário. Quando o usuário preenche todos os campos o botão “cadastrar!” (ponto 1 da Figura 36) é liberado, e ao clicar nele o usuário é redirecionado para a tela de login. Mas, antes disso é feita uma requisição para a API do Gitlab, passando os “Username” e o “Private Token”, para então buscar a foto de perfil do usuário no Gitlab e anexar ao corpo da requisição que faz o cadastro do usuário na *backend*. Essa ação é imperceptível ao usuário.

4.3.2.2. Header

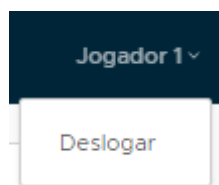
Com o jogador logado a parte superior de toda a tela é alterada, onde vão aparecer as opções de páginas que o jogador pode acessar. Como mostra a Figura 37.

Figura 37: Header de todo o Frontend.



O ponto 1 da Figura 37 é o que redireciona o jogador para a tela de dashboard, já 2 o redireciona para a tela de missões, 3 para a tela de desafio, o 4 para a tela de evento, o 5 para a tela de *burndown* e o 6 abre as opções do jogador deslogar, como mostra a Figura 38.

Figura 38: Opção do jogador de deslogar do sistema.



Quando o jogador clica em “deslogar”, conforme apresentado na Figura 38, automaticamente é limpo os dados dele do *local storage* e ele é redirecionado para a tela inicial, que é a de login.

4.3.2.3. Dashboard

A tela de dashboard é composta por 6 áreas, sendo elas a de perfil, mochila, ranking xp, ranking missões, ranking desafio e ranking evento. Essa seção realiza 3 requisições para o *backend*, uma para buscar o jogador logado, outra para buscar os itens desse jogador e outra para listar todos os jogadores.

Figura 39: Área de perfil.



A Figura 39 aponta como ficou a área de perfil, da qual o ponto 1 apresenta a foto do jogador, que é a mesma foto do perfil do usuário do Gitlab deste jogador. O ponto 2 apresenta os dados desse jogador que vem das ações realizadas dele, dentro da

plataforma. E o ponto 3 aponta a barra de progresso para o próximo nível do jogador, ou seja, aponta o quanto ele já progrediu em relação a alcançar o próximo nível.

Figura 40: Área de mochila.



A Figura 40 apresenta a mochila do jogador em questão, da qual o ponto 1 mostra a imagem do item, e o ponto 2 mostra o nome do item, a quantidade e a quantidade bloqueado que o jogador tem desse determinado item.

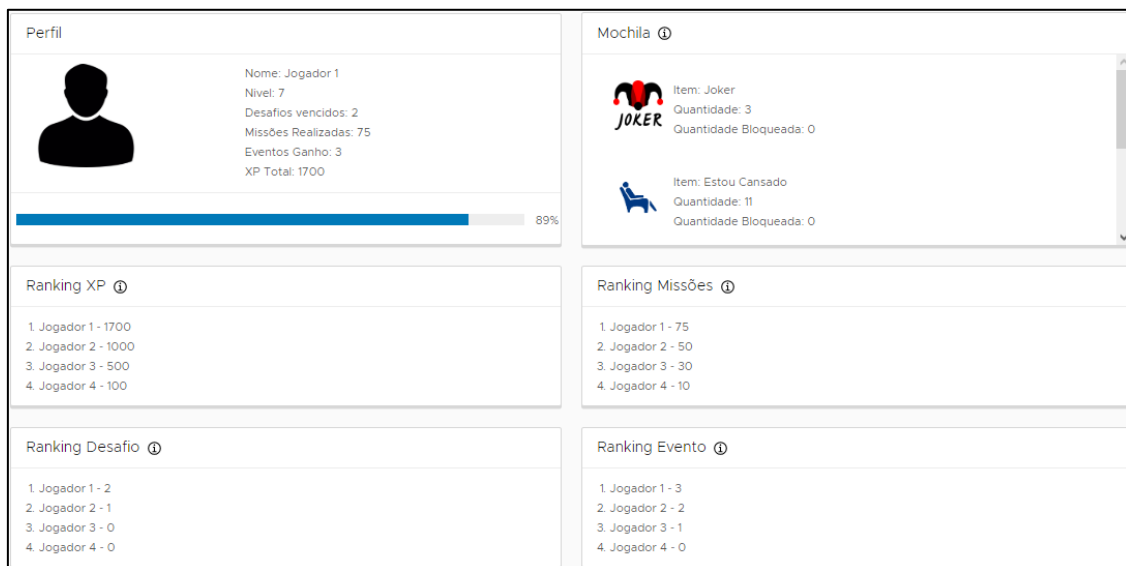
Figura 41: Rankings do dashboard.

Ranking XP ① 1 1. Jogador 1 - 1700 2. Jogador 2 - 1000 3. Jogador 3 - 500 4. Jogador 4 - 100	Ranking Missões ① 2 1. Jogador 1 - 75 2. Jogador 2 - 50 3. Jogador 3 - 30 4. Jogador 4 - 10
Ranking Desafio ① 3 1. Jogador 1 - 2 2. Jogador 2 - 1 3. Jogador 3 - 0 4. Jogador 4 - 0	Ranking Evento ① 4 1. Jogador 1 - 3 2. Jogador 2 - 2 3. Jogador 3 - 1 4. Jogador 4 - 0

A Figura 41 apresenta toda a área de ranking do dashboard, que é responsável por dar uma visão geral do desenvolvimento do jogador na plataforma, da qual o ponto 1 é ranking que ordena o jogadores por aquele que tem a maior XP, o ponto 2 é o ranking que ordena os jogadores pelo que tem a maior quantidade de missões finalizadas, o ponto 3 é o ranking que ordena os jogadores pelo que tem a maior

quantidade de desafios vencidos, e o ponto 4 ordena os jogadores pelo que tem a maior quantidade de eventos vencidos, ou seja, quantidade de eventos que o determinado jogador ficou em primeiro.

Figura 42: Tela de dashboard.



A Figura 42 apresenta a tela completa de *dashboard*. Da qual é possível ver todos os elementos descritos anteriormente em união.

4.3.2.4. Missões

Assim que o jogador é redirecionado para a tela de missões, ele tem duas opções, escolher as suas missões solos, ou as suas missões em grupo.

Figura 43: Tela de missões.



O ponto 1 da Figura 43 mostra onde o usuário precisa clicar para ser redirecionado para as missões solo. Já o ponto 2 mostra onde o usuário precisa clicar para ser redirecionado para as missões em grupo.

Figura 44: Projetos do Gitlab da Fábrica de Software.

Projetos		
Projeto	Descrição	Missões
Projeto 1 1	Descrição do projeto 1 2	IR! 3
Projeto 2	Descrição do projeto 2	IR!

Clicando em missões solo, o jogador é redirecionado para a tela de projetos (Figura 44) e é mostrado todos os projetos do Gitlab da fábrica de *software*. O ponto 1 da Figura 44 representa o nome do projeto, já o ponto 2 apresenta a descrição desse projeto, e o ponto 3 é um botão que ao se clicado pelo usuário, o redireciona para a tela de missões desse projeto.

Figura 45: Missões concluídas de um determinado projeto.

Missões do Projeto: Projeto 1 ⓘ	
Concluídas	Em Andamento
1	2
Missão: Missão fechada 1 3 XP adquirida com a missão: 10 4	
<div>Não terminada na data certa!</div> <div>Não terminada no tempo certo!</div> 5	
Missão: Missão fechada 2 XP adquirida com a missão: 80	
<div>Terminada na data certa!</div> <div>Terminada em tempo record!</div>	

Primeiramente é realizado uma requisição para a API do DevMaster, que retorna todas as missões, que são organizadas em missões concluídas e missões em andamento. A Figura 45 mostra as missões, do jogador logado, de um determinado projeto, por padrão será mostrado apenas as missões finalizadas (ponto 1 da Figura 45). O ponto 3 dessa Figura mostra o nome da missão em questão, o ponto 4 mostra a quantidade de

XP que o jogador ganhou com essa missão e o ponto 5 mostra as informações dessa missão, essas informações assim como a XP ganha, depende de como o usuário fechou essa missão, se foi no tempo ou data correta. Já o ponto 2 ao ser clicado mostra as missões daquele projeto que o usuário ainda não concluiu, como mostra a Figura 46.

Figura 46: Missões não concluídas de um determinado projeto.



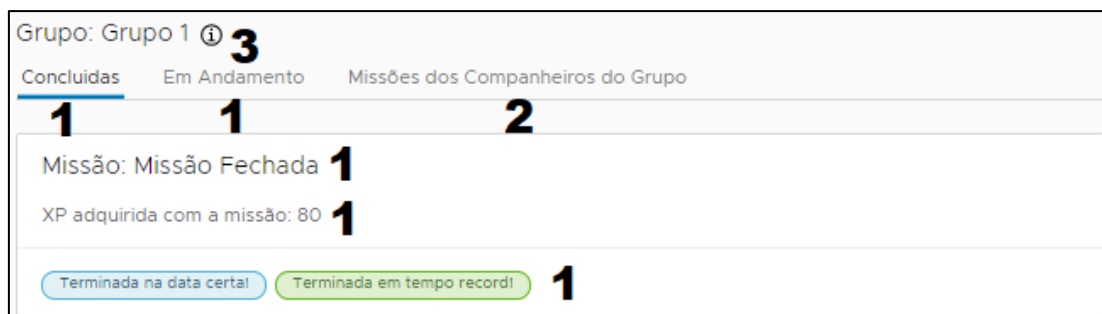
O ponto 1 da Figura 46 mostra o nome da missão, o ponto 2 mostra o quanto de XP o usuário pode adquirir ao concluir essa missão, já o ponto 3 mostra quanto de XP o usuário vai perder se ele não finalizar essa missão na data e no tempo correto.

Figura 47: Grupos de missões do DevMaster.



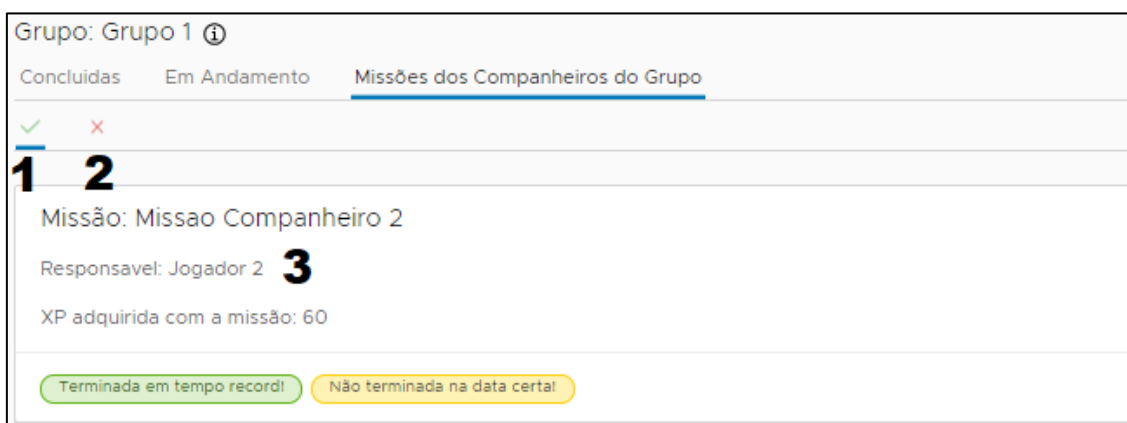
Primeiramente é realizado uma requisição para a API do DevMaster, onde são buscados todos os grupos da plataforma. A Figura 47 mostra a tela de grupos, que é para onde o jogador é redirecionado ao clicar em missão em grupo (Figura 43). Da qual o ponto 1 representa o nome do grupo, o ponto 2 representa a XP ganha ao finalizar esse grupo e o ponto 3 é um botão que ao ser clicado pelo usuário, o redireciona para a tela de missões desse grupo.

Figura 48: Missões fechadas de um determinado grupo.



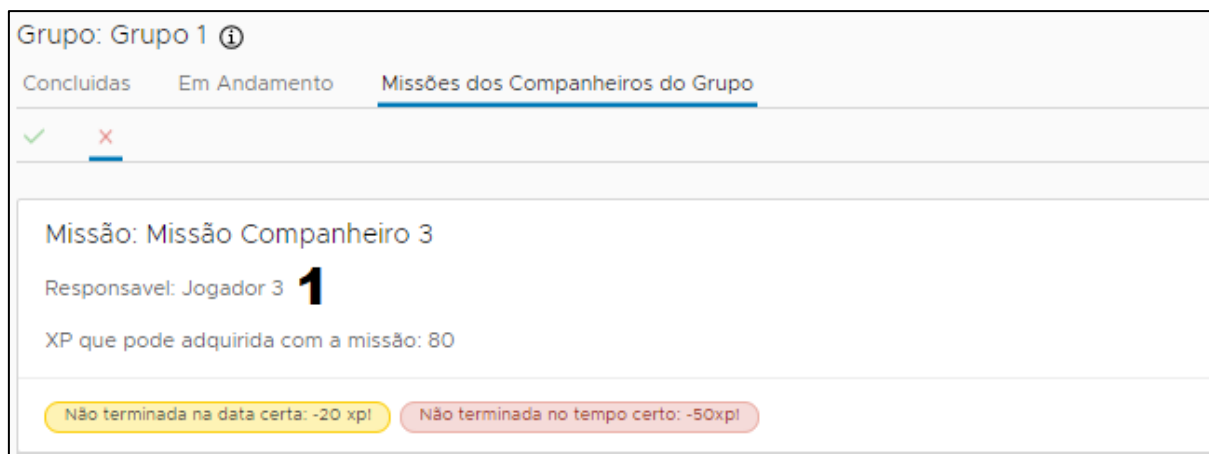
Assim que é redirecionado para essa tela, é realizada uma requisição para a API do DevMaster, que busca todas as missões daquele grupo. A Figura 48 mostra as missões do grupo, das quais inicialmente, por padrão é mostrada apenas as missões daquele grupo que foram concluídas. Os pontos da Figura 48 que são marcados como "1" são os pontos referente as missões do jogador, que foram explicadas anteriormente. O ponto 3 refere ao nome do grupo, já o ponto 2 refere as missões dos companheiros do grupo, com mostra a Figura 49.

Figura 49: Missões fechadas dos companheiros de um determinado grupo.



Ao clicar no ponto 2 da Figura 49, é mostrado para o usuário as missões que os companheiros daquele grupo finalizaram, como mostrou a Figura 50 no ponto 1. Já o ponto 3 da Figura 49 mostra o companheiro responsável por aquela determina missão, por fim o ponto 2 mostra a missões daquele grupo que não foram finalizadas.

Figura 50: Missões abertas dos companheiros de um determinado grupo.

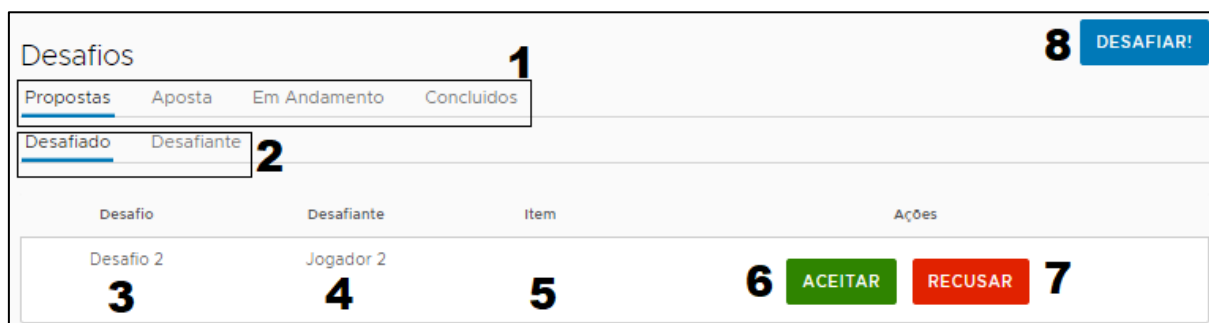


A Figura 50 mostra as missões dos companheiros do grupo, que não foram finalizadas. O ponto 1 dessa Figura mostra o companheiro responsável por essa missão.

4.3.2.5. Desafio

Assim que o jogador é redirecionado para a tela de desafio é realizada uma requisição para a API do DevMaster para retornar todos os desafios daquele usuário. A tela Desafio é representada pela Figura 51.

Figura 51: Tela inicial de desafio.



O ponto 8 da Figura 51, é um botão que ao ser clicado abre uma modal (que é uma janela secundaria que aparece na principal, que é apresentado na Figura 52) que é responsável por criar um desafio, da qual possibilita o usuário escolher um nome, escolher o desafiante e o item que será apostado, onde clicando em “ok” é realizada uma requisição que cria o desafio.

Figura 52: Modal de criação do desafio.


Modal de criação do desafio com o título "Crie seu desafio". O formulário contém três campos: "Nome do Desafio:" (campo de texto), "Desafiado:" (campo de seleção com uma seta para baixo) e "Item:" (campo de seleção com uma seta para baixo). No canto inferior direito, há dois botões: "CANCEL" (azul claro) e "OK" (azul escuro).

O ponto 1 da Figura 51 é o que indica o estágio do desafio, logo ao clicar em qualquer um dos pontos (proposta, aposta, em andamento e concluídos), os desafios que apareceram são os desafios daquele jogador referente ao determinado estágio. Já o ponto 2, refere-se aos desafios que o jogador logado é o desafiado ou o desafiante. Por padrão são mostrados os desafios que estão no estágio de proposta, que tem o jogador logado como desafiado. Já os pontos 3, 4 e 5, mostram: o nome, o desafiante e o item apostado pelo desafiante no desafio.

Os pontos 6 e 7 da Figura 51 são as ações que o jogador pode realizar. Clicando no botão 6, o sistema realiza uma requisição na API do DevMaster que faz com que o jogador escolha um item e aceite o desafio (Figura 54), sendo que quando essa ação é realizada, o desafio tem seu estágio mudado de “proposta” para “aposta”. O ponto 7 da Figura 51 permite a recusa do desafio, quando selecionada é o responsável por realizar uma requisição que deleta esse desafio.

Figura 53: Desafio no estágio de proposta na visão do desafiante.

Propostas	Aposta	Em Andamento	Concluídos
Desafiado	Desafiante		
Desafio	Desafiado	Item	Ações
Desafio 1	Jogador 2		<div>1</div> <div>EDITAR</div> <div>EXCLUIR</div> <div>2</div>

A Figura 53 mostra quais ações o jogador que criou o desafio pode realizar. Onde o ponto 1 cria uma modal (Figura 54) para que o desafiante possa trocar o item que ele desafiou, e o ponto 2 exclui o desafio, é claro que apenas desafios no estágio de “proposta” podem ser apagados. Ambos os pontos fazem uma requisição para o *backend*, uma que atualiza o desafio e a outra que o deleta.

Figura 54: Modal para troca do item do desafiante do desafiado.



Escolha o item:

Item:

OK

Assim que o desafiado aceita o desafio a área que se localiza esse desafio é alterada de “proposta” para “aposta”. A Figura 55 aponta os desafios do jogador logado que estão no estágio de “aposta”, onde tanto se o jogador for o desafiado quanto ou desafiante, a tela terá as mesmas opções de ações que o jogador pode realizar.

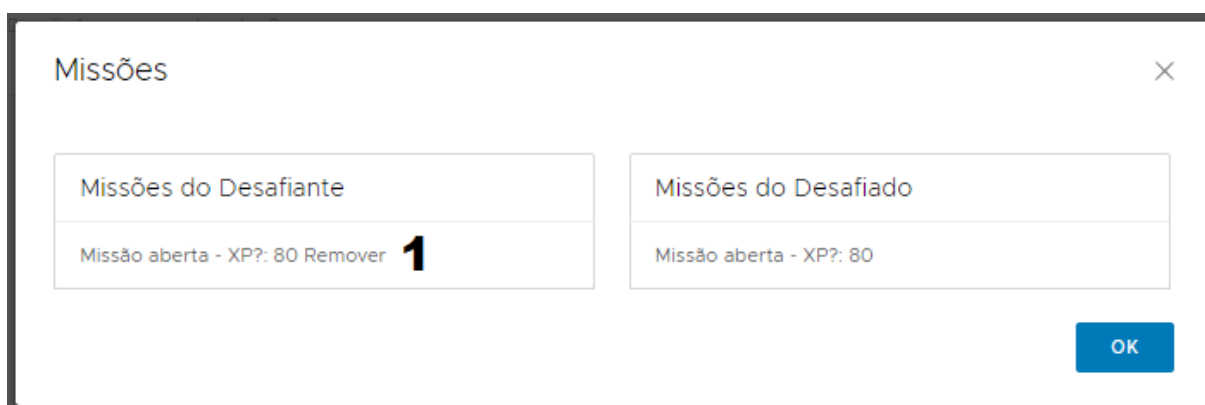
Figura 55: Desafios do jogador que estão em estágio de aposta.

Propostas	<u>Aposta</u>	Em Andamento	Concluídos	
<u>Desafiado</u>	Desafiante			
Desafio	Desafiante	Item	Missões	Ações
Desafio 2	Jogador 2		1 <div>MISSÕES</div>	2 <div>ADD MISSÃO</div> <div>NEXT!</div> 3

O ponto 2 da Figura 55 é um botão que abre uma modal para que o jogador adicione uma missão dele, ao desafio em questão, como mostra a Figura 56.

Figura 56: Modal para adicionar as missões ao desafio.

O ponto 1 da Figura 55 é um botão que abre uma modal, da qual mostra todas as missões do desafio em questão. Como mostra a Figura 57.

Figura 57: Modal para mostrar as missões do desafio.

Vale lembrar que se o jogador logado for o dono da missão, adicionada ao desafio, ele pode removê-la, para isso basta clicar em “remover” do lado da missão na modal que mostra as missões do desafio (Figura 57, ponto 1), sendo que só é possível remover as missões de um desafio que esteja no estágio de “aposta”. Já o ponto 3 da Figura 55 é usado para atualizar o estágio do desafio, onde é alterado de “aposta” para “andamento”.

Figura 58: Desafios do jogador que estão no estágio de andamento.

Propostas	Aposta	Em Andamento	Concluídos	
Desafiado	Desafiante			
Desafio	Desafiado	Item	Missões	Ações
Desafio 1	Jogador 2		MISSÕES	CONCLUIR! 1

A Figura 58 mostra os desafios do jogador que estão no estágio de “andamento”, logo o ponto 1 dessa Figura é um botão que ao ser clicado conclui o desafio em questão. Tanto o desafiante quanto o desafio têm as mesmas opções de ações nesse estágio.

Figura 59: Desafios do jogador que estão no estágio concluído.

Propostas	Aposta	Em Andamento	Concluídos
Desafiado	Desafiante	✓	✗
1	2	3	4
Desafio	Desafiante	Item	Missões
Desafio 5	Jogador 3	Joker	5 Status
			Ganhou!
			MISSÕES

Para finalizar a seção de desafio, é apresentada a Figura 59, na qual o ponto 5 mostra o status daquele desafio, ou seja, mostra se o jogador logado ganhou o perdeu o desafio. O ponto 1 e 2 mostra os desafios do jogador logado, como desafiado e desafiante. Já o ponto 3 mostra todos os desafios que o jogador logado ganhou, e o ponto 4 mostra todos os desafios que ele perdeu.

4.3.2.6. Evento

Assim que o jogador é redirecionado para a tela de evento é realizada uma requisição para a API do DevMaster para retornar todos os eventos. Por padrão é mostrado para ele todos os eventos que estão ativos, como mostra a Figura 60.

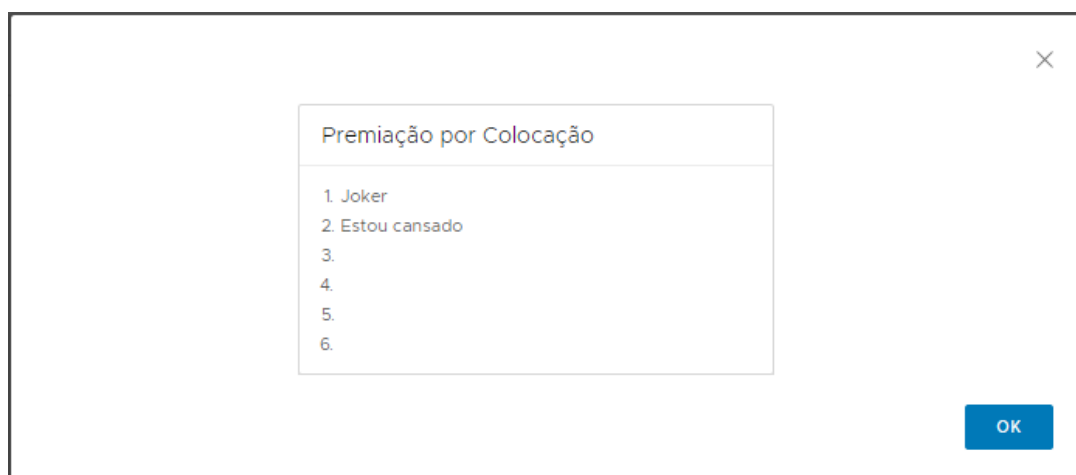
Figura 60: Tela de Eventos ativos.

Eventos	Ativos	Encerrados
1	2	
Evento	XP	Data de Inicio
Evento ativo	2	2019-06-02T20:49:39Z
3	4	5
		Premiação
		Gerar Premiação
		6
		GERAR! 7
		PREMIOS

O ponto 1 da Figura 60 é o que retrata que os eventos que estão sendo exibidos são os que estão ativos, o ponto 3 mostra o nome do evento, o ponto 4 mostra o multiplicado de XP do evento, o ponto 5 mostra a data que o evento iniciou, o ponto 7 é um botão que gera as premiações, ao clicar nesse botão é realizada uma determinada requisição para a API do DevMaster que gera essa ação, só irá ser gerado se o evento

não estiver ativo e se ainda não foi gerada essa premiação para esse evento, logo se nenhum desses requisitos forem atendidos não vai acontecer nada. Por último o ponto 6 dessa Figura, abre um modal que apresenta os possíveis prêmios para esse evento, como mostra a Figura 61.

Figura 61: Modal de prêmios de um evento.



A Figura 61 mostra as premiações para cada colocação no ranking do evento, como o moderador pode escolher quais colocação são premiadas, alguns pontos podem não ter nenhum item.

Ao clicar no ponto 2 da Figura 60, mostra para o usuário quais eventos foram encerrados, como mostra a Figura 62.

Figura 62: Tela de Eventos encerrados.

Eventos				
Ativos		Encerrados		
Evento	XP	Data de Inicio	Data de Final	Premiados
Evento Encerrado	2	2019-05-18T03:41:15Z	2019-05-30T03:41:23Z	1
				2

O ponto 1 da Figura 62 mostra a data em que esse evento foi encerrado, já o ponto 2 é um botão que ao ser clicado pelo usuário mostra a premiação desse evento. Como aponta a Figura 63.

Figura 63: Modal de premiações de um evento encerrado.

O ponto 1 da Figura 63 mostra o nome do jogador que ficou na primeira colocação no ranking do evento, igual acontece com todos os outros jogadores do ranking desse evento, só que em colocações diferente, já o ponto 2 mostra o item que esse jogador conseguiu, é claro que apenas as colocações com prêmios vão ser premiadas com algum item.

4.3.2.7. *Burndown*

Assim que o jogador é redirecionado para a tela de *burndown*, é realizada uma requisição para a API do DevMaster para retornar todos os burndowns. Dos quais são apresentados para o jogador, como mostra a Figura 64.

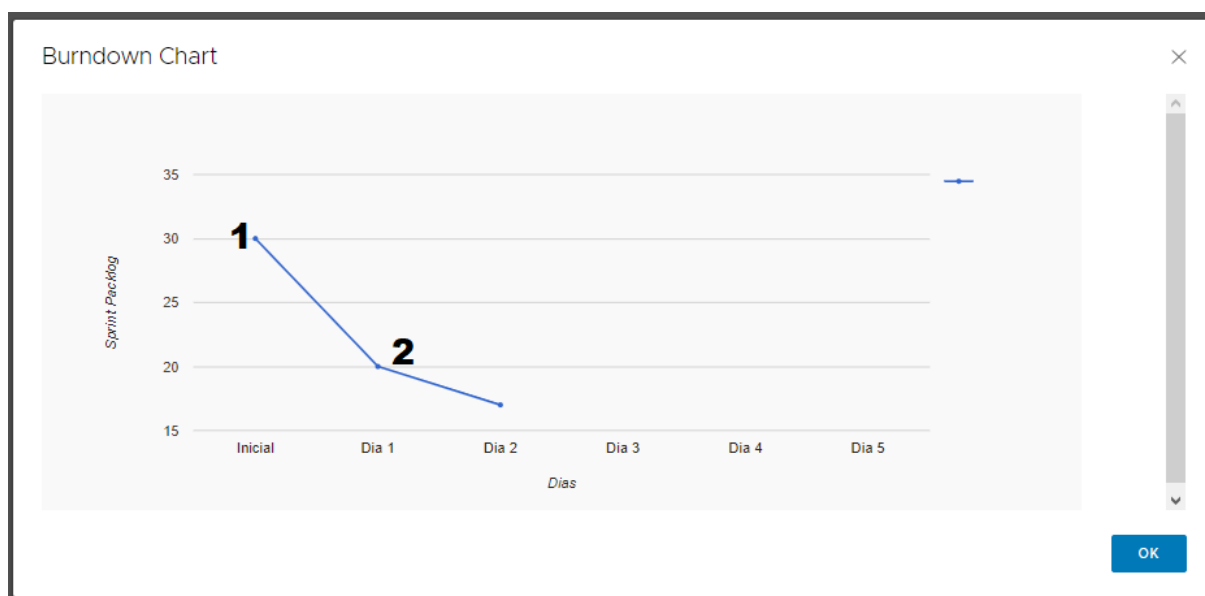
Figura 64: Tela de *Burndown*.

Burndowns					
1 Ativo	2 Data Inicio	3 Data Término	4 Missões	5 Missões Queimadas	Gráfico
true	2019-05-26		50	50	ABRIR 6
false	2019-05-18	2019-05-26	30	13	ABRIR

O ponto 1 da Figura 64 é onde mostra de aquele *burndown* está ativo ou não, já o ponto 2 mostra a data de início dele, o ponto 3 mostra a data de termino, apenas os *burndowns* que terminaram tem essa data, o ponto 4 mostra as todas as missões (*issues*)

que devem ser queimadas ao longo da semana, e o ponto 5 mostra as missões que foram queimadas.

Figura 65: Gráfico de um determinado *burndown*.



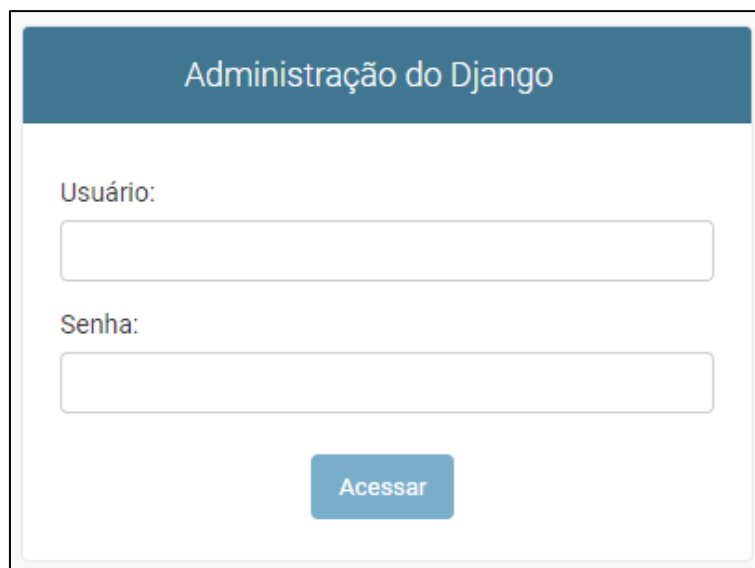
Ao clicar no botão do ponto 6 da Figura 64, abre uma modal com o gráfico referente ao determina *burndown* como mostra a Figura 65. Nessa Figura o ponto 1 refere a quantidade de missões que devem ser queimadas durante a semana, e o ponto 2 refere aos pontos de intersecção entre o dia e a quantidade, do qual mostra quantas missões ainda devem ser queimadas com o decorrer da semana.

4.3.2.8. Área Administrativa

A área administrativa não faz parte do mesmo *frontend* que os jogadores acessam, logo que essa área foi provida pelo Django Admin, porém, pelo fato de ainda se tratar de telas, essa área foi mostrada aqui e não no *backend*.

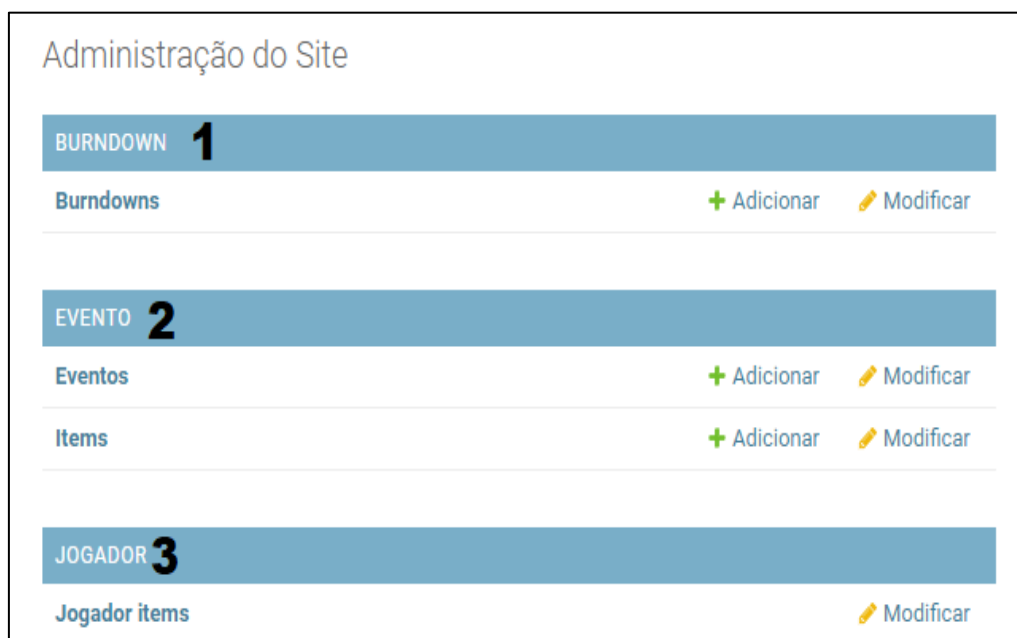
O moderador do DevMaster também deve criar uma conta, igual todos os jogadores, contudo, logo foram alterados o tipo e as permissões dele, para que assim ele consiga acessar os Django Admin. Logo ao acessar a rota do *backend*:

/admin/

Figura 66: Login do Django Admin.

The image shows the Django Admin login interface. At the top, there is a dark blue header with the text "Administração do Django" in white. Below the header, the page has a light gray background. There are two input fields: "Usuário:" (Username) and "Senha:" (Password), both with white text labels and light gray borders. Below the password field is a blue button with the text "Acessar" (Access) in white.

A Figura 66 mostra o retorno da rota em questão. Com esse retorno o moderador consegue realizar o login com as suas credenciais. Após logado, a Figura 67 mostra a tela inicial da área de administrativa.

Figura 67: Login do Django Admin.

The image shows the Django Admin dashboard. At the top, there is a header with the text "Administração do Site" in gray. Below the header, the dashboard is divided into three main sections, each with a blue header bar and a list of items. The first section is labeled "BURNDOWN 1" and contains a list of "Burndowns" with a "+ Adicionar" button and a "Modificar" button. The second section is labeled "EVENTO 2" and contains a list of "Eventos" with a "+ Adicionar" button and a "Modificar" button, and a list of "Items" with a "+ Adicionar" button and a "Modificar" button. The third section is labeled "JOGADOR 3" and contains a list of "Jogador items" with a "Modificar" button.

O ponto 1 da Figura 67 refere-se as ações de criação, edição e exclusão de *burndown*, da mesma forma que no ponto 2, porém, o ponto 2 refere a eventos e a itens. Já o ponto 3, refere a apenas a ação de modificar um determinado item de um jogador.

Figura 68: Ações do moderador.

Barra de ações do moderador com os seguintes botões:

- Apagar (botão vermelho)
- Salvar e adicionar outro(a) (botão azul claro)
- Salvar e continuar editando (botão azul claro)
- SALVAR (botão azul escuro)

Nos pontos 1 e 2 da Figura 67 o moderador vai conseguir realizar todas as ações, mostradas na Figura 68, para cada um dos itens, exclusivamente na opção de jogador item que o moderador foi restrito apenas as duas últimas ações da Figura 68 (“salvar e continuar editando” e “salvar”).

Para que seja possível o moderador realizar umas dessas ações primeiramente ele deve clicar no item (burndowns, eventos, itens e jogador itens) que ele deseja, com isso, ele é redirecionado a tela de gerenciamento desse determinado item, como mostra a Figura 69.

Figura 69: Tela de gerenciamento de evento.

Tela de gerenciamento de evento com os seguintes elementos:

- Título: Seleccione evento para modificar
- Botão: 2 ADICIONAR EVENTO +
- Filtro: Ação: [dropdown] Ir 0 de 2 selecionados
- Lista de eventos:
 - ☐ EVENTO
 - ☐ Nome: Evento ativo
 - ☐ Nome: Evento Encerrado **1**
- Resumo: 2 eventos

O ponto 1 da Figura 69, mostra os eventos que existe, logo se o moderador clicar em um, ele é redirecionado para a tela de modificação desse evento, como mostra a Figura 68.

Figura 70: Tela de modificação de um evento.

Modificar evento

Name:

☒ Is active

Multiplicador xp:

Data inicio:

Data: Hoje

Hora: Agora

Nota: Você está 3 horas atrás do horário do servidor.

Data final:

Data: Hoje

Hora: Agora

Nota: Você está 3 horas atrás do horário do servidor.

☐ Is finalized

Primeiro:

Segundo:

Terceiro:

Quarto:

Quinto:

Sexto:

Já o ponto 2 da Figura de 69, redireciona o moderador para a página de criação, que é exatamente a mesma página de modificação (Figura 68), porém, sem os dados pré-cadastrados.

Figura 71: Tela de criação de um item.

Name:	<input type="text"/>
Descricao:	<div></div>
Url image:	<div></div>

A Figura 71 referência a tela de criação/modificação de um item, que deve ser passado o nome do item, a descrição do item, que nada mais é do que o efeito dele, e a *url image* que é a *url* responsável por atribuir uma imagem a esse item.

Figura 72: Tela de criação de um *burndown*.

<input checked="" type="checkbox"/> Is active	
Data inicio:	<input type="text"/> Hoje
Nota: Você está 3 horas atrás do horário do servidor.	
Data termino:	<input type="text"/> Hoje
Nota: Você está 3 horas atrás do horário do servidor.	
Quantidade de missões da semana:	<input type="text" value="0"/>
Quantidade de missões da semana passada, não queimadas:	<input type="text"/>
Quantidade queimada dia1:	<input type="text"/>
Quantidade queimada dia2:	<input type="text"/>
Quantidade queimada dia3:	<input type="text"/>
Quantidade queimada dia4:	<input type="text"/>
Quantidade queimada dia5:	<input type="text"/>

A Figura 72 referência a tela de criação/modificação de um *burndown*. Após a criação de um *burndown* o mesmo deve ser atualizado pelo moderador todo dia, com a relação das missões que foram fechadas no dia, essas missões são obtidas através do “*daily scrum*”.

Figura 73: Tela de modificação do item de um jogador.

Modificar jogador item	
Jogador:	<input type="text" value="Jogador: Jogador 2"/>
Item:	<input type="text" value="Joker"/>
Quantidade:	<input type="text" value="1"/>
Quantidade bloqueada:	<input type="text" value="0"/>

Para finalizar, a Figura 73 mostra a tela de modificação de um item de um jogador, vale deixar claro que o moderador apenas modifica a quantidade desse item, quando um jogador o usar, logo se a quantidade for igual a quantidade bloqueada, o moderador não pode fazer nada.

5 CONSIDERAÇÕES FINAIS

Como apresentado no decorrer desse trabalho, a utilização do Gitlab e do *Scrum*, proporcionou que a gamificação, idealizada, fosse realmente concluída. No contexto de planejamento da gamificação, a utilização do *Octalysis* juntamente com o *Framework D6* aconteceu de maneira extremamente agradável, quase como se um complementasse o outro.

De modo geral, levando o presente trabalho como base, é possível afirmar que o *Octalysis* foi responsável por criar o contexto, apresentando quais elementos, presentes na fábrica de software, foram gamificados. Esses elementos não são relacionados aos elementos do desenvolvimento de software, mas sim, a quais pontos devem ser gamificados, esses pontos são os mesmos relacionados aos seus *core drives*.

Já o *Framework D6* foi responsável por buscar os elementos de jogos que compuseram os pontos, abordados pelo *Octalysis*, e como foi estruturado o ambiente que os comportam.

A escolha e aplicação desses dois *frameworks* não é uma garantia de que será criado uma boa gamificação, para isso é necessário entender o contexto que está sendo gamificado.

No DevMaster é tido o contexto da fábrica de software do CEULP/ULBRA, onde graças ao *Octalysis*, foi decidido que pontos devem ser gamificado dentro da equipe. Porém, outro contexto é o de desenvolvimento de software esse que tem como base os elementos de desenvolvimento.

Ou seja, não só é necessário que seja explorado o contexto da convivência da equipe de desenvolvimento, mas também o desenvolvimento em si. De modo a saber como a equipe trabalha.

Graças ao aprofundamento dentro desse contexto que foi possível a criação do DevMaster, logo que, sem isso não seria encontrado as *issues* e as *milestones*, que por sua vez não seriam aplicados o *Octalysis* e o *D6*.

De modo geral, pode ser concluído que a base para criação do DevMaster se deu através de etapas bem definidas, que buscam o entendimento do contexto de

desenvolvimento (elementos de desenvolvimento de software), o contexto da equipe em si (Octalysis) e por fim do desenvolvimento da gamificação (*framework D6*).

5.1. TRABALHOS FUTUROS

Pelo fato do desenvolvimento de software se tratar de algo gigantesco, onde apenas alguns itens, como as *issues* e as *milestones/sprint*, proporcionarem uma gamificação razoavelmente grande, itens como os *merge request*, *commits*, *features*, *branchs*, reuniões diárias, *sprint retrospective*, dentre outros, não foram abordados, logo que para cada um desses itens deve ser feito um trabalho totalmente novo, pelo fato de cada um conter sua própria peculiaridade e complexidade.

Porém, antes mesmo de apresentar novos itens para a gamificação, deve ser feito uma análise do que já foi desenvolvido e verificar se a implantação foi benéfica ou não. Com isso em mente, para trabalhos futuros, primeiramente deve ser analisado o que foi desenvolvido. Essa análise deve ser feita, com base nos resultados da fábrica de software, antes e depois da implantação do presente trabalho. Com a utilização do Gitlab, cada ação realizada, dentro dela, gera uma data, onde através da API do Gitlab, passando essa data como parâmetro, é possível rastrear o que foi desenvolvido, nesse determinado intervalo de tempo.

É claro que não apenas esses dados devem ser levados em consideração, uma vez que os desenvolvedores utilizaram a plataforma, estarão aptos a opinar. Por fim, com essas duas métricas (relação do que foi desenvolvido com a opinião dos desenvolvedores) o primeiro trabalho futuro ganha forma, do qual ganha o nome de “análise”.

Já o segundo trabalho futuro tem como base acrescentar ao DevMaster mais elementos do desenvolvimento de software e de jogos. Elementos de desenvolvimento como os do início dessa seção (5.1.). Já os elementos de jogos, antes mesmo da análise, podem ser citados três, sendo eles:

- **Enredo:** Esse elemento é o que será responsável por montar uma história, que pode ser própria ou baseada em uma já existe, que envolva o jogador a plataforma de maneira mais divertida.

- **Avatar:** Esse elemento é o que será responsável por representar o jogador dentro da plataforma, podendo ser personalizável a maneira que o jogador preferir.
- **Itens dentro do jogo:** Esse elemento é o que traz itens para dentro do jogo, diferentes dos itens já presentes no DevMaster, esses itens podem ser utilizados para alguma ação dentro da plataforma, onde inicialmente serão ser utilizados para equipar o avatar.

Com isso em mente, esse trabalho será chamado de “incremento”. Logo será necessário a realização de uma espécie de análise nessas novas funcionalidades que serão adicionadas, da qual entra a ideia do primeiro trabalho relacionado, de modo que ao seu fim encadeia o segundo trabalho relacionado, dessa forma entrando em *loop*.

A ideia de *loop* é o foco da presente seção, pois através dele será possível que o presente trabalho evolua de maneira exponencial. Dessa forma alcançando novos contextos, com novos focos, dos quais iniciaram novamente esse ciclo.

REFERÊNCIAS

BRASIL, Udacity. **8 motivos para aprender a programar em Python**. Disponível em: <<https://br.udacity.com/blog/post/motivos-aprender-programar-python>>. Acesso em: 27 maio 2018.

CARDOSO, D. C. **CODE LIVE: Gamificação de um ambiente virtual de programação**. 2015. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação). Centro Universitário Luterano de Palmas, Palmas, Tocantins, 2015. Disponível em: <<http://ulbra-to.br/bibliotecadigital/publico/home/documento/147>>. Acesso em: 02 jun. 2018

COSTA, Amanda Cristina Santos; MARCHIORI, Patrícia Zeni. Gamificação, elementos de jogos e estratégia: uma matriz de referência. **Incid: Revista de Ciência da Informação e Documentação**, [s.l.], v. 6, n. 2, p.44-65, 2 out. 2015. Universidade de São Paulo Sistema Integrado de Bibliotecas - SIBiUSP. <http://dx.doi.org/10.11606/issn.2178-2075.v6i2p44-65>.

CHOU, Yu-kai. **Octalysis – the complete Gamification framework**. Disponível em: <<https://yukaichou.com>>. Acesso em: 14 abr. 2019.

ESPÍNDOLA, Rafaela. **O que é a gamificação e como ela funciona?** Disponível em: <<https://www.edools.com/o-que-e-gamificacao/>>. Acesso em: 03 jun. 2018.

HASSAN, A. E. **The road ahead for mining software repositories**. IEEE, 2008.

HONÓRIO, Miguel Carlos. **Scrum: Por dentro do Framework de desenvolvimento ágil mais utilizado no mundo**. 2012. Disponível em: <<https://www.devmedia.com.br/scrum-por-dentro-do-framework-de-desenvolvimento-agil-mais-utilizado-no-mundo/25121>>. Acesso em: 10 jun. 2018.

KUUTTI, Julius. Designing Gamification. **Marketing, (May)** n. May, 2013. Disponível em: . Acesso em: 30 maio. 2018.

LOPES, Cristiana Sofia da Silva. **Scrum para ambientes de software distribuído: análise crítica e estudo de casos**. 2014. 116 f. Dissertação (Mestrado) - Curso de Engenharia e Gestão de Sistemas de Informação, Universidade do Minho, Braga, 2014. Disponível em: <https://repositorium.sdum.uminho.pt/bitstream/1822/35261/1/Dissertação_Cristiana_Sofia_da_Silva_Lopes_2014.pdf>. Acesso em: 17 mar. 2018.

MELHORAMENTOS. **Michaelis**. Disponível em: <<http://michaelis.uol.com.br/>>. Acesso em: 02 jun. 2018.

NUNES, Rodrigo Dantas. A Implantação das metodologias ágeis de desenvolvimento de software scrum e extreme programming(XP): uma alternativa para pequenas empresas do setor de tecnologia da informação. **ForScience**, [s.l.], v. 4, n. 2, p.0-0, 26 fev. 2017. ForScience: Revista Científica do IFMG. <http://dx.doi.org/10.29069/forscience.2016v4n2.e117>. Disponível em: <<http://www.forscience.ifmg.edu.br/forscience/index.php/forscience/article/view/117/134>>. Acesso em: 11 mar. 2018.

PANTOJA, Ailton da Silva; PEREIRA, Luandierison Marques. Gamificação: como jogos e tecnologias podem ajudar no ensino de idiomas. Estudo de caso. **Estação Científica (unifap)**, [s.l.], v. 8, n. 1, p.111-120, 30 jan. 2018. Universidade Federal do Amapá. <http://dx.doi.org/10.18468/estcien.2018v8n1.p111-120>.

PEREIRA, Denilson José. Comparação entre Metodologias de Desenvolvimento de Software Baseadas nos Métodos RUP e XP. **Revista de Tecnologia Aplicada**, [s.l.], v. 5, n. 3, p.46-52, 24 fev. 2017. ANPAD. <http://dx.doi.org/10.21714/2237-3713rta2016v5n3p46>.

QUECONCEITO. **Jogo**. Disponível em: <<http://queconceito.com.br/jogo>>. Acesso em: 02 jun. 2018.

RIKER, Diogo. **A INFLUÊNCIA DA GAMIFICAÇÃO NO SCRUM**. 2016. Disponível em: <<http://agile.pub/assuntos-diversos/a-influencia-da-gamificacao-no-scrum/>>. Acesso em: 10 jun. 2018

ROMA, Douglas N. **Uma Ferramenta para Mineração de Dados de Projetos de Software, Criação de Redes e Cálculo de Métricas Sócio-Técnicas**. 59 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná – UTFPR. Campo Mourão, 2013.

SANTANA, F. W.; OLIVA, G. A ; DE SOUZA, C. R. B. ; GEROSA, M. A. **XFlow: An Extensible Tool for Empirical Analysis of Software Systems Evolution**. In: VIII Experimental Software Engineering Latin American Workshop, 2011, Rio de Janeiro. Anais do VIII Experimental Software Engineering Latin American Workshop. Rio de Janeiro: PUC-Rio, 2011. p. 1-12.

SCHWABER, Ken; SUTHERLAND, Jeff. **The Scrum Guide**. Disponível em: <<https://www.scrumguides.org/scrum-guide.html>>. Acesso em: 11 mar. 2018.

SILVA, Iago Moreira da. **Benefícios e malefícios na adoção de gamificação em projetos de software: Um questionário com usuários de ferramentas de gamificação**. 2017. 30 f. TCC (Graduação) - Curso de Engenharia de Software, Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal, 2017.

SILVA, Maria Goreti Simão da. **Mineração de Repositórios de Software Modelos de Previsão de Pedidos de Evolução de Software**. 2013. 121 f. Dissertação (Mestrado) - Curso de Engenharia Informática, Universidade Nova, Lisboa, 2013.

TAMEIRÃO, Nathália. **GAMIFICATION: O CONCEITO, AS VANTAGENS E APLICAÇÃO NO CONTEXTO EDUCACIONAL**. 2016. Disponível em: <<https://sambatech.com/blog/insights/gamification/>>. Acesso em: 17 abr. 2018.

VIEIRA, Denisson. **Scrum: A Metodologia Ágil Explicada de forma Definitiva**. 2014. Disponível em: <<http://www.mindmaster.com.br/scrum/>>. Acesso em: 12 mar. 2018.

WERBACH, Kevin; HUNTER, Dan. **For the win: How game thinking can revolutionize your business**. [S.l: s.n.], 2012. .9781613630228.