

Guia de Preparação para a Certificação Funifier Gamification Developer I

Introdução

Bem-vindo ao Guia de Preparação para a Certificação Funifier Gamification Developer I. Este documento foi desenvolvido para orientar profissionais que desejam conquistar a certificação Funifier Developer I, uma credencial que valida conhecimentos técnicos essenciais para implementar e integrar soluções gamificadas utilizando a plataforma Funifier.

A certificação Funifier Developer I é voltada para profissionais que já têm uma base técnica em desenvolvimento de software e desejam especializar-se na criação e configuração de soluções gamificadas, aproveitando ao máximo os módulos, API, triggers e ferramentas avançadas oferecidas pelo Funifier Studio. O exame avalia desde a configuração de módulos básicos até integrações com sistemas externos, manipulação de dados com Aggregates, desenvolvimento de widgets e aplicação de princípios básicos de segurança.

Estrutura do Guia

- [1. Visão Geral da Certificação](#)
- [2. Papel e Responsabilidades do Funifier Gamification Developer](#)
- [3. Arquitetura da Plataforma Funifier](#)
- [4. Conceitos Básicos da Plataforma Funifier](#)
- [5. Configuração e Uso dos Módulos Básicos](#)
- [6. Integração com a Funifier API \(Rest API\)](#)
- [7. Manipulação de Dados com o Endpoint Database e Aggregates](#)
- [8. Desenvolvimento de Interfaces Gráficas](#)
- [9. Automação com Triggers Simples](#)
- [10. Princípios Básicos de Segurança](#)
- [11. Logística e Políticas do Exame](#)
- [12. Manutenção da Certificação](#)

[Anexo I - Módulos Funifier](#)

[Anexo II - API Funifier](#)

[Anexo III - Aggregates Funifier](#)

[Anexo IV - Triggers Funifier](#)

1. Visão Geral da Certificação

A Certificação Funifier Gamification Developer I é destinada a desenvolvedores e profissionais técnicos que atuam diretamente na implementação, configuração e integração de soluções gamificadas usando o Funifier. Essa credencial demonstra que o candidato domina os módulos técnicos essenciais, entende a arquitetura da plataforma, sabe como automatizar processos simples com triggers, manipular dados com aggregates e construir interfaces que se conectam à API do Funifier.

Objetivos da Certificação

- Validar a capacidade de configurar e conectar módulos básicos do Funifier Studio.
- Demonstrar conhecimento prático na utilização da Funifier REST API.
- Comprovar habilidades em consultas de dados e agregações com uso do Aggregates.
- Garantir domínio básico no desenvolvimento de interfaces gráficas integradas à plataforma.
- Assegurar a compreensão de práticas básicas de segurança e publicação de soluções gamificadas.

Público-Alvo

A certificação é recomendada para:

- Desenvolvedores iniciantes ou intermediários que atuam em projetos de gamificação.
- Engenheiros de software e profissionais de TI responsáveis por integrar o Funifier a outros sistemas.
- Equipes técnicas em empresas de consultoria, marketing, recursos humanos e educação que desenvolvem soluções gamificadas.

Principais Competências Avaliadas

- Entendimento da arquitetura e infraestrutura do Funifier.
- Habilidade para configurar corretamente os módulos básicos da plataforma.

- Capacidade de integrar sistemas externos via REST API.
- Domínio de consultas e agregações usando Aggregates.
- Conhecimento prático em automação de processos simples usando triggers.
- Noções fundamentais de segurança e boas práticas de publicação e manutenção de gamificações.

2. Papel e Responsabilidades do Funifier Gamification Developer

O **Funifier Gamification Developer** é o profissional responsável por transformar os conceitos e modelos desenhados pelo time de design em soluções técnicas reais dentro da plataforma Funifier. Ele conecta a criatividade do design com a robustez técnica, garantindo que as estratégias gamificadas sejam implementadas corretamente, integradas a sistemas externos e funcionando de forma segura e eficiente.

Na certificação **Funifier Gamification Developer I**, o candidato deve demonstrar não apenas conhecimentos técnicos básicos, mas também compreensão do seu papel dentro do ciclo completo de desenvolvimento de um projeto de gamificação.

Responsabilidades do Gamification Developer

Configuração técnica no Funifier Studio

- Configurar corretamente os módulos básicos (ações, pontos, níveis, desafios, rankings, moedas virtuais, sorteios, etc.) conforme os requisitos do projeto.
- Garantir que as dependências entre módulos estejam bem definidas e funcionais.

Integração com sistemas externos

- Realizar integrações utilizando a Funifier REST API para garantir que sistemas externos (ex.: sites, aplicativos, CRMs, ERPs) possam enviar e receber dados da gamificação.
- Usar ferramentas de teste de API para validar chamadas e respostas antes da implantação.

Desenvolvimento de automações

- Criar triggers simples para automatizar lógicas de negócios, como cálculos de pontos, envio de notificações ou desbloqueio de recompensas.
- Entender a diferença entre triggers e schedulers e saber quando usar cada um.

Desenvolvimento de interfaces gráficas

- Criar widgets personalizados em HTML, CSS e JavaScript que apresentem informações relevantes ao jogador (ex.: rankings, painéis de progresso, lojas virtuais).
- Integrar essas interfaces com a API para garantir que os dados estejam atualizados em tempo real.

Garantia de segurança e boas práticas

- Configurar corretamente autenticação, permissões e escopos de acesso na API.
- Manter boas práticas de segurança no manuseio de dados, autenticação e autorização.

Testes, depuração e implantação

- Realizar testes manuais para garantir que ações e conquistas estão funcionando conforme esperado.
- Depurar problemas técnicos, identificar erros e corrigir falhas antes da publicação final.
- Participar ativamente do processo de staging e deploy para ambientes de produção.

O que não é responsabilidade direta do Developer (mas ele deve colaborar):

- Definir as estratégias e mecânicas de jogo (responsabilidade do designer).
- Levantar objetivos de negócio e personas (responsabilidade do time de design/estratégia).
- Aprovar decisões de negócio ou design (embora ele possa ajudar a validar a viabilidade técnica).

No contexto da certificação, o candidato deve demonstrar capacidade para implementar soluções gamificadas usando os módulos básicos do Funifier, integrar dados via API, desenvolver widgets simples e configurar automações que atendam às demandas funcionais do projeto.

3. Arquitetura da Plataforma Funifier

Antes de mergulharmos na estrutura técnica da Funifier, é fundamental entender o que exatamente é uma plataforma de gamification, por que ela existe e qual papel ela desempenha no contexto de transformação digital, engajamento humano e performance organizacional.

3.1. O que é uma Plataforma de Gamification

A palavra **gamification** representa a aplicação estratégica de técnicas de jogos para estimular comportamentos em contextos não relacionados a jogos, como empresas, escolas, aplicativos, comunidades ou instituições públicas. Mais do que uma ferramenta, a gamificação é uma **estratégia centrada no ser humano**.

Uma **plataforma de gamification** é um sistema tecnológico que permite que empresas gerenciem o engajamento de seus públicos — clientes, colaboradores, alunos ou parceiros — de maneira contínua, automatizada e escalável.

Ela **monitora as ações das pessoas, identifica o que as motiva e fornece feedbacks em tempo real** para guiar comportamentos na direção dos objetivos do negócio.

O que uma plataforma de gamification faz:

- Observa o comportamento individual de cada pessoa.
- Estimula ações desejadas pelo negócio (como comprar, vender, aprender, colaborar).
- Cria experiências divertidas e motivadoras com técnicas como pontos, níveis, desafios e recompensas.
- Apresenta os resultados e impactos do engajamento para o negócio.

Por que ela é importante?

- **Automatiza o engajamento humano.**
- **Torna o comportamento mensurável.**
- **Sustenta a motivação ao longo do tempo.**
- **Melhora a performance e a experiência das pessoas.**

Em vez de depender de ações pontuais de líderes ou campanhas esporádicas, uma plataforma de gamification transforma o engajamento em um **sistema de gestão contínuo**, que atua 24 horas por dia, com inteligência e precisão.

3.2. Componentes da Arquitetura Funifier

A plataforma Funifier foi projetada com foco em **escalabilidade**, **segurança** e **flexibilidade** para atender empresas de diferentes tamanhos e setores. Ela adota uma arquitetura moderna baseada em **cloud computing** e **multi-tenancy**, operando como uma solução **SaaS** (Software as a Service), o que permite que clientes utilizem suas funcionalidades pela internet, sem necessidade de instalação local.

Essa arquitetura é composta por **três elementos principais**:

► Funifier Studio

É a interface gráfica usada pelos administradores e gamificadores para criar e gerenciar gamificações. Através do Studio, é possível configurar todos os módulos da gamificação — como pontos, ações, moedas, desafios e rankings — de forma visual e intuitiva.

 Acesso: <https://my.funifier.com>

► Funifier REST API

A API é o canal de comunicação entre os sistemas externos (como sites e aplicativos) e a Engine da Funifier. Ela permite que desenvolvedores consultem e manipulem dados da gamificação, apresentem feedbacks aos jogadores (como listas de desafios e rankings) e integrem a gamificação com outros sistemas.

► Funifier Engine

É o motor central da plataforma, onde todas as regras de negócio e processamento de dados da gamificação são executadas. É a parte “invisível” para o usuário final, mas essencial para que tudo funcione. Roda em servidores de cloud como AWS, DigitalOcean, Google Cloud e Azure.

Ao logar no Studio, o usuário precisa selecionar o endereço do servidor (Engine) onde sua gamificação está hospedada. O servidor padrão é:

 <https://service2.funifier.com>

A partir dele, é possível acessar a API, por exemplo:

 <https://service2.funifier.com/v3/player> → Endpoint para consultar os jogadores.

3.3. Modelo SaaS, Multi-Tenancy e Infraestrutura de Cloud Computing

► SaaS (Software como Serviço)

O Funifier adota o modelo SaaS, no qual o software é acessado diretamente via internet, sem necessidade de instalação no computador do usuário. Isso significa que atualizações, melhorias e correções são feitas de forma centralizada e automática, sem exigir ações do cliente.

Vantagens do SaaS:

- Acesso remoto a qualquer hora e lugar.
- Atualizações automáticas e transparentes.
- Redução de custos com infraestrutura local.
- Suporte técnico e manutenção centralizada.

Pergunta relacionada:

O que significa SaaS?

R: É um modelo de distribuição de software baseado em assinatura, acessado via internet, sem necessidade de instalação local.

► Multi-Tenancy (Multilocalização)

A Funifier utiliza uma arquitetura *multi-tenant*, na qual uma única instância do software (a Engine) serve a múltiplos clientes. Cada cliente tem sua própria conta, com gamificações, dados e configurações totalmente isolados dos demais, mesmo compartilhando a mesma infraestrutura.

Vantagens do modelo Multi-Tenant:

- Redução de custos operacionais por compartilhar recursos.
- Atualizações e manutenção centralizadas, beneficiando todos os clientes.
- Escalabilidade otimizada, com uso inteligente dos recursos computacionais.
- Isolamento lógico dos dados, garantindo segurança entre clientes.

Pergunta relacionada:

O que é Multi-Tenancy?

R: É um sistema onde múltiplos clientes compartilham a mesma infraestrutura, com dados isolados logicamente.

► Cloud Computing

Toda a estrutura da Funifier é baseada em *cloud computing*, ou computação em nuvem. Isso significa que os servidores, bancos de dados e mecanismos de processamento rodam em plataformas escaláveis e seguras como AWS, Google Cloud, DigitalOcean e Azure.

Vantagens do uso de Cloud:

- Alta disponibilidade e confiabilidade.
- Escalabilidade automática para lidar com picos de acesso.
- Redução de custos com infraestrutura local.
- Facilidade de acesso global.

Pergunta relacionada:

O que é Cloud Computing?

R: É a tecnologia que permite acessar servidores, armazenamento e serviços via internet.

Esses três pilares — SaaS, Multi-Tenancy e Cloud Computing — são o alicerce que tornam o Funifier robusto, escalável, seguro e pronto para atender desde pequenos negócios até grandes corporações com múltiplas gamificações em execução simultaneamente.

3.4. Noções de Escalabilidade, Segurança e Compliance

A plataforma Funifier foi desenvolvida para oferecer um ambiente robusto e confiável, capaz de atender diferentes volumes de usuários com segurança e em conformidade com normas legais. Para isso, ela adota práticas modernas relacionadas a **escalabilidade**, **segurança** e **compliance**, que são essenciais tanto para garantir a estabilidade da operação quanto para proteger os dados dos usuários e da empresa cliente.

► Escalabilidade

Escalabilidade é a capacidade de um sistema crescer de forma eficiente à medida que a demanda aumenta — seja pelo número de usuários, transações, ou dados.

Na arquitetura da Funifier, isso é possível graças ao uso de cloud computing e à separação lógica de cada gamificação, permitindo que mais recursos computacionais sejam alocados conforme necessário.

Exemplo prático:

Se uma campanha gamificada atingir milhares de jogadores simultaneamente, a plataforma pode escalar automaticamente os recursos (memória, CPU, etc.) para manter a performance estável, sem interrupções.

Benefícios da escalabilidade:

- Suporte a picos de acesso sem perda de desempenho.
- Crescimento gradual conforme o projeto expande.
- Estabilidade da plataforma mesmo com múltiplos clientes ativos.

Pergunta relacionada:

O que é escalabilidade?

R: É a habilidade da plataforma de crescer e lidar com aumento de carga sem comprometer o desempenho.

► Segurança

A Funifier adota uma série de práticas de segurança para proteger dados sensíveis e garantir o uso adequado dos recursos da plataforma.

Principais medidas de segurança:

- **Criptografia de dados em repouso e em trânsito:** protege as informações mesmo que interceptadas.
- **Autenticação com múltiplos fatores (MFA):** adiciona uma camada extra de proteção ao login.
- **Controle de acesso baseado em funções (RBAC):** define permissões específicas conforme o papel de cada usuário (administrador, operador, etc).
- **Uso de tokens de autenticação:** acesso à API é feito com tokens seguros, com escopos bem definidos.
- **Isolamento de dados por gamificação e por cliente:** mesmo compartilhando infraestrutura, os dados de cada cliente são logicamente separados.

► Compliance

Compliance é o conjunto de políticas e práticas que asseguram que a plataforma esteja em conformidade com leis, normas e exigências regulatórias aplicáveis — como LGPD (Lei Geral de Proteção de Dados), GDPR (Regulamento Geral Europeu de Proteção de Dados) e outras normas corporativas de segurança da informação.

Práticas de compliance na Funifier:

- Armazenamento seguro e rastreável de logs de acesso.

- Consentimento para coleta de dados e transparência nos termos de uso.
- Atualizações contínuas para se alinhar a legislações internacionais.
- Monitoramento e auditoria de acessos administrativos.

Importante:

Ter compliance não é apenas uma formalidade — é um diferencial competitivo e uma exigência em empresas que tratam dados de clientes, como bancos, seguradoras e grandes varejistas.

Ao dominar esses conceitos, o desenvolvedor estará preparado para trabalhar com segurança, confiabilidade e profissionalismo, garantindo que a solução gamificada não apenas funcione bem, mas também seja sustentável e segura ao longo do tempo.

4. Conceitos Básicos da Plataforma Funifier

Antes de criar uma gamificação funcional na Funifier, é fundamental entender alguns conceitos-chave que sustentam a lógica da plataforma. Esses conceitos ajudam a diferenciar a **intenção estratégica do design** (por exemplo, técnicas de jogos) da **implementação técnica** (como os módulos), bem como o papel de cada tipo de dado e cada tipo de pessoa envolvida no processo.

4.1. Técnicas de Jogos vs. Módulos Funifier

Técnica de Jogo é o nome dado ao mecanismo psicológico que ativa uma motivação humana específica. Por exemplo, quando um jogador sobe posições em um ranking, ele sente progresso e desenvolvimento — isso ativa o *core drive* de "crescimento". O **ranking**, nesse caso, é uma técnica de jogo.

A Funifier oferece **diversas técnicas de jogos** que podem ser aplicadas aos objetivos do negócio, como: pontos, medalhas, níveis, desafios, rankings, lojas virtuais, sorteios, conquistas, entre outras.

Porém, para transformar essas técnicas em algo funcional dentro da plataforma, usamos os **módulos**.

Módulos são os componentes técnicos da Funifier usados para configurar as técnicas de jogo. Eles são como blocos de construção da gamificação.

Exemplo prático:

Para implementar um ranking semanal de jogadores com mais experiência (XP), usamos:

- O **módulo de pontos (point)** para criar o tipo de ponto "XP";
- O **módulo de leaderboard** para montar o ranking com base no XP;
- O **módulo de widget** para exibir o ranking visualmente para os jogadores.

Resumo da relação:

- Técnicas de jogos = estratégia de engajamento.
- Módulos = ferramentas para implementar essa estratégia.

4.2. Ação vs. Log de Ação

No Funifier, os conceitos de **ação** e **log de ação** são centrais para o funcionamento da plataforma.

- **Ação** é a definição do que os jogadores *podem fazer* na gamificação.
Exemplo: comprar, assistir, compartilhar, concluir, acessar, etc.
- **Log de Ação** é o registro de que uma determinada **ação foi realizada**, por quem, e em que momento.
Exemplo: "Jogador João comprou no dia 16/09/2023 às 14:30".

Esses logs são essenciais para calcular pontos, desbloquear recompensas e gerar feedbacks — pois representam o comportamento real dos jogadores.

Resumo da diferença:

- **Ação:** é a regra — o que *pode* ser feito.
- **Log de ação:** é a evidência — o que *foi* feito.

4.3. Usuário vs. Jogador

É comum confundir os papéis de "usuário" e "jogador" dentro da plataforma, mas eles têm **funções bem diferentes**:

- **Usuário** é quem acessa o **Funifier Studio** para configurar e administrar a gamificação.
Exemplos: gerentes de projeto, designers, desenvolvedores, administradores.
- **Jogador** é quem **interage com a gamificação** — ou seja, participa das missões, ganha pontos, sobe de nível, etc.
Exemplos: funcionários, clientes, alunos, representantes de vendas, etc.

Resumo da diferença:

- **Usuário:** administra a gamificação.
- **Jogador:** vive a experiência da gamificação.

4.4. Conquistas vs. Configurações de Módulos

Na plataforma Funifier, o termo **conquista** (*achievement*) representa um **reconhecimento** registrado quando o jogador atinge um marco importante. Conquistas podem acontecer ao:

- Subir de nível;
- Completar um desafio;
- Acumular uma certa quantidade de pontos;
- Comprar um item na loja virtual;
- Ganhar uma competição ou sorteio.

Já as **configurações de módulos** são as regras que definem **quando e como** essas conquistas são possíveis. Ou seja, o módulo define *a regra*, e a conquista representa *o momento em que a regra foi alcançada por alguém*.

Resumo da diferença:

- **Configuração de módulo:** Define a mecânica e a regra da gamificação.
- **Conquista:** É o marco atingido por um jogador quando a regra é cumprida.

Recapitulando os Conceitos-Chave

Conceito	Significado
Técnica de Jogo	Estratégia que ativa uma motivação humana (ex.: ranking, loja, desafio)

Módulo	Elemento técnico usado para implementar uma técnica (ex.: leaderboard, point, store)
Ação	O que o jogador pode fazer (ex.: comprar, responder, visitar)
Log de Ação	Registro de uma ação realizada, com data e jogador (ex.: “João comprou um carro...”)
Usuário	Quem administra/configura a gamificação (ex.: gestor, desenvolvedor)
Jogador	Quem participa da gamificação (ex.: colaborador, cliente, aluno)
Conquista	Reconhecimento pelo alcance de um marco (ex.: concluir missão, subir de nível, ganhar pontos)
Configuração de Módulo	A regra que define quando a conquista pode acontecer

Compreender essas diferenças é essencial para planejar e construir uma gamificação funcional e eficaz. É isso que permite ao desenvolvedor traduzir objetivos estratégicos em configurações técnicas coerentes dentro da Funifier.

5. Configuração e Uso dos Módulos Básicos

A arquitetura modular da plataforma Funifier é o que permite transformar conceitos de gamificação em experiências reais e interativas. Cada módulo representa uma peça dessa engrenagem — configurável e combinável com outras para formar sistemas completos de motivação humana.

Neste capítulo, você entenderá:

- Como os módulos estão agrupados por função dentro da plataforma.
- Para que serve cada grupo e quem são os profissionais mais indicados para operá-los.

- Quais módulos são essenciais para implementar as técnicas de jogos.
- Como os módulos se interligam para criar experiências gamificadas completas.

5.1. Grupos Funcionais de Módulos no Funifier

Os módulos da Funifier estão organizados por **assuntos** (ou grupos funcionais), que refletem as diferentes etapas de um projeto de gamificação e os diferentes perfis profissionais envolvidos. A seguir, você conhecerá cada grupo, seu propósito e os módulos que o compõem:

5.1.1. Configuração Básica

Este grupo é o ponto de partida da gamificação. Define **quem vai jogar, o que será monitorado e como os jogadores estão organizados**.

Principais módulos:

- **Action** – Define quais ações serão acompanhadas (ex: comprar, vender, completar tarefa).
- **Player** – Gerencia os participantes da gamificação.
- **Team** – Permite agrupar jogadores em equipes.

Usuários típicos: gestores de projeto, profissionais de negócios, responsáveis por definir o comportamento desejado.

5.1.2. Mecânicas Genéricas de Jogos

Aqui estão os módulos que dão vida às **regras do jogo**. Eles são responsáveis por definir a mecânica de progressão, recompensas, desafios e até jogos com chance ou narrativa.

Principais módulos:

- **Point, Level, Leaderboard** – Pontuação, níveis e rankings.
- **Challenge, Virtual Good, Lottery, Competition, Swap** – Desafios, recompensas, sorteios, competições e trocas.
- **Question, Quiz, Mystery, Crossword, Story, Avatar, LastMile, Folder** – Diversas formas interativas de engajamento e narrativa.

Usuários típicos: gamification designers, criativos, agências, profissionais de marketing.

5.1.3. Registro de Atividades e Conquistas

Esse grupo é responsável por **rastrear as ações dos jogadores**, registrar progressos e emitir **feedbacks automáticos**.

Principais módulos:

- **Action Log** – Registra tudo o que o jogador faz.
- **Progress Log** – Monitora o avanço em relação aos objetivos.
- **Achievement** – Marca quando marcos são atingidos.
- **Notification** – Envia mensagens automáticas baseadas em eventos.
- **Technique Link** – Conecta módulos às técnicas de jogos correspondentes.

Usuários típicos: desenvolvedores, analistas técnicos e especialistas em engajamento.

5.1.4. Configurações Avançadas

Quando a lógica padrão não é suficiente, esses módulos permitem **regras personalizadas, cálculos avançados e automações baseadas em eventos**.

Principais módulos:

- **Trigger** – Executa código Java sob certas condições.
- **Scheduler** – Executa códigos em horários agendados (CRON).

Usuários típicos: desenvolvedores, arquitetos de gamificação, especialistas técnicos.

5.1.5. Interface Visual e Feedback

Módulos que controlam **como a gamificação será vista e sentida** pelo jogador, oferecendo interfaces interativas e feedbacks em tempo real.

Principais módulos:

- **Widget** – Elementos visuais para exibir rankings, progresso, recompensas, etc.
- **WebSocket** – Entrega feedbacks em tempo real para o front-end.
- **Static Repo** – Armazena interfaces gráficas em subdomínios públicos.

- **Studio Page** – Cria páginas personalizadas dentro do Funifier Studio.

Usuários típicos: designers, desenvolvedores front-end, UX specialists.

5.1.6. Integração e Conectividade

Módulos voltados para **comunicação com sistemas externos**, permitindo que a gamificação se conecte com CRMs, sites, apps e outros sistemas.

Principais módulos:

- **API Restful** – Interface para integrar com sistemas externos.
- **Request** – Ferramenta para testar chamadas à API.
- **Find** – Comandos de consulta encapsulados para simplificar integrações.
- **SDK** – Conjunto de ferramentas para desenvolvedores.

Usuários típicos: desenvolvedores de integração, arquitetos de sistemas.

5.1.7. Segurança e Controle de Acesso

Responsável por manter a gamificação segura, garantindo **autenticação, autorização e rastreabilidade**.

Principais módulos:

- **Auth, Auth Module** – Regras básicas e personalizadas de autenticação.
- **Audit** – Registro de alterações administrativas.
- **Crypt** – Criptografia de dados sensíveis.

Usuários típicos: profissionais de segurança, admins de TI.

5.1.8. Gestão e Otimização de Dados

Esses módulos oferecem ferramentas para **armazenamento, consulta, análise, backup e importação/exportação de dados**.

Principais módulos:

- **Database, Custom Object** – Acesso e criação de estruturas de dados customizadas.

- **Csv Data, Upload** – Importação/exportação de arquivos.
- **Backup, Compact** – Backup e otimização da base de dados.

Usuários típicos: analistas de dados, administradores da gamificação.

5.1.9. Ambientes e Publicação

Módulos para controlar **ambientes de teste e produção**, bem como o uso de componentes reutilizáveis.

Principais módulos:

- **Staging** – Migração entre ambientes de homologação e produção.
- **Marketplace** – Biblioteca de componentes reutilizáveis.

Usuários típicos: técnicos de implantação, devs sêniores.

5.1.10. Automação e Produtividade

Ferramentas que aumentam a produtividade por meio de **inteligência artificial, sugestões automáticas e suporte especializado**.

Principais módulos:

- **AI Agent** – Agente de IA que executa comandos via linguagem natural.
- **Specialist** – Consultor de gamificação integrado à plataforma.

Usuários típicos: gamificadores, designers, desenvolvedores que desejam mais eficiência no dia a dia.

5.2. Módulos Essenciais e Como Configurá-los

Depois de entender como os módulos da Funifier estão organizados em grupos funcionais, é hora de mergulhar nos **módulos mais utilizados** na prática e mais importantes para a prova de certificação.

Nesta seção, você aprenderá:

- A função de cada módulo essencial dentro de uma gamificação.
- Quais campos devem ser configurados.
- Como esses módulos se conectam com outros para aplicar técnicas de jogos.
- Boas práticas para uma configuração eficaz e segura.

5.2.1. Módulo Ação (Action)

O módulo **Ação (Action)** é um dos pilares da plataforma Funifier. Ele define **quais atividades os jogadores podem realizar** dentro da gamificação e **quais dados devem ser capturados** quando essas ações ocorrem.

O que é uma Ação?

Uma **ação** representa uma **atividade esperada de um jogador**, como:

- Comprar
- Vender
- Assistir a um vídeo
- Compartilhar um conteúdo
- Visitar um lugar
- Preencher um formulário

Essas ações podem ser disparadas de forma automática (via integração com outros sistemas) ou por interação direta do jogador em uma interface conectada à API.

A gestão das ações pode ser feita de duas formas:

- Pelo **Funifier Studio**: via interface gráfica acessível em </studio/action>.
- Pela **Funifier API**: utilizando o endpoint REST </v3/action>.

Tela de Configuração de Ação no Studio

Ação
Back

Image Picker

Active

Action

Sell

Id

sell

Attributes

+

×	↑ ↓	product	String
×	↑ ↓	price	Number

More

Save

Exemplo de JSON de configuração de uma Ação via API

```
{
  "_id": "sell",
  "active": true,
  "action": "Sell",
  "attributes": [
    {"name": "product", "type": "String"},
    {"name": "price", "type": "Number"}
  ]
}
```

Campos principais do módulo Action

Campo	Descrição
-------	-----------

<code>_id</code>	Identificador único da ação (ex: <code>sell</code>). Deve ser curto, sem espaços ou caracteres especiais.
<code>active</code>	Define se a ação está ativa (<code>true</code>) ou inativa (<code>false</code>). Ações inativas não podem ser registradas.
<code>action</code>	Nome amigável da ação. Será exibido nas interfaces gráficas (ex: "Sell", "Vender").
<code>attributes</code>	Lista de atributos personalizados que serão registrados junto com o log da ação.
<code>attributes.name</code>	Nome do atributo (ex: <code>product</code> , <code>price</code>). Deve seguir a mesma regra do <code>_id</code> .
<code>attributes.type</code>	Tipo de dado esperado: <code>String</code> , <code>Number</code> , ou <code>Boolean</code> .

Exemplo prático de uso

Você pode configurar uma ação chamada `sell` com os atributos `product` (nome do produto vendido) e `price` (valor da venda). Sempre que um jogador realizar essa ação, o Funifier registrará um log contendo:

- Quem fez
- O que foi feito
- Quando foi feito
- Quais foram os valores passados nos atributos

Importante

A ação **só será registrada se estiver ativa**. Se o campo "`active`" estiver como `false`, qualquer tentativa de enviar essa ação será ignorada pela plataforma.

Dicas práticas

- Use identificadores simples e consistentes (ex: `buy`, `watch_video`, `complete_challenge`).

- Evite usar espaços, acentos ou letras maiúsculas no campo `_id`.
- Crie atributos somente quando necessário, e com nomes claros.
- Ações bem definidas são fundamentais para o sucesso da gamificação, pois são a base para pontos, conquistas, notificações, níveis e rankings.

Com o módulo Ação configurado corretamente, você estabelece **os gatilhos principais da gamificação**, que serão usados por outros módulos como Pontos, Níveis, Conquistas e Rankings.

5.2.2. Módulo Jogador (Player)

O módulo **Jogador (Player)** representa as **pessoas que participarão da gamificação** — ou seja, os indivíduos que serão engajados, motivados e reconhecidos ao longo da experiência. Cada jogador possui um cadastro personalizado com dados básicos e adicionais, que podem ser utilizados tanto para exibição quanto para lógica de segmentação, progressão, feedback e recompensa.

A gestão dos jogadores pode ser feita de duas formas:

- Pelo **Funifier Studio**: via interface gráfica acessível em `/studio/player`.
- Pela **Funifier API**: utilizando o endpoint REST `/v3/player`.

O que é um Jogador?

Jogadores são os **participantes ativos da gamificação** — clientes, funcionários, alunos ou qualquer pessoa que interaja com o sistema gamificado. Seu cadastro armazena:

- Informações de identificação
- Imagem de perfil
- Grupos aos quais pertence (equipes)
- Conexões sociais (amigos)
- Dados personalizados para a estratégia de engajamento (campo `extra`)

Essas informações são fundamentais para:

- Apresentar dashboards e rankings personalizados
- Enviar mensagens segmentadas
- Atribuir pontos ou recompensas com base em perfil

- Criar dinâmicas sociais como comparações entre amigos ou desafios em grupo

Tela de Configuração de Jogador no Studio

Player



Image Picker

Login

ricardo

Name

Ricardo

Email

ricardo@funifier.com

Password

Password

More



Extra



company

Funifier

string



Friends



IMAGE

NAME

ID

TYPE



Igor

igor

player



Teams



Exemplo de JSON de Configuração de Jogador via API

```
{
  "_id": "john",
  "name": "John Travolta",
  "email": "john@funifier.com",
  "image": {
    "small": {"url": "https://a.com/a.jpg"},
    "medium": {"url": "https://a.com/a.jpg"},
    "original": {"url": "https://a.com/a.jpg"}
  },
  "teams": ["sales"],
  "friends": ["ricardo", "sandra", "igor", "marcilio"],
  "extra": {
    "country": "USA",
    "department": "IT",
    "active": true,
    "sports": ["soccer", "cycling", "surf"]
  },
  "created": 1688590645776,
  "updated": 1688930284891
}
```

Campos principais do módulo Player

Campo	Descrição
<code>_id</code>	Login único do jogador. É obrigatório e não pode conter espaços.
<code>name</code>	Nome completo ou nome de exibição do jogador.
<code>email</code>	E-mail do jogador (opcional, mas recomendável para feedbacks e notificações).
<code>image</code>	Objeto com URLs para imagens nos tamanhos pequeno, médio e original.
<code>teams</code>	Lista de IDs das equipes às quais o jogador pertence.
<code>friends</code>	Lista de logins dos amigos do jogador, usada para criar conexões sociais.

extra	Campo aberto para incluir qualquer dado adicional relevante (ex: país, cargo, departamento, interesses).
created	Data/hora de criação do cadastro (gerado automaticamente).
updated	Data/hora da última atualização do jogador (gerado automaticamente).

Campos obrigatórios

- **"_id"** e **"name"** são os únicos campos **obrigatórios** para cadastrar um jogador com sucesso.

Dicas práticas

- Use logins curtos e sem espaços para o campo **_id**.
- Importe jogadores em massa via CSV ou integração com sistemas legados.
- Aproveite o campo **extra** para incluir dados relevantes para filtros e segmentações.
- Utilize o campo **friends** para criar experiências sociais como comparações, desafios e elogios entre amigos.

5.2.3. Módulo Equipe (Team)

O módulo **Equipe (Team)** permite organizar os jogadores em **grupos estratégicos** para criar experiências sociais dentro da gamificação. Com ele, é possível configurar tanto **equipes fixas**, com membros definidos manualmente, quanto **equipes dinâmicas**, com membros calculados automaticamente com base em critérios personalizados.

O agrupamento por equipes é uma técnica muito poderosa para:

- Criar competições em grupo (ex: vendas por time, turmas de alunos).
- Aumentar o senso de pertencimento.
- Aplicar desafios ou rankings coletivos.
- Reforçar laços sociais ou departamentais dentro de uma organização.

As equipes podem ser configuradas:

- Pelo **Funifier Studio**, no caminho: **/studio/team**

- Pela **API REST**, usando o endpoint: `/v3/team`

Campos principais de configuração

Campo	Descrição
<code>_id</code>	Identificador único da equipe (ex: <code>sales</code>). Não pode conter espaços.
<code>name</code>	Nome visível da equipe (ex: "Equipe de Vendas").
<code>description</code>	Texto descritivo sobre a equipe.
<code>image</code>	Imagem representativa da equipe, com versões pequena, média e original.
<code>owner</code>	ID do jogador que representa a equipe (opcional).
<code>extra</code>	Objeto com atributos personalizados da equipe (ex: país, segmento, faixa).
<code>created</code>	Timestamp de criação (gerado automaticamente).
<code>updated</code>	Timestamp da última atualização (gerado automaticamente).

Equipe Estática

Uma equipe **estática** é aquela em que os membros são definidos manualmente.

Tela de Configuração de Equipe no Studio

Team




Image Picker




Name

Sales

Description

Team Description

Members

IMAGE	NAME	ID	TYPE	
	Igor	igor	player	<div><div>×</div><div></div></div>
	Ricardo	ricardo	player	<div><div>×</div><div></div></div>
	Sandra	sandra	player	<div><div>×</div><div></div></div>

More

Cancel

Save

Exemplo de JSON de Configuração de Equipe via API

```
{  
  "_id": "sales",  
  "name": "Sales",  
}
```

```

"description": "Sales team",
"image": {
  "small": {"url": "https://a.com/a.jpg"},
  "medium": {"url": "https://a.com/a.jpg"},
  "original": {"url": "https://a.com/a.jpg"}
},
"owner": "john",
"extra": {
  "country": "USA"
},
"created": 1688590265930,
"updated": 1689955901394
}

```

Equipe Dinâmica

Uma equipe **dinâmica** define seus membros automaticamente, com base em um comando **aggregate** que consulta a base de dados.

Exemplo de equipe dinâmica:

```

{
  "_id": "usa_team",
  "name": "USA Team",
  "description": "All players that have the extra.country field equals to USA",
  "dynamic": {
    "entity": "player",
    "aggregate": "[{"$match\":{\"extra.country\":\"USA\"}},{\"$project\":{\"_id\":1}}]"
  },
  "extra": {},
  "created": 1689956363282,
  "updated": 1689956473159
}

```

Campos do objeto **dynamic**:

- **entity**: nome da coleção onde será feita a consulta (ex: **player**).

- **aggregate**: comando MongoDB que retorna os jogadores (exige apenas o campo **_id** no retorno).

Isso é muito útil para formar equipes como:

- Jogadores de um determinado país.
- Funcionários de um departamento específico.
- Alunos que atingiram determinada nota.

Boas práticas

- Nomear equipes com identificadores claros e concisos no campo **_id**.
- Use imagens representativas para reforçar identidade visual.
- Utilize equipes dinâmicas para reduzir a manutenção manual de membros.
- O campo **extra** pode ser útil para segmentar campanhas específicas (ex: faixa etária, localização, etc.).

5.2.4. Módulo Ponto (Point)

O módulo **Ponto (Point)** permite criar **tipos de pontuação personalizados** que serão usados para medir o progresso dos jogadores em diferentes dimensões da gamificação. Pontos são uma das formas mais eficazes de fornecer **feedback imediato e contínuo**, ajudando os jogadores a entenderem o quanto estão avançando.

Os pontos funcionam como **unidades de medida do status** do jogador. Eles podem representar conhecimento, reputação, experiência, moedas ou qualquer outro valor relevante para o contexto do seu projeto.

Exemplos de tipos de pontos:

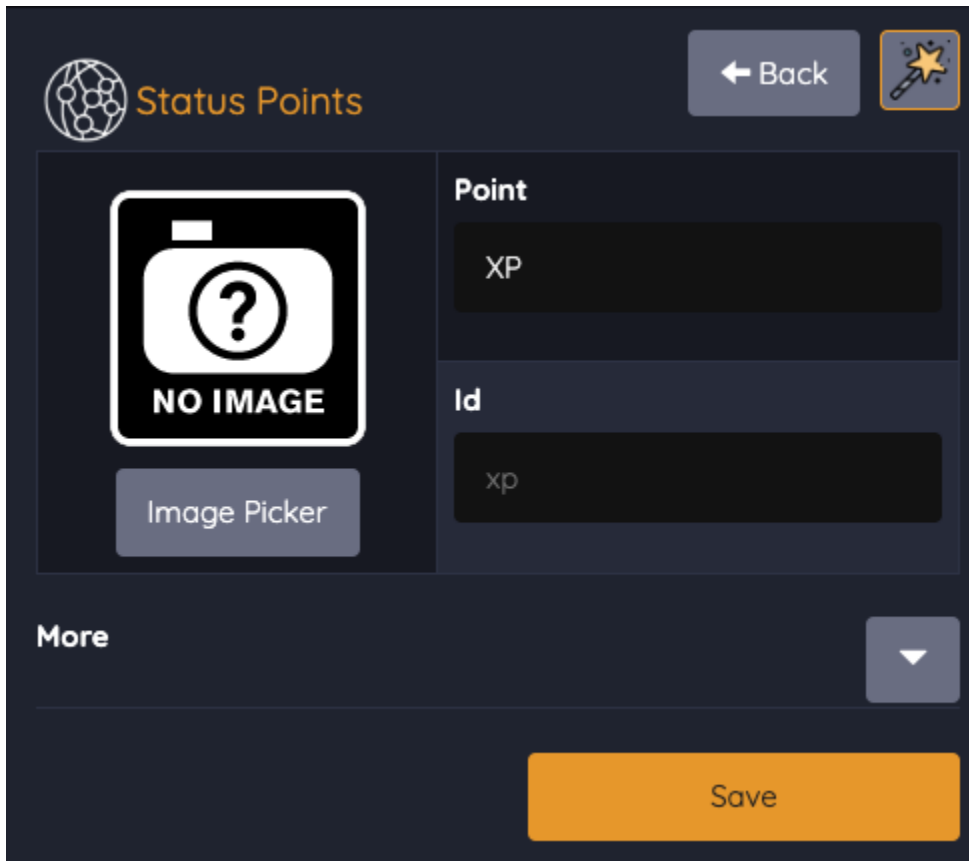
- **XP (Experience Points)**: mostra o quanto o jogador já avançou.
- **Karma**: representa reputação com base em boas ações.
- **Moedas virtuais**: que podem ser trocadas por itens.
- **Pontos por disciplina**: em projetos educacionais (ex: matemática, física, literatura).

Você pode criar **quantos tipos de pontos quiser**, e usá-los em conjunto com módulos como níveis, rankings, desafios e lojas virtuais.

Onde configurar os pontos:

- No Funifier Studio: `/studio/point`
- Via API REST: `POST /v3/point`

Tela de Configuração de Ponto no Studio



Exemplo de JSON de Configuração de Ponto via API

```
{
  "_id": "xp",
  "category": "Experience Points",
  "shortName": "XP"
}
```

Campos principais do módulo Point

Campo	Descrição
<code>_id</code>	Identificador único do ponto (ex: <code>xp</code> , <code>karma</code> , <code>moeda</code>). Não pode conter espaços nem caracteres especiais.
<code>category</code>	Nome amigável para exibição (ex: "Experience Points", "Pontos de Conhecimento").
<code>shortName</code>	Abreviação a ser usada em interfaces e widgets (ex: "XP", "KR").

Os pontos são armazenados na coleção interna `point_category`.

Boas práticas

- Crie tipos de pontos distintos para diferentes objetivos (ex: um para experiência, outro para compras).
- Evite usar nomes genéricos como “pontos1” ou “teste”. Use nomes significativos como “xp”, “karma”, “coin”.
- O `shortName` é ideal para widgets com espaço limitado — use siglas curtas e reconhecíveis.

5.2.5. Módulo Desafio (Challenge)

O módulo **Challenge** permite configurar os **desafios que os jogadores ou equipes devem cumprir** na gamificação. É uma das ferramentas mais versáteis e importantes da plataforma, usada para estimular comportamentos desejados por meio de metas claras e recompensas.

Desafios são construídos com base nas **ações realizadas pelos jogadores**, avaliadas pela Funifier Engine em tempo real. Ao cumprir os critérios de um desafio, o jogador pode receber pontos, itens, tickets para sorteios ou até conquistas visuais como badges.

Onde configurar o módulo Challenge

- No Funifier Studio: </studio/challenge>
- Via API REST: [POST /v3/challenge](#)

Técnicas de jogos implementadas com desafios

O módulo Challenge é usado para aplicar várias **técnicas de jogos clássicas**, como:

Técnica de Jogo	Implementação
Badges	Um desafio com imagem, completável uma única vez.
Quest List	Um desafio comum com um objetivo claro.
Group Quest	Um desafio configurado para ser cumprido por equipes.
Miniquest	Desafios simples com progressão rápida.

Interface no Studio – Tela de configuração de Desafios



Quest List



Image Picker

Active



Challenge

Every 10 Sales, earn 50 XP and 1 Coin

Description

The challenge involves perform the action 'Sell' from 10 in 10 times. As a reward the player receives 50 points 'XP', and 1 point

Rules

Actions that players should execute to be able to complete this challenge



Sell



%



10

times



Points

Points that will given to the player when complete this challenge



10

XP



1

Gold



More



Exemplo de JSON de Configuração de Desafios via API

```
{
  "challenge": "Every 10 Sales, earn 50 XP and 1 Coin",
  "active": true,
  "_id": "C05",
  "description": "The challenge involves perform the action 'Sell' from 10 in 10 times. As a reward the player receives 50 points 'XP', and 1 point 'Coin'.",
  "rules": [
    {
      "actionId": "sell",
      "operator": 40,
      "total": 10
    }
  ],
  "points": [
    {
      "total": 50,
      "category": "xp",
      "operation": 0
    },
    {
      "total": 1,
      "category": "coin",
      "operation": 0
    }
  ]
}
```

Campos principais na configuração de desafios

Campo	Descrição
<code>_id</code>	Identificador único do desafio. Se omitido, será gerado automaticamente.
<code>challenge</code>	Título do desafio, visível ao jogador.

description	Texto explicativo sobre o que deve ser feito e qual a recompensa.
active	Define se o desafio está ativado ou não.
rules	Conjunto de regras que o jogador precisa cumprir para completar o desafio.
points	Pontos ganhos ao completar o desafio.
filters	Condições específicas aplicadas às ações (ex: produto vendido, valor mínimo).
badge	Imagem usada para representar conquistas visuais (badges).
teamChallenge	Indica se o desafio é em grupo.
rewards	Itens ou tickets de sorteios recebidos como prêmio.
principals	IDs de jogadores ou equipes específicas autorizadas a participar.
join	Define se o jogador precisa aceitar participar e quanto tempo tem para completar o desafio.
limit	Quantas vezes o desafio pode ser concluído.

5.2.6. Módulo Nível (Level)

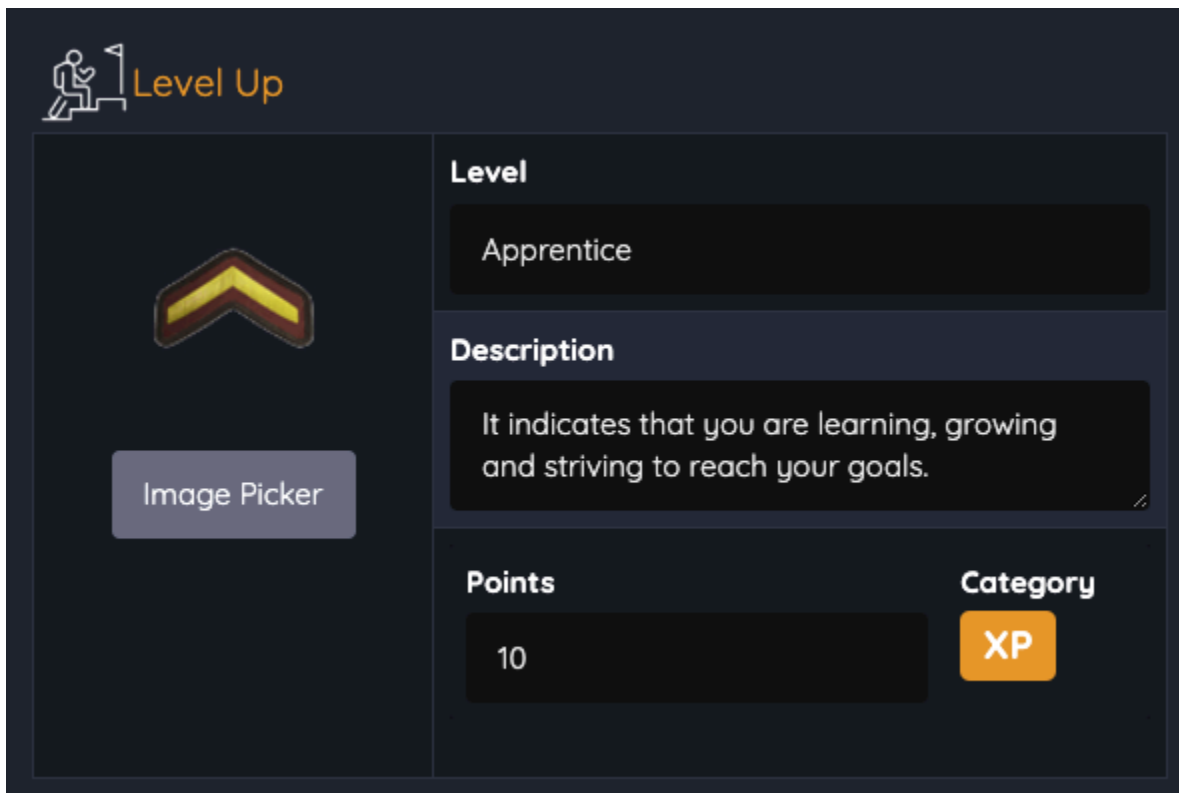
O módulo **Level** permite criar um **sistema de progressão baseado em pontos**, no qual o jogador sobe de nível conforme acumula conquistas. Ele é essencial para reforçar a **sensação de evolução** e status dentro da gamificação, funcionando como uma jornada simbólica que representa crescimento e reconhecimento.

Assim como nas artes marciais (onde passamos da faixa branca à preta), os níveis na Funifier ajudam a mostrar ao jogador que ele está avançando — incentivando-o a continuar engajado.

Onde configurar o módulo Level

- **No Funifier Studio:** </studio/level>
- **Via API REST:** `POST /v3/level`

Tela de Configuração de Nível no Studio



Level
Apprentice

Description
It indicates that you are learning, growing and striving to reach your goals.

Points	Category
10	XP

Exemplo de JSON de Configuração de um Nível via API

```
{
  "level": "Apprentice",
  "position": 0,
  "description": "It indicates that you are learning, growing and striving to reach your goals.",
  "minPoints": 10,
  "techniques": ["GT85"],
  "_id": "level0"
}
```

Campos principais do módulo Level

Campo	Descrição
<code>_id</code>	Identificador único do nível (ex: <code>level10</code>). Sem espaços ou caracteres especiais.
<code>level</code>	Nome amigável do nível, que será exibido nas interfaces para o jogador.
<code>position</code>	Índice numérico do nível dentro da progressão (ex: 0 para o primeiro nível, 1 para o segundo, etc.).
<code>minPoints</code>	Quantidade mínima de pontos necessários para atingir esse nível. Por padrão, usa o total geral de pontos.
<code>description</code>	Texto lúdico ou motivacional sobre o significado do nível.
<code>techniques</code>	Lista de códigos que associam esse nível a uma técnica de jogo (opcional).

Configuração Global de Level

Por padrão, o sistema de níveis considera a **soma total de todos os pontos** acumulados pelo jogador.

Mas você pode (e deve) configurar qual **tipo específico de ponto** será usado para avaliar o progresso nos níveis (ex: só considerar os pontos de experiência `xp`, e ignorar moedas ou karma).

Isso é feito com a seguinte configuração global:

```
{
  "_id": "global",
  "pointCategory": "xp"
}
```

Notificações Automáticas

O módulo de **níveis** permite **ativar** o envio de **notificações padrão** quando o jogador sobe de nível. Isso ocorre de forma nativa, sem necessidade de configurar manualmente.

A plataforma também permite criar **notificações personalizadas** com o módulo **Notification**, caso queira usar mensagens visuais específicas, multimídia ou segmentadas.

Boas práticas

- Mantenha a progressão de pontos **coerente e gradual** (evite saltos grandes entre níveis sem motivo).
- Use nomes e descrições que **representem conquistas reais** ou simbólicas.
- Personalize a **categoria de ponto usada nos níveis** com a configuração global, para evitar distorções na progressão.

5.2.7. Módulo Ranking (Leaderboard)

O módulo **Leaderboard** permite criar **rankings personalizados** que classificam jogadores ou equipes com base em seus comportamentos e resultados ao longo do tempo.

Rankings são uma das **técnicas de jogos mais poderosas** para estimular a competição, o reconhecimento e o desejo de melhoria contínua. Eles criam um senso de visibilidade e status entre os participantes, além de fornecer feedback comparativo.

Onde configurar o módulo Leaderboard

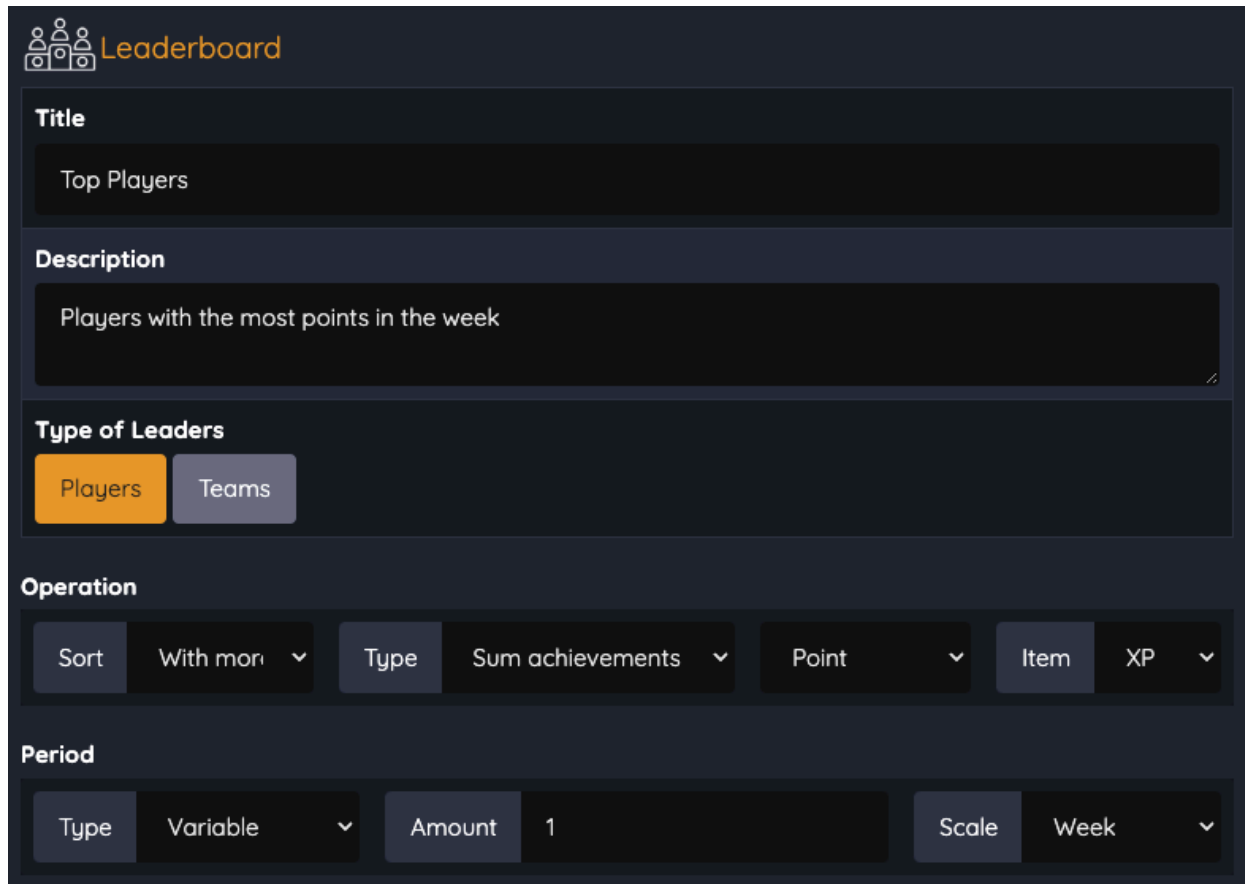
- **No Funifier Studio:** </studio/leaderboard>
- **Via API REST:** `POST /v3/leaderboard`

O que você pode fazer com um ranking:

- Exibir os **10 jogadores com mais XP da semana**.
- Mostrar a **equipe mais ativa do mês**.
- Ordenar operadores por **menor tempo médio de atendimento**.
- Ranquear jogadores com **melhor desempenho em quizzes**.

- Mostrar as pessoas que mais **convidaram amigos**, mais compraram ou mais venderam.

Tela de Configuração de Leaderboard no Studio



The screenshot shows the 'Leaderboard' configuration interface. It has a dark theme with orange and grey accents. The interface is divided into several sections:

- Title:** A text input field containing 'Top Players'.
- Description:** A text input field containing 'Players with the most points in the week'.
- Type of Leaders:** Two buttons, 'Players' (highlighted in orange) and 'Teams' (grey).
- Operation:** A row of controls including a 'Sort' button, a 'With more' dropdown, a 'Type' button, a 'Sum achievements' dropdown, a 'Point' dropdown, an 'Item' button, and an 'XP' dropdown.
- Period:** A row of controls including a 'Type' button, a 'Variable' dropdown, an 'Amount' input field with the value '1', a 'Scale' button, and a 'Week' dropdown.

Exemplo de JSON de configuração de Leaderboard via API

```
{
  "title": "Top Players",
  "description": "Players with the most points in the week",
  "principalType": 0,
  "operation": {
    "type": 3,
    "achievement_type": 0,
    "item": "xp",
    "sort": -1
  },
  "period": {
    "type": 0,
    "timeAmount": 1,

```

```
"timeScale": 6
},
"_id": "DTjTvZ5"
}
```

Explicando os principais campos

Campo	Descrição
<code>title</code>	Título do ranking exibido na interface (ex: "Top Players").
<code>description</code>	Descrição explicativa do que está sendo ranqueado.
<code>principalType</code>	Tipo de entidade: <code>0</code> para jogador, <code>1</code> para equipe.
<code>operation.type</code>	Tipo de cálculo a ser realizado (ex: soma, média, total de ações).
<code>operation.item</code>	Qual ponto ou item será usado na apuração (ex: " <code>xp</code> ").
<code>operation.sort</code>	Direção da ordenação: <code>-1</code> (maior para menor), <code>1</code> (menor para maior).
<code>period.type</code>	Tipo de período: <code>0</code> (rolling), <code>1</code> (fixo), <code>2</code> (acumulado).
<code>period.timeAmount</code>	Quantidade de tempo (ex: <code>1</code>).
<code>period.timeScale</code>	Escala de tempo: <code>5</code> (dia), <code>6</code> (semana), <code>7</code> (mês), etc.
<code>_id</code>	Identificador único do ranking (sem espaços ou acentos).

Sobre o cache e tempo real

Por padrão, os rankings são **gerados em cache**, ou seja, atualizados em intervalos programados para economizar processamento.

Mas é possível configurar rankings para **tempo real**, quando for necessário refletir mudanças instantâneas no comportamento do jogador.

Boas práticas

- Crie rankings por períodos curtos (ex: semanal) para **evitar frustração** de jogadores iniciantes.
- Use rankings para **reconhecer diferentes tipos de performance** (não só “quem tem mais pontos”).

5.2.8. Módulo Itens Virtuais (Virtual Good)

O módulo **Virtual Good** permite configurar os **objetos tangíveis ou benefícios virtuais** que os jogadores podem adquirir ou receber como recompensa dentro da gamificação.

Esses itens servem como **incentivos concretos ou simbólicos** e têm um papel fundamental para manter o engajamento, criar desejo, e dar significado às conquistas dos jogadores.


Você pode utilizar este módulo para cadastrar:

- Brindes físicos (ex: camisetas, drones, livros)
- Recompensas simbólicas (ex: troféus, títulos, avatares)
- Benefícios (ex: acesso VIP, cursos, descontos)
- Partes de itens colecionáveis
- Itens que são liberados ao completar desafios

Onde configurar o módulo Virtual Good

- **No Funifier Studio:** </studio/virtualgoods/item>
- **Via API REST:** `POST /v3/virtualgoods/item`

Tela de Configuração de Virtual Good no Studio


Loja Virtual




Image Picker

Active
☒

Name

Description

Amount How many items of this type is available in this gamification? eg. 100 or -1 for unlimited

Requirements

+

It is the cost the player must pay to purchase this item, eg. Deduct 10 coins, Verify if player is in Level Master

×	15	Point ▼	Gold ▼	Deduct ▼
---	----	---------	--------	----------

Mais

▼

Exemplo de JSON de Configuração de um Item Virtual via API

```
{
  "catalogId": "gifts",
  "name": "T-shirt",
  "description": "White t-shirt with the Funifier logo",
  "amount": 100,
  "active": true,
  "extra": {},
}
```

```

"requires": [
  {
    "total": 15,
    "type": 0,
    "item": "coin",
    "operation": 1
  }
],
"rewards": [],
"notifications": [],
"i18n": {},
"_id": "DTjVpVA"
}

```

Explicando os principais campos

Campo	Descrição
<code>catalogId</code>	Identificador da pasta ou coleção onde o item será exibido (ex: <code>"gifts"</code>).
<code>name</code>	Nome do item visível nas interfaces (ex: <code>"T-shirt"</code>).
<code>description</code>	Descrição explicativa do item.
<code>amount</code>	Quantidade de itens disponíveis no estoque (ex: <code>100</code>).
<code>active</code>	Indica se o item está disponível para compra ou resgate.
<code>requires</code>	Lista de requisitos para adquirir o item, como pontos, moedas ou outros objetos.
<code>rewards</code>	Pode conter benefícios adicionais dados ao jogador ao adquirir esse item.
<code>notifications</code>	Permite configurar mensagens automáticas ao adquirir o item.
<code>extra</code>	Campo para armazenar dados personalizados (ex: tamanho da camiseta, fornecedor).

<code>_id</code>	Identificador único do item. Não deve conter espaços.
------------------	---

Exemplos práticos

Exemplo	Descrição
Camiseta do Evento	Pode ser adquirida por 100 moedas em um catálogo de brindes.
Drone Colecionável	Requer ter dois itens anteriores: "Parte 1/2" e "Parte 2/2".
Gift Card	Pode ser concedido automaticamente ao completar um desafio.
Óculos de Sol (Virtual)	Usado para customizar o avatar do jogador.

Combinando com outros módulos

- Você pode **entregar um item** como recompensa em um **desafio** (`rewards.type = 2`).
- Pode usar **camadas de desbloqueio** — por exemplo, para adquirir o Drone, o jogador precisa ter completado outros desafios ou conquistado outras peças.

Boas práticas

- Dê nomes atraentes e use **imagens de alta qualidade** para os itens.
- Crie **catálogos segmentados** com diferentes tipos de itens (ex: roupas, eletrônicos, avatares).
- Use o campo `extra` para registrar **características específicas** como cor, tamanho ou edição limitada.
- Combine com notificações para reforçar a conquista.

5.2.9. Módulo Sorteio (Lottery)

O módulo **Lottery** permite criar **sorteios gamificados**, nos quais os jogadores concorrem a prêmios ao acumular cupons — geralmente obtidos como recompensa por completar desafios, resolver mistérios ou participar de ações específicas dentro da gamificação.

O sorteio é uma poderosa ferramenta de **incentivo comportamental**, que pode aumentar drasticamente o engajamento, principalmente quando associado a prêmios de alto valor simbólico ou material.

Onde configurar o módulo Lottery

- No Funifier Studio: </studio/lottery>
- Via API REST: [POST /v3/lottery](#)


Exemplo de sorteio na vida real

A **Caixa Econômica Federal** já utilizou sorteios como estratégia de incentivo:

A cada R\$100 em compras no cartão, o cliente ganhava 1 cupom para concorrer a uma viagem com tudo pago.

Esse tipo de mecânica pode ser replicado facilmente usando o módulo **Lottery** em conjunto com os módulos **Challenge**, **Mystery** ou **Trigger**, que distribuem os cupons.

Tela de Configuração de Sorteios no Studio

 **Loteria**

Na loteria, os apostadores compram bilhetes de loteria e quem tiver os números sorteados ganha o grande prêmio. A cada rodada alguém sempre ganha, e as recompensas são dadas a um número selecionado de vencedores, por acaso, após realizar uma ação específica.




Image Picker


Title

Travel to Cancun

Description

Travel with a companion to Cancun, with airfare and accommodation, for 7 days.

Configuration

 Draw Date

22 mai 2025, 2

Auto Execute

☐

Choice

Random

▼

Max Winners

10


Max Per Player

1

▼

Recompensas

Quando o desafio for concluído pelo jogador, o que mais ganhará como recompensa



Exemplo de configuração de um Sorteio

```
{
  "title": "Travel to Cancun",
  "description": "Travel with a companion to Cancun, with airfare and accommodation, for 7 days.",
  "drawDate": 1690824650503,
  "autoExecute": true,
  "choiceMethod": "random_ticket",
  "maxWinners": 1,
  "maxPerPlayer": 1,
  "notifications": [],
  "extra": {},
  "_id": "DTj0x5z"
}
```

Explicando os principais campos

Campo	Descrição
<code>title</code>	Título do sorteio, visível para os jogadores.
<code>description</code>	Descrição detalhada do prêmio ou do sorteio.
<code>drawDate</code>	Data e hora programada para a realização do sorteio (timestamp em milissegundos).

<code>autoExecute</code>	Define se o sorteio será executado automaticamente na data programada.
<code>choiceMethod</code>	Método de escolha dos ganhadores. O valor " <code>random_ticket</code> " indica sorteio aleatório com base nos cupons.
<code>maxWinners</code>	Número máximo de ganhadores permitidos.
<code>maxPerPlayer</code>	Quantas vezes o mesmo jogador pode ser sorteado. (1 = apenas uma vez).
<code>notifications</code>	Permite configurar mensagens automáticas para vencedores ou participantes.
<code>extra</code>	Campo para informações adicionais customizadas (ex: regras, link de regulamento).
<code>_id</code>	Identificador único do sorteio. Sem espaços ou caracteres especiais.

Como os jogadores ganham cupons?

Os cupons não são dados diretamente pelo módulo Lottery. Eles são **atribuídos por outras ações**, como:

- Completar um desafio com `rewards.type = 50`
- Resolver um mistério
- Receber por ação específica com Trigger

Isso permite criar condições criativas como:

- “Complete 5 desafios este mês e ganhe um cupom”
- “Participe de quizzes semanais e receba cupons aleatórios”
- “Troque pontos por cupons em uma loja virtual”

Boas práticas

- Utilize **descrições empolgantes** e imagens atrativas nas interfaces que exibem os sorteios.

- Combine com notificações para aumentar o impacto da premiação.
- Crie regras claras e justas — principalmente para sorteios com valor financeiro ou material relevante.
- Para sorteios recorrentes (mensais, semanais), automatize a execução com `autoExecute: true`.

O módulo **Lottery** é uma ferramenta perfeita para gerar expectativa, surpresa e recompensa em massa. Quando bem configurado, ele transforma ações simples em **oportunidades emocionantes**, incentivando o jogador a manter-se ativo e engajado na jornada.

6. Integração com a Funifier API (Rest API)

A Funifier oferece uma API REST robusta e flexível, permitindo que sistemas externos interajam diretamente com as gamificações. Por meio dessa interface, é possível cadastrar jogadores, registrar ações, consultar status, distribuir pontos, acompanhar conquistas, integrar com outros sistemas e criar experiências gamificadas dinâmicas em tempo real.

Essa API é utilizada principalmente por desenvolvedores que precisam conectar a gamificação a sites, aplicativos, CRMs, ERPs ou sistemas educacionais. Também é amplamente usada para criar dashboards, lojas virtuais, widgets personalizados, e outras interfaces voltadas aos jogadores.

6.1. Introdução à Funifier API

A **Funifier API** segue o padrão RESTful e é organizada por recursos (como jogadores, ações, pontos, desafios etc.). Cada recurso é acessado por uma rota (**endpoint**) e manipulado por meio de métodos HTTP.

Você pode usar a API para:

- Criar e gerenciar jogadores e equipes
- Registrar logs de ações
- Distribuir pontos e recompensas
- Consultar status, conquistas e rankings
- Configurar módulos remotamente
- Integrar gamificações com sistemas externos (ex: websites, apps, bots)

Todos os dados trafegam no formato **JSON** e a autenticação é obrigatória para praticamente todas as operações.

6.2. Estrutura da API e Métodos HTTP

A estrutura básica de uma chamada à API é:

`https://[engine-url]/v3/[recurso]`

Por exemplo:

- `GET https://service2.funifier.com/v3/player` → Lista os jogadores
- `POST https://service2.funifier.com/v3/player` → Cria um novo jogador

A versão mais usada atualmente é a **v3**, e a base da URL (`engine-url`) depende do servidor onde sua gamificação está hospedada (ex: `service2.funifier.com`, `eu1.service.funifier.com`, `na1.service.funifier.com`, etc.).

Principais métodos HTTP utilizados

Método	Função	Exemplo
GET	Consultar informações	<code>GET /v3/player</code>
POST	Criar ou atualizar recurso	<code>POST /v3/player</code>
DELETE	Remover recurso	<code>DELETE /v3/player/:id</code>

Exemplos práticos:

Criar jogador

`POST /v3/player`

```
{
  "_id": "player123",
  "name": "Alice",
```

```
"email": "alice@example.com"
}
```

Criar uma ação com atributos

POST /v3/action

```
{
  "action": "comprar",
  "attributes": [
    { "name": "produto", "type": "String" },
    { "name": "preco", "type": "Number" }
  ]
}
```

Registrar log de ação

POST /v3/action/log

```
{
  "actionId": "vender",
  "userId": "maria",
  "attributes": {
    "produto": "notebook",
    "preco": 2000
  }
}
```

Esses exemplos ilustram como criar recursos e interagir com os dados da gamificação usando a API.

6.3. Autenticação e Tokens

Para utilizar a API da Funifier, é obrigatório autenticar-se antes de realizar qualquer operação. A autenticação garante que apenas usuários autorizados possam acessar ou manipular dados da gamificação.

O processo de autenticação é baseado em **tokens de acesso**, utilizando padrões como **OAuth 2.0** (Bearer Token) e, em alguns casos, autenticação básica (Basic Token) para usuários do Studio.

Como funciona a autenticação na Funifier API?

A autenticação na API da Funifier ocorre em dois passos:

1. Solicitar um token de acesso

Envie uma requisição **POST** para o endpoint:

POST /auth/token

2. Utilizar esse token em todas as requisições subsequentes

Incluindo no header da requisição:

Authorization: Bearer [seu_token_de_acesso]

Exemplo de requisição para gerar um token

POST https://service2.funifier.com/auth/token

```
{
  "apiKey": "APIKEY123",
  "grant_type": "password",
  "username": "player1",
  "password": "senha123"
}
```

Explicando os campos:

Campo	Descrição
apiKey	Chave da sua gamificação (visível no Studio).
grant_type	Sempre "password" para autenticação de jogadores.
username	Login do jogador (ou usuário administrador).
password	Senha do jogador (ou usuário administrador).

Exemplo de resposta bem-sucedida:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

O campo **access_token** é o que deve ser usado nas chamadas subsequentes.

Como enviar o token nas requisições?

Adicione no cabeçalho (header) da requisição:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Erros comuns na autenticação

Código	Erro	Causa provável
401	Unauthorized	Token ausente, expirado ou inválido
403	Forbidden	Token válido, mas sem permissão suficiente
400	Invalid request / Bad Request	Dados incorretos na solicitação do token

Tipos de autenticação suportados

Tipo	Descrição
OAuth 2.0	Com token de acesso do tipo Bearer (recomendado)
Basic Token	Autenticação simples, usada em testes no Studio ou APIs internas

Resumo rápido – Checklist da autenticação

1. Obtenha a sua **apiKey** no Funifier Studio.
2. Tenha um **username** e **password** válidos (do jogador ou administrador).
3. Envie um **POST /auth/token** com as credenciais.
4. Guarde o **access_token** da resposta.
5. Em cada requisição à API, inclua no header:
Authorization: Bearer [access_token]

A autenticação é o primeiro e mais importante passo para garantir segurança, integridade dos dados e controle sobre o acesso à sua gamificação.

6.4. Manipulação de Dados com a API

A Funifier oferece uma API REST poderosa, que permite realizar todas as operações necessárias para cadastrar, atualizar, consultar e excluir dados dentro da gamificação. Toda a lógica de manipulação de dados — incluindo jogadores, ações, pontos, desafios, rankings, itens virtuais, sorteios, quizzes e muito mais — pode ser feita diretamente através da API, permitindo a integração completa com qualquer sistema externo, aplicativo mobile ou site.

Importante: A documentação oficial e completa dos endpoints da API pode ser acessada diretamente no Funifier Studio, através do link:

👉 <https://my.funifier.com/doc/api>

Nela você encontra exemplos, payloads, descrições de parâmetros e respostas para todos os endpoints.

A API segue uma estrutura baseada em **recursos**, onde cada recurso tem um endpoint específico e aceita operações via métodos HTTP como **GET**, **POST**, **PUT** e **DELETE**.

Principais operações que você pode realizar:

- **Jogadores (Player):** Criar, listar, atualizar, excluir, consultar status, conquistas, pontos, progresso, e verificar itens adquiridos.
- **Equipes (Team):** Criar, listar, atualizar, excluir, adicionar/remover membros, criar equipes dinâmicas (por critérios).
- **Ações (Action):** Cadastrar as ações que os jogadores podem executar, listar e excluir ações.
- **Logs de Ações (Action Log):** Registrar os comportamentos dos jogadores (logs) e consultar logs de ações.
- **Pontos (Point):** Definir categorias de pontos, listar pontos, deletar pontos e consultar status de pontos dos jogadores.
- **Desafios (Challenge):** Criar desafios, configurar regras, limites, recompensas (pontos, itens e tickets), listar desafios e excluir.
- **Rankings (Leaderboard):** Criar rankings, listar, consultar os líderes, resetar o cache dos rankings e deletar rankings.
- **Conquistas (Achievement):** Consultar todas as conquistas dos jogadores, tanto individuais quanto em grupo.
- **Níveis (Level):** Criar níveis, definir requisitos, listar e deletar níveis, além de consultar o progresso de nível dos jogadores.
- **Itens Virtuais (Virtual Good):** Criar catálogos e itens, listar, excluir, configurar preços, limites, realizar compras, consultar compras e estoques.
- **Sorteios (Lottery):** Criar sorteios, listar sorteios, criar e listar tickets, executar sorteios, listar ganhadores e participantes, deletar sorteios.
- **Perguntas (Question):** Criar perguntas isoladas, listar, excluir, registrar respostas (logs).

- **Quiz (Quiz):** Criar quizzes, listar, deletar, iniciar quiz, enviar respostas em lote, finalizar quiz, consultar resultados.
- **Mystery Box (Mystery):** Criar caixas de prêmios aleatórios (ex.: raspadinhas, roletas), listar, executar sorteio, e deletar.

Player (Jogadores)

São os usuários que interagem com a gamificação. Neste endpoint é possível criar o cadastro desses jogadores com quaisquer informações relevantes para estratégia de engajamento como por exemplo nome, e-mail, foto, departamento, sexo, localização geográfica, telefone, e qualquer outro atributo que seja relevante para o jogador dentro do processo de engajamento.

Criar jogador

POST /v3/player

```
{
  "_id": "player123",
  "name": "Alice",
  "email": "alice@example.com"
}
```

Listar jogadores

GET /v3/player

Consultar jogador específico

GET /v3/player/player123

Consultar status do jogador (pontos, níveis, progresso)

GET /v3/player/player123/status

Excluir jogador

DELETE /v3/player/player123

Team (Equipes)

Equipes são grupos de jogadores. Isso permite motivar os jogadores com estímulos sociais. Um jogador pode estar em mais de uma equipe se for necessário, e cada equipe pode ter um dono, que representa a equipe sempre que necessário.

Criar equipe

POST /v3/team

```
{
  "_id": "equipe_tech",
  "name": "Equipe de Tecnologia",
}
```

```
"owner": "joao"
}
```

Adicionar membro à equipe

GET /v3/team/equipe_tech/member/add/player123

Listar IDs dos membros de uma equipe

GET /v3/team/equipe_tech/memberids

Action (Ações)

Ação é a declaração do que os jogadores podem fazer na gamificação. Por exemplo, “vender”, “comprar”, “comentar”, “compartilhar”, “seguir”, “visitar”, “logar”, “reunir”, “ligar”, “estudar”, etc. No contexto de um cenário de vendas, por exemplo, você pode configurar uma ação “vender”, que pode ter atributos adicionais como “produto” e “valor” da venda.

Criar uma ação

POST /v3/action

```
{
  "action": "comprar",
  "attributes": [
    { "name": "produto", "type": "String" },
    { "name": "preco", "type": "Number" }
  ]
}
```

Listar ações configuradas

GET /v3/action

Registrar Logs de Ação (Action Log)

Criar log de ação (registrar evento realizado)

POST /v3/action/log

```
{
  "actionId": "vender",
  "userId": "maria",
  "attributes": {
    "produto": "notebook",
    "preco": 2000
  }
}
```

```
}
```

Registrar múltiplos logs de uma vez (envio em lote)

```
POST /v3/action/log/bulk
```

Point (Pontos)

Pontos são unidades de medida do status do jogador. Uma gamificação pode ter vários tipos de pontos diferentes. Em uma gamificação para escola você poderia ter pontos para cada disciplina. Por exemplo: Pontos de matemática, história, física, etc. Ao configurar os pontos, você declara a categoria do ponto, por exemplo, "xp", "karma", "moedas" ou simplesmente "pontos".

Criar categoria de ponto

```
POST /v3/point
```

```
{
  "_id": "xp",
  "category": "Experiência",
  "shortName": "XP"
}
```

Consultar status dos pontos de um jogador

```
GET /v3/player/player123/status
```

Challenge (Desafios)

Challenge são tarefas que os jogadores precisam executar. No desafio configuramos as ações que o jogador precisa executar e as conquistas que isso irá gerar para ele. Por exemplo: Podemos ter um desafio “Faça 10 Vendas para ganhar 25 pontos xp”. No desafio configuramos as regras que o jogador precisa cumprir, neste caso completar a ação “vender” 10 vezes. E os pontos que ele irá ganhar, neste caso 25 pontos da categoria xp.

Criar desafio

```
POST /v3/challenge
```

```
{
  "challenge": "Vendas Semanais",
  "rules": [
    { "actionId": "vender", "total": 10 }
  ]
}
```

Achievement (Conquistas)

Reconhecimentos que os jogadores recebem ao atingir certos marcos, como completar um desafio, subir de nível, ganhar pontos, comprar um item da loja virtual.

Listar conquistas de um jogador

GET /v3/achievement

Leaderboard (Rankings)

Leaderboard é uma classificar jogadores com base em resultados. Por exemplo: Um ranking que mostra os 10 jogadores com mais pontos acumulados na semana atual, ordenados de quem tem mais pontos para quem tem menos pontos. Para configurar um ranking você informa o título do ranking; a operação de como calcular os líderes; e o período de apuração.

Criar um ranking

POST /v3/leaderboard

```
{
  "title": "Top Players",
  "operation": {
    "type": 3,
    "achievement_type": 0,
    "item": "xp",
    "sort": -1
  },
  "period": {
    "type": 0,
    "timeAmount": 1,
    "timeScale": 6
  },
  "_id": "ranking_xp"
}
```

Consultar resultados do ranking

GET /v3/leaderboard/ranking_xp/leader/aggregate

Resetar cache do ranking

GET /v3/leaderboard/reset

Level (Níveis)

Level é um sistema de evolução para o jogador. Por exemplo: Nas artes marciais, as faixas que vão da cor branca até a cor preta representam um sistema de níveis. Você pode ter um número ilimitado de níveis na gamificação. Para cada nível você define a quantidade mínima de pontos para chegar no nível e também pode definir outros requisitos para alcançar o nível.

Listar níveis

```
GET /v3/level
```

Criar nível

```
POST /v3/level
```

```
{
  "_id": "L0",
  "level": "Apprentice",
  "position": 0,
  "description": "It indicates that you are learning, growing and striving to reach your goals.",
  "minPoints": 10,
  "techniques": ["GT85"]
}
```

Deletar nível

```
DELETE /v3/level/:id
```

Virtual Goods (Itens Virtuais)

Virtual goods é um módulo para configurar lojas onde os jogadores podem adquirir itens físicos ou virtuais dentro da gamificação. Por exemplo: Uma loja virtual onde os jogadores podem trocar pontos por souvenirs; Para configurar um item você informa o nome, descrição, quantidade disponível do item, requisitos para comprar o item e também pode definir um limite de quantos deste item o jogador pode comprar. Por exemplo: uma camiseta, que custa 15 pontos “moedas”, e pode ser comprada apenas uma vez por mês por cada jogador.

Listar catálogos

```
GET /v3/virtualgoods/catalog
```

Criar catálogo

```
POST /v3/virtualgoods/catalog
```

Deletar catálogo

```
DELETE /v3/virtualgoods/catalog/:id
```

Listar itens do catálogo

```
GET /v3/virtualgoods/item
```

Criar item

```
POST /v3/virtualgoods/item
```

```
{
  "catalogId": "gifts",
```

```
"name": "Bike",
"description": "Eletric bike",
"amount": -1,
"active": true,
"requires": [{"total": 15, "type": 0, "item": "coin", "operation": 1}],
"_id": "DTj7IVn"
}
```

Deletar item

DELETE /v3/virtualgoods/item/:id

Realizar compra de item

POST /v3/virtualgoods/purchase

```
{
  "player": "tom",
  "item": "DTj7IVn",
  "total": 1
}
```

*Retorna status **OK** se a compra for bem-sucedida ou **UNAUTHORIZED** se houver algum impedimento (saldo insuficiente, limite excedido etc.).*

Lottery (Sorteios)

Módulo para configurar sorteios. Quando o sorteio é realizado as recompensas são dadas a um número de vencedores selecionado de forma aleatória. Por exemplo: Um sorteio de uma viagem para Cancún, onde temos apenas um ganhador, e o sorteio será realizado em uma data estipulada pelo administrador. Para participar do sorteio você acumula cupons do sorteio, que podem ser conquistados por exemplo realizando algum desafio específico.

Listar sorteios

GET /v3/lottery

Criar sorteio

POST /v3/lottery

```
{
  "title": "Travel to Cancun",
  "description": "7 days trip to Cancun",
  "drawDate": 1690824650503,
  "autoExecute": true,
  "choiceMethod": "random_ticket",
  "maxWinners": 1,
  "maxPerPlayer": 1,
}
```

```
"rewards": [{"total": 1, "type": 2, "item": "flight_ticket"}],
"_id": "DTj0x5z"
}
```

Deletar sorteio

```
DELETE /v3/lottery/:id
```

Listar tickets de um sorteio

```
GET /v3/lottery/ticket?lottery=DTj0x5z
```

Criar ticket para um jogador

```
POST /v3/lottery/ticket
```

```
{
  "lottery": "DTj0x5z",
  "player": "tom"
}
```

Executar sorteio (realizar o sorteio)

```
GET /v3/lottery/DTj0x5z/execute
```

Listar ganhadores

```
GET /v3/lottery/winner?lottery=DTj0x5z
```

Listar participantes

```
GET /v3/lottery/participants?lottery=DTj0x5z
```

Question (Perguntas)

Módulo para criar perguntas para os jogadores. A apresentação da pergunta pode se dar no formato tradicional com a pergunta e as opções de resposta abaixo, mas também em formatos de mini games como por exemplo, um caça palavras, uma cruzadinha, etc. Cada pergunta pode apresentar texto, imagens, conteúdos audiovisuais, e as respostas podem ser de múltipla escolha, verdadeiro ou falso, dissertação, etc.

Listar perguntas

```
GET /v3/question
```

Criar pergunta

```
POST /v3/question
```

```
{
  "_id": "64a5b2c2d8dcca49bcf7eb6e",
  "type": "MULTIPLE_CHOICE",
  "title": "Visual Components",
}
```

```
"question": "In Funifier, what is the small visual component that displays feedback?",
"choices": [
  {"answer": "1", "label": "Points", "grade": 0},
  {"answer": "5", "label": "Widgets", "grade": 1}
],
"select": "one_answer"
}
```

Deletar pergunta

DELETE /v3/question/:id

Registrar resposta à pergunta

POST /v3/question/log

```
{
  "question": "64a5b2c2d8dcca49bcf7eb6e",
  "answer": ["5"],
  "player": "tom"
}
```

Quiz (Prova)

Módulo para agrupar várias perguntas em um bloco único a ser respondido pelos jogadores. Como por exemplo uma prova que tem várias perguntas. O administrador pode definir que uma prova vale 10 pontos, e quando o jogador responde a prova o FUNIFIER calcula o percentual de acerto, e a nota final do jogador. Estes questionários também podem ser apresentados na forma de mini games, como uma corrida espacial por exemplo.

Listar quizzes

GET /v3/database/quiz

Criar quiz

POST /v3/quiz

```
{
  "_id": "650c82fe832",
  "title": "Funifier Exam",
  "description": "Test your knowledge about Funifier.",
  "grade": 10
}
```

Deletar quiz

DELETE /v3/quiz/:id

Iniciar um quiz

POST /v3/quiz/start

```
{
  "quiz": "650c82fe832",
  "player": "tom"
}
```

Enviar respostas (em lote)

POST /v3/question/log/bulk

```
[
  {
    "quiz": "650c82fe832",
    "quiz_log": "650dc6168325771ffaa94098",
    "question": "650dc4d98325771ffaa93e5e",
    "answer": ["1"],
    "player": "tom"
  }
]
```

Finalizar quiz

POST /v3/quiz/finish

```
{
  "quiz_log": "650dc6168325771ffaa94098"
}
```

Mystery Box (Caixa Misteriosa)

Módulo para configurar prêmios aleatórios. O jogador não sabe o prêmio que irá ganhar. Por exemplo: uma raspadinha onde o jogador só descobre se ganhou algum prêmio ou não depois de raspar a cartela; um jogo de cara ou coroa, onde existe 50% de chance para cada opção; uma roda da fortuna, onde existem várias opções, algumas com prêmios legais e outras nem tanto. Você configura este módulo informando um título para caixa surpresa, as opções possíveis e a probabilidade de cada opção a ser escolhida pela plataforma, e as combinações vencedoras com seus respectivos prêmios.

Listar mystery boxes

GET /v3/mystery

Criar mystery box

POST /v3/mystery

```
{
  "title": "Heads or Tails",
  "options": [
```

```

    {"title": "Heads", "value": "heads", "probability": 0.5},
    {"title": "Tails", "value": "tails", "probability": 0.5}
  ],
  "win_chart": [
    {
      "combination": ["heads"],
      "reward": {"total": 1, "type": 0, "item": "coin"}
    }
  ],
  "_id": "64a5b464d8dcca49bcf7edd0"
}

```

Deletar mystery box

DELETE /v3/mystery/:id

Executar mystery box (realizar sorteio aleatório)

GET /v3/mystery/execute/:id?player=tom

A resposta indicará se o jogador ganhou (*status*: "WIN") ou não, além de mostrar os itens ou pontos conquistados.

Resumo das Principais Operações por Módulo

Módulo	Métodos Principais
Player	GET / POST / PUT / DELETE /v3/player GET /v3/player/:id/status GET /v3/player/:id/achievements
Team	GET / POST / DELETE /v3/team GET /v3/team/:id/memberids GET /v3/team/:id/member/add/:player
Action	GET / POST / DELETE /v3/action
Action Log	POST /v3/action/log POST /v3/action/log/bulk GET /v3/action/log
Achievement	GET /v3/achievement GET /v3/player/:id/achievements
Point	GET / POST / DELETE /v3/point

Challenge	GET / POST / DELETE /v3/challenge
Leaderboard	GET / POST / DELETE /v3/leaderboard GET /v3/leaderboard/:id/leader/aggregate GET /v3/leaderboard/reset
Level	GET / POST / DELETE /v3/level
Virtual Good	GET / POST / DELETE /v3/virtualgoods/catalog GET / POST / DELETE /v3/virtualgoods/item POST /v3/virtualgoods/purchase
Lottery	GET / POST / DELETE /v3/lottery GET /v3/lottery/ticket POST /v3/lottery/ticket GET /v3/lottery/:id/execute GET /v3/lottery/winner GET /v3/lottery/participants
Mystery Box	GET / POST / DELETE /v3/mystery GET /v3/mystery/execute/:id?player=:player
Question	GET / POST / DELETE /v3/question POST /v3/question/log
Quiz	GET / POST / DELETE /v3/quiz POST /v3/quiz/start POST /v3/question/log/bulk POST /v3/quiz/finish

Boas práticas na manipulação de dados

- **Valide tokens** antes de qualquer operação.
- Utilize **GET** para consultar e **POST** para criar dados. Nunca use **GET** para criar.
- Utilize o campo **extra** em jogadores, equipes e ações para armazenar informações personalizadas.
- Tenha cuidado com **DELETE** — ele **realmente apaga os dados** de forma irreversível.

Resumo dos conceitos avaliados na prova

- Criação e manipulação de jogadores, equipes, ações, logs e rankings.

- Correta utilização dos métodos HTTP (**GET**, **POST**, **DELETE**).
- Conhecimento sobre campos obrigatórios e suas funções.
- Uso correto dos endpoints e interpretação dos dados retornados.

7. Manipulação de Dados com o Endpoint Database e Aggregates

A plataforma Funifier oferece um poderoso endpoint chamado **database**, que permite acesso direto às coleções de dados da sua gamificação. Com esse recurso, é possível realizar operações de **consulta, inserção, atualização e remoção de registros**, além de executar **comandos de agregação (aggregates)** para gerar análises, relatórios e métricas personalizadas.

Cada gamificação na Funifier possui seu próprio banco de dados isolado, e o endpoint **/v3/database** dá acesso total a essas informações de forma segura e controlada.

Além disso, a Funifier utiliza como motor de banco de dados o **MongoDB**, e por isso a sintaxe dos comandos de agregação é exatamente a mesma utilizada pelo MongoDB. Isso significa que qualquer pessoa familiarizada com o MongoDB poderá aproveitar todo o poder desse recurso dentro da plataforma.

O uso correto do endpoint **database** e dos aggregates permite extrair insights profundos, automatizar relatórios e até criar regras de negócio personalizadas baseadas nos dados da sua gamificação.

7.1. Introdução ao Endpoint Database

O endpoint **/v3/database** permite que você interaja diretamente com qualquer coleção de dados da sua gamificação. Isso inclui tanto as coleções padrões da plataforma, como jogadores, ações, desafios, conquistas e logs, quanto **coleções personalizadas**, criadas conforme a necessidade de cada projeto.

O que é possível fazer com o Endpoint Database?

- Listar todas as coleções do banco de dados.
- Ler dados de qualquer coleção.
- Criar novos registros em qualquer coleção (inclusive objetos customizados).
- Atualizar registros existentes.
- Remover registros do banco.
- Executar consultas complexas usando pipelines de agregação (aggregates).

Quando usar esse recurso?

- Quando você precisa acessar dados além das telas padrão do Studio.
- Para criar relatórios customizados diretamente a partir do banco.
- Para alimentar dashboards externos ou sistemas terceiros.
- Para implementar regras ou verificações específicas que não estão cobertas pelos módulos tradicionais.

Exemplo de listagem de coleções existentes:

GET /v3/database/collections

Resposta exemplo:

```
[  
  "player",  
  "team",  
  "action",  
  "action_log",  
  "achievement",  
  "challenge",  
  "point_category",  
  "catalog_item",  
  ...  
]
```

Isso mostra todas as coleções disponíveis no banco da gamificação. Você pode consultar qualquer uma delas simplesmente substituindo **:collection** pelo nome da coleção desejada na URL.

7.2. Operações CRUD no Database

O endpoint **/v3/database** permite executar operações CRUD (**Create**, **Read**, **Update**, **Delete**) diretamente nas coleções da gamificação. Isso inclui tanto as coleções padrões (como **player**, **team**, **action**, **achievement**, etc.) quanto coleções personalizadas criadas pelo usuário, como por exemplo **car__c**, **req__c** ou qualquer outro objeto customizado.

Essas operações são extremamente úteis quando você precisa gerenciar dados, corrigir registros ou alimentar a gamificação com informações externas de forma programática.

Listar Dados de uma Coleção (READ)

Endpoint:

GET /v3/database/:collection

Descrição:

Retorna todos os registros da coleção informada.

Exemplo:

Listar todos os jogadores da coleção `player`:

`GET /v3/database/player`

Resposta exemplo:

```
[
  {
    "_id": "john",
    "name": "John Travolta",
    "email": "john@funifier.com",
    "teams": ["sales"],
    "extra": {"country": "USA", "department": "IT"},
    "created": {"$date": "2023-07-05T20:57:25.776Z"},
    "updated": {"$date": "2023-07-05T20:57:25.777Z"}
  }
]
```

Criar Dados em uma Coleção (CREATE)**Endpoint:**

`POST /v3/database/:collection`

Descrição:

Insere um novo documento na coleção especificada.

Exemplo:

Criar um carro na coleção `car__c`:

`POST /v3/database/car__c`

Body:

```
{
  "_id": "car001",
  "name": "Civic",
  "brand": "Honda",
  "year": 2010,
  "fuel": "gasoline",
  "price": 50000,
  "description": "Honda Civic"
}
```

Atualizar Dados de uma Coleção (UPDATE)

Endpoint:

PUT /v3/database/:collection

Descrição:

Atualiza um documento existente na coleção. É necessário informar o campo `_id` no corpo da requisição para identificar o registro que será alterado.

Exemplo:

Atualizar o preço do carro `car001`:

PUT /v3/database/car__c

Body:

```
{
  "_id": "car001",
  "name": "Civic",
  "brand": "Honda",
  "year": 2010,
  "fuel": "gasoline",
  "price": 55000,
  "description": "Honda Civic - updated"
}
```

Excluir Dados de uma Coleção (DELETE)

Endpoint:

DELETE /v3/database/:collection?q=_id:'valor_do_id'

Descrição:

Remove um documento da coleção. O filtro para exclusão é passado na URL utilizando o parâmetro `q`, geralmente com base no campo `_id`.

Exemplo:

Deletar o carro com `_id = car001`:

DELETE /v3/database/car__c?q=_id:'car001'

Inserção em Massa (BULK CREATE)

Endpoint:

POST /v3/database/:collection/bulk

Descrição:

Permite criar ou atualizar múltiplos registros de uma vez.

Exemplo:

Inserir dois carros na coleção `car__c`:

```
[
  {
    "_id": "car002",
    "name": "Corolla",
    "brand": "Toyota",
    "year": 2012,
    "fuel": "gasoline",
    "price": 70000,
    "description": "Toyota Corolla"
  },
  {
    "_id": "car003",
    "name": "Model 3",
    "brand": "Tesla",
    "year": 2021,
    "fuel": "electric",
    "price": 250000,
    "description": "Tesla Model 3"
  }
]
```

Resumo dos Métodos CRUD no Database:

Operação	Método	Endpoint	Descrição
Create	POST	<code>/v3/database/:collection</code>	Cria um novo registro
Read	GET	<code>/v3/database/:collection</code>	Lista todos os registros da coleção
Update	PUT	<code>/v3/database/:collection</code>	Atualiza um registro existente (via <code>_id</code>)
Delete	DELETE	<code>/v3/database/:collection?q=...</code>	Remove registros com filtro na URL
Bulk	POST	<code>/v3/database/:collection/bulk</code>	Inserção ou atualização em massa

Quais Coleções posso acessar?

Você pode acessar qualquer coleção padrão da Funifier (como `player`, `team`, `action_log`, `achievement`, `challenge`, `leaderboard`, `virtualgoods`, `lottery`, `mystery_box`, `quiz`, `question...`) ou qualquer coleção personalizada criada no projeto.

Atenção:

- O uso desse endpoint deve ser feito com cuidado, pois altera diretamente o banco de dados da sua gamificação.
- Recomenda-se que apenas administradores ou sistemas backend utilizem esses recursos.

7.3. Entendendo Aggregates no Funifier

O **Aggregate** é uma poderosa funcionalidade disponível na API da Funifier que permite realizar consultas avançadas, gerar relatórios, criar dashboards e calcular indicadores personalizados diretamente sobre o banco de dados da gamificação.

O funcionamento dos aggregates no Funifier é baseado na mesma lógica do **pipeline de agregação do MongoDB**, já que a plataforma utiliza MongoDB como banco de dados. Portanto, qualquer pessoa que conheça MongoDB, ou que consulte a [documentação oficial de Aggregates do MongoDB](#), estará apta a criar queries no Funifier.

Onde executar Aggregates no Funifier?

No endpoint:

`POST /v3/database/:collection/aggregate`

Você informa na URL o nome da coleção onde o aggregate será executado (ex.: `achievement`, `action_log`, `player`, `catalog_item`, `challenge`, ou qualquer coleção customizada) e, no corpo da requisição, envia a pipeline no formato JSON.

O que é uma Pipeline de Aggregates?

Uma pipeline é uma sequência de estágios, onde cada estágio transforma os dados e entrega o resultado para o próximo. Esse encadeamento permite construir consultas desde as mais simples até as mais complexas.

Principais Operadores Suportados no Funifier (Base MongoDB)

Operador	Descrição
\$match	Filtra documentos com base em critérios (equivalente ao WHERE)
\$group	Agrupa os dados por uma chave e permite fazer somas, contagens, médias, etc.

\$sort	Ordena os resultados (1 = crescente, -1 = decrescente)
\$limit	Limita o número de registros retornados
\$lookup	Faz um join com outra coleção
\$project	Define quais campos serão exibidos e permite renomeá-los
\$unwind	Desmembra arrays em múltiplas linhas
\$count	Conta o número total de registros
\$addFields	Adiciona campos calculados aos documentos
\$regex	Filtra documentos com base em padrões de texto
\$sum, \$avg, \$min, \$max, \$first, \$last	Operações dentro do \$group

Exemplo de Pipeline Simples – Total de Pontos XP do Jogador John

```
[
  { "$match": { "player": "john", "type": 0, "item": "xp" } },
  { "$group": { "_id": null, "totalXP": { "$sum": "$total" } } }
]
```

Resultado esperado:

```
[
  { "totalXP": 250 }
]
```

Exemplo Completo – Top 10 Jogadores com Mais Pontos no Mês Atual

```
[
  { "$match": { "type": 0, "item": "xp", "time": { "$gte": { "$date": "-0M-" }, "$lte": { "$date": "-0M+" } } } },
  { "$group": { "_id": "$player", "totalXP": { "$sum": "$total" } } },
  { "$sort": { "totalXP": -1 } },
  { "$limit": 10 },
  { "$lookup": { "from": "player", "localField": "_id", "foreignField": "_id", "as": "playerData" } },
  { "$unwind": "$playerData" },
  { "$project": { "playerId": "$_id", "playerName": "$playerData.name", "totalXP": 1, "_id": 0 } }
]
```

Exemplos Diversos Usando de Regex

EXEMPLO	DESCRIÇÃO
<code>name:{\$regex:'dani', \$options:'i'}</code>	Todos os nomes que possuam a palavra dani em alguma posição. Exemplo Paula Danielly por exemplo entraria no resultado.
<code>name:{\$regex:'^dani', \$options:'i'}</code>	Todos os nomes que começam com dani. Exemplo DANILA ALVES DE CARVALHO. Obs. Paula Danielly NÃO entraria no resultado.
<code>name:{\$regex:'rosa\$', \$options:'i'}</code>	Todos os nomes que terminam com rosa. Exemplo FAGNER ALEXANDER ROSA.
<code>name:{\$regex:'[0-9]', \$options:'i'}</code>	Apenas nomes que tenham algum número. Exemplo 00910734801 ou TRT 1A REGIAO
<code>name:{\$regex:'^[0-9]+\$', \$options:'i'}</code>	Apenas nomes compostos exclusivamente de números. Exemplo 00910734801
<code>name:{\$regex:'^[a-zA-Z]+\$', \$options:'i'}</code>	Apenas nomes compostos exclusivamente de letras. Exemplo Vasconcelos
<code>name:{\$regex:'^((?!DANIELE).)*\$', \$options:'i'}</code>	Todos os nomes que NÃO contenha a palavra DANIELE.

Entendendo cada estágio:

1. **\$match** → Filtra apenas conquistas de pontos XP dentro do mês atual.
2. **\$group** → Agrupa por jogador, somando seus pontos.
3. **\$sort** → Ordena do maior para o menor total de pontos.
4. **\$limit** → Pega apenas os 10 primeiros.
5. **\$lookup** → Faz um join com a coleção **player** para trazer o nome dos jogadores.
6. **\$unwind** → Desfaz o array gerado pelo lookup.
7. **\$project** → Define quais campos aparecem no resultado.

Expressões de Data no Funifier

O Funifier tem uma extensão sobre MongoDB que permite usar **expressões de datas relativas** diretamente nos filtros:

Expressão	Significado
-----------	-------------

-0d-	Início do dia atual (00:00:00)
-0d+	Fim do dia atual (23:59:59)
-1d-	Início do dia anterior
-1d+	Fim do dia anterior
-0w-	Início da semana atual
-0w+	Fim da semana atual
-0M-	Início do mês atual
-0M+	Fim do mês atual
-0y-	Início do ano atual
-0y+	Fim do ano atual

Por que usar Aggregates no Funifier?

- Gerar relatórios de desempenho dos jogadores.
- Construir dashboards customizados.
- Obter insights de negócios, como produtos mais vendidos, desafios mais concluídos, ou rankings personalizados.
- Criar automações e regras complexas sem depender de desenvolvimento externo.

7.4. Principais Exemplos de Aggregates no Funifier

Para te ajudar a entender na prática como utilizar a funcionalidade de **Aggregates** no Funifier, aqui estão alguns dos exemplos mais comuns e poderosos aplicados em projetos reais. Estes exemplos mostram como você pode criar relatórios, dashboards, e extrair insights valiosos sobre o comportamento dos jogadores, vendas, uso da plataforma, e muito mais.

Exemplo 1 – Top 10 Jogadores com Mais Pontos XP no Mês Atual

Endpoint: `POST /v3/database/achievement/aggregate`

Descrição: Este aggregate retorna os 10 jogadores que mais acumularam pontos do tipo "XP" no mês atual.

```
[
  {
    "$match": {
```

```

    "type": 0,
    "item": "xp",
    "time": {
      "$gte": { "$date": "-0M-" },
      "$lte": { "$date": "-0M+" }
    }
  },
  { "$group": { "_id": "$player", "totalXP": { "$sum": "$total" } } },
  { "$sort": { "totalXP": -1 } },
  { "$limit": 10 },
  {
    "$lookup": {
      "from": "player",
      "localField": "_id",
      "foreignField": "_id",
      "as": "playerData"
    }
  },
  { "$unwind": "$playerData" },
  {
    "$project": {
      "playerId": "$_id",
      "playerName": "$playerData.name",
      "totalXP": 1,
      "_id": 0
    }
  }
}
]

```

Exemplo 2 – Total de Jogadores que Completaram um Desafio Específico

Endpoint: [POST /v3/database/achievement/aggregate](#)

Descrição: Conta quantos jogadores únicos completaram o desafio com ID "DTo8dS3".

```

[
  { "$match": { "type": 1, "item": "DTo8dS3" } },
  { "$group": { "_id": "$player", "challenge": { "$first": "$item" } } },
  { "$group": { "_id": "$challenge", "totalPlayers": { "$sum": 1 } } },
  {
    "$lookup": {
      "from": "challenge",
      "localField": "_id",
      "foreignField": "_id",
      "as": "c"
    }
  }
]

```

```

    }
  },
  {
    "$project": {
      "_id": 1,
      "totalPlayers": 1,
      "challengeName": { "$first": "$c.challenge" }
    }
  }
}
]

```

Exemplo 3 – Média de Pontos XP dos Jogadores em Agosto de 2023

Endpoint: [POST /v3/database/achievement/aggregate](#)

Descrição: Calcula a média dos pontos XP ganhos pelos jogadores em agosto de 2023.

```

[
  {
    "$match": {
      "type": 0,
      "item": "xp",
      "time": {
        "$gte": { "$date": "2023-08-01T00:00:00.000Z" },
        "$lte": { "$date": "2023-08-31T23:59:59.999Z" }
      }
    }
  },
  { "$group": { "_id": "$player", "averageXP": { "$avg": "$total" } } }
]

```

Exemplo 4 – Top 3 Itens Mais Vendidos na Loja Virtual no Ano Atual

Endpoint: [POST /v3/database/achievement/aggregate](#)

Descrição: Consulta quais itens foram mais adquiridos no ano atual.

```

[
  {
    "$match": {
      "type": 2,
      "time": {
        "$gte": { "$date": "-0y-" },
        "$lte": { "$date": "-0y+" }
      }
    }
  },
  { "$group": { "_id": "$item", "totalSold": { "$sum": "$total" } } },
]

```

```

{
  "$lookup": {
    "from": "catalog_item",
    "localField": "_id",
    "foreignField": "_id",
    "as": "itemData"
  }
},
{
  "$project": {
    "_id": 1,
    "itemName": { "$first": "$itemData.name" },
    "totalSold": 1
  }
},
{ "$sort": { "totalSold": -1 } },
{ "$limit": 3 }
]

```

Exemplo 5 – Jogadores que Executaram Ações Ontem

Endpoint: `POST /v3/database/action_log/aggregate`

Descrição: Lista os jogadores que realizaram qualquer ação no dia anterior.

```

[
  {
    "$match": {
      "time": {
        "$gte": { "$date": "-1d-" },
        "$lte": { "$date": "-1d+" }
      }
    }
  },
  { "$group": { "_id": "$userId" } },
  {
    "$lookup": {
      "from": "player",
      "localField": "_id",
      "foreignField": "_id",
      "as": "playerData"
    }
  },
  {
    "$project": {
      "_id": 1,

```

```

    "playerName": { "$first": "$playerData.name" }
  }
}
]

```

Exemplo 6 – Jogadores com Mais de 1000 XP no Total

Endpoint: `POST /v3/database/achievement/aggregate`

Descrição: Verifica quantos jogadores possuem mais de 1000 pontos XP acumulados.

```

[
  { "$match": { "type": 0, "item": "xp" } },
  { "$group": { "_id": "$player", "totalXP": { "$sum": "$total" } } },
  { "$match": { "totalXP": { "$gte": 1000 } } },
  { "$count": "playersWithOver1000XP" }
]

```

Exemplo 7 – Listagem de Vendas de Produto "Bike" com Preço Maior que 500

Endpoint: `POST /v3/database/action_log/aggregate`

Descrição: Lista vendas específicas de bicicletas com preço igual ou superior a 500.

```

[
  {
    "$match": {
      "actionId": "sell",
      "attributes.product": "bike",
      "attributes.price": { "$gte": 500 }
    }
  }
]

```

Observação Importante:

Os comandos aggregates são executados diretamente no banco de dados da sua gamificação, portanto são extremamente poderosos e devem ser utilizados com responsabilidade, principalmente quando envolvem atualizações ou consultas em grandes volumes de dados.

7.5. Expressões de Data no Funifier – Como Utilizar Correto nos Aggregates

Ao trabalhar com **aggregates no Funifier**, você frequentemente precisará filtrar informações com base em datas, como: *"ações realizadas ontem"*, *"pontos conquistados este mês"* ou *"itens vendidos no ano passado"*.

O Funifier facilita esse tipo de consulta ao permitir o uso de **expressões de data simplificadas** no lugar de datas absolutas, eliminando a necessidade de calcular manualmente os timestamps.

Sintaxe das Expressões de Data

As expressões são compostas de **3 partes obrigatórias** e uma **opcional**:

Parte	Descrição
Sinal (+/-)	Indica se a data é relativa ao futuro (+) ou passado (-).
Unidade	Unidade de tempo: d (dia), w (semana), M (mês), y (ano).
Quantidade	Quantidade de unidades de tempo.
Precisão (+/-)	(Opcional) Se inclui um segundo sinal para arredondar para o início (-) ou final (+) da unidade.

Exemplos e Significados

Expressão	Significado
-0d-	Hoje, às 00:00:00 (início do dia).
-0d+	Hoje, às 23:59:59 (final do dia).
-1d	Exatamente 24h atrás do momento atual.
-1d-	00:00:00 de ontem.
-1d+	23:59:59 de ontem.
-0w-	Início da semana atual .
-0w+	Final da semana atual .
-1w-	Início da semana passada .
-1w+	Final da semana passada .
-0M-	Primeiro dia do mês atual às 00:00:00 .
-0M+	Último dia do mês atual às 23:59:59 .
-1M-	Início do mês passado .
-1M+	Final do mês passado .

-0y-	Início do ano atual .
-0y+	Final do ano atual .

Como Usar nas Queries Aggregate

As expressões são inseridas dentro de operadores de comparação de data, como **\$gte** (maior ou igual) e **\$lte** (menor ou igual).

Exemplo: Ações realizadas ontem

```
[
  {
    "$match": {
      "time": {
        "$gte": { "$date": "-1d-" },
        "$lte": { "$date": "-1d+" }
      }
    }
  }
]
```

Exemplo: Total de pontos XP acumulados este mês

```
[
  {
    "$match": {
      "type": 0,
      "item": "xp",
      "time": {
        "$gte": { "$date": "-0M-" },
        "$lte": { "$date": "-0M+" }
      }
    }
  },
  {
    "$group": { "_id": "$player", "totalXP": { "$sum": "$total" } }
  }
]
```

Exemplo: Itens vendidos no ano atual

```
[
  {
    "$match": {
      "type": 2,
      "time": {
```

```

    "$gte": { "$date": "-0y-" },
    "$lte": { "$date": "-0y+" }
  }
},
{
  "$group": { "_id": "$item", "totalSold": { "$sum": "$total" } }
}
]

```

Por que usar expressões de data?

- Evita erros manuais com datas fixas.
- Permite criar relatórios dinâmicos e permanentes (ex.: “sempre o mês atual”).
- Facilita automações, dashboards e notificações periódicas.

Documentação complementar

- [MongoDB Aggregation Pipeline](#)

7.6. Cuidados e Boas Práticas ao Usar Aggregates no Funifier

O uso de aggregates no Funifier é extremamente poderoso e flexível, permitindo realizar análises, relatórios avançados e manipulações de dados diretamente sobre qualquer coleção do banco de dados da gamificação. Contudo, é fundamental seguir boas práticas para garantir segurança, desempenho e estabilidade tanto da sua gamificação quanto da plataforma.

1. Segurança e Limitação de Acesso

- O endpoint `/v3/database/:collection/aggregate` permite acesso total às coleções do banco de dados da gamificação.
- Utilize este recurso com cautela, principalmente se você estiver expondo endpoints para frontends, apps ou integrações externas.
- Sempre que possível, proteja essas requisições usando servidores intermediários (backend próprio ou middleware) para evitar que usuários finais executem queries diretamente no banco.

2. Cuidado com o Volume de Dados

- Aggregates são processados diretamente sobre os dados da sua gamificação. Consultas sem filtros podem gerar grandes volumes de dados processados, impactando o desempenho.
- Sempre utilize operadores como `$match`, `$limit` ou filtros específicos nas suas consultas para reduzir a carga no banco.

- Para relatórios históricos de longo prazo, prefira criar rotinas periódicas (com **Scheduler**) que consolidam os dados e salvam os resultados em coleções próprias, em vez de executar aggregates pesados em tempo real.

3. Entenda Como Funciona o Cache do Sistema

- Aggregates não usam cache interno da plataforma, diferentemente de módulos como **Leaderboard**. Isso significa que cada aggregate executado faz leitura e processamento diretamente na base de dados no momento da requisição.
- Use aggregates para relatórios, dashboards administrativos ou análises, mas evite utilizá-los como base para feedbacks em tempo real aos jogadores (neste caso, prefira usar **Leaderboard**, **Status**, **Progress Log** ou **Achievement**).

4. Boas Práticas de Construção dos Aggregates

- Sempre comece com um **\$match** para filtrar os dados o máximo possível.
- Utilize **\$group**, **\$sum**, **\$avg**, **\$count**, **\$lookup** e **\$project** com moderação e propósito bem definido.
- Ao usar **\$lookup**, esteja ciente de que ele realiza um "join" entre coleções. Isso pode ser custoso em bases muito grandes.
- Prefira sempre limitar o resultado com **\$limit** se possível, especialmente em consultas que retornam rankings, listas ou análises temporais.

5. Armazene Resultados Quando Faz Sentido

- Se você precisa de um relatório que será consultado muitas vezes, vale considerar rodar o aggregate uma vez (manualmente ou com **Scheduler**) e armazenar os resultados em uma coleção customizada (**Custom Object**).
- Isso reduz drasticamente o custo computacional de consultas repetidas.

6. Documentação e Manutenção

- Documente seus aggregates, especialmente os mais complexos.
- Mantenha arquivos JSON ou scripts versionados com os aggregates usados na sua operação.
- Se a estrutura dos dados mudar (ex.: alteração no nome de atributos), revise os aggregates para garantir que continuem funcionando corretamente.

7. Tenha Atenção com Expressões de Data

- As expressões de data do Funifier (**-0d-**, **-0M+**, etc.) são extremamente úteis, mas precisam ser usadas corretamente.
- Um erro comum é esquecer de definir corretamente o intervalo, o que pode resultar em aggregates muito pesados por abranger um período maior do que o necessário.

8. Evite Usar para Funções que Já Existem Prontas

- Para cálculos de status, progresso ou conquistas, utilize os endpoints específicos da plataforma como `/v3/player/:id/status`, `/v3/player/:id/achievements` ou `Leaderboard`.
- Use aggregates quando realmente precisar de consultas específicas, personalizadas ou cruzamento de dados que os endpoints padrões não atendem.

8. Desenvolvimento de Interfaces Gráficas

A plataforma Funifier oferece uma API completa que permite aos desenvolvedores criar qualquer tipo de interface gráfica, utilizando a linguagem de desenvolvimento de sua preferência. Toda a lógica de gamificação fica centralizada na Funifier Engine, enquanto a apresentação e a experiência do usuário podem ser personalizadas e distribuídas em sites, aplicativos, portais internos, dashboards ou qualquer outro ambiente.

A criação das interfaces é feita através de chamadas para a **Funifier API (REST API)**, e para facilitar esse processo, a própria plataforma disponibiliza uma ferramenta chamada **Request**, acessível diretamente no **Studio** (<https://my.funifier.com/studio/request>). Essa ferramenta permite não só testar requisições, como também gerar automaticamente o código de integração em diferentes linguagens, como Java (Unirest), Javascript (AngularJS, jQuery) e outras.

Além disso, a plataforma Funifier conta com um sistema de **Widgets**, que são blocos de interface compostos por **HTML, CSS e JavaScript**, encapsulados e hospedados na própria Funifier. Esses widgets podem ser facilmente incorporados em qualquer sistema web, facilitando a construção de elementos como rankings, dashboards de status, listas de desafios, lojas virtuais e muito mais.

Seja criando interfaces do zero, utilizando chamadas diretas na API, ou desenvolvendo componentes reutilizáveis na forma de Widgets, os desenvolvedores têm total liberdade para criar experiências customizadas, responsivas e integradas aos objetivos da gamificação.

8.1. Estrutura das Interfaces na Funifier: Opções e Possibilidades

Ao desenvolver interfaces para uma gamificação na Funifier, o desenvolvedor possui total flexibilidade para escolher como deseja entregar a experiência visual ao usuário. Existem basicamente **duas abordagens principais**:

1. Interfaces Personalizadas usando a API

- O desenvolvedor cria toda a interface gráfica utilizando qualquer linguagem ou framework frontend (HTML, CSS, JavaScript, React, Angular, Vue, Next, etc.).

- A interface se comunica diretamente com a **Funifier REST API**, consumindo endpoints para operações como autenticação, registros de ações, consulta de status, pontos, desafios, rankings, loja, sorteios e muito mais.
- Este modelo é ideal para quem deseja criar experiências 100% customizadas, responsivas e alinhadas com o design da aplicação ou site.

2. Uso de Widgets Nativos da Funifier

- Widgets são blocos de interface prontos, compostos por HTML, CSS e JavaScript, hospedados na própria Funifier.
- Eles podem ser embutidos em qualquer sistema web, bastando inserir um pequeno script.
- A manutenção visual e lógica do widget pode ser feita dentro do próprio **Studio Funifier**, sem necessidade de alterações no código do sistema externo.
- Ideal para acelerar o desenvolvimento, garantindo padronização e facilidade na manutenção.

Liberdade Total no Desenvolvimento

- Se você domina HTML e JavaScript, pode construir qualquer coisa.
- Se você prefere frameworks como Angular, React ou Vue, a API te entrega todos os dados.
- Se deseja uma solução prática, pode usar Widgets.
- Não importa o caminho, o backend da gamificação estará sempre rodando na Funifier Engine, oferecendo dados atualizados, regras de negócio e persistência dos registros.

Exemplos de Aplicações que Podem ser Construídas:

- Portais de colaboradores com dashboards gamificados.
- Aplicativos mobile de incentivo.
- Sistemas de loja de recompensas internas.
- Rankings públicos de desafios.
- Áreas de perfil com acompanhamento de progresso.
- Jogos simples, simuladores e experiências interativas.

8.2. Como Usar a Ferramenta Request para Gerar Código de Integração

A Funifier disponibiliza, dentro do **Studio**, uma ferramenta extremamente útil chamada **Request**, localizada no menu principal. Essa ferramenta foi criada para simplificar a vida dos desenvolvedores, permitindo que eles testem qualquer endpoint da API e, além disso, gerem automaticamente o código de integração em diferentes linguagens de programação.

Essa funcionalidade não só acelera o desenvolvimento, como também elimina erros comuns que podem ocorrer na construção manual das requisições, especialmente na montagem dos headers de autenticação, corpo da requisição (payload) e parâmetros.

Principais Funcionalidades da Ferramenta Request

- Testar qualquer endpoint da API da Funifier.
- Visualizar de forma clara a URL, os headers, o body e a resposta da API.
- Gerar automaticamente o código da requisição em diferentes linguagens.
- Validar se os dados estão sendo enviados corretamente.
- Visualizar a resposta da API de forma formatada (JSON prettify).
- Depurar chamadas antes de implementar na aplicação real.

Localização da Ferramenta Request no Studio

- Acesse: <https://my.funifier.com/studio/request>
- No menu lateral esquerdo, clique em **Request**.

Passos para Utilizar a Ferramenta Request

1. **Selecione o Método HTTP:**
 - GET, POST, PUT ou DELETE.
2. **Informe o Endpoint:**
 - Exemplo: `/v3/player` para criar um jogador ou `/v3/action/log` para registrar uma ação.
3. **Preencha os Headers:**
 - A aba de Headers já traz campos automáticos, como:
 - **Authorization**: onde você informa seu token (Basic ou Bearer).
 - **Content-Type**: geralmente `application/json`.
4. **Monte o Request Body (quando necessário):**
 - Exemplo de JSON para criação de um jogador:

```
{
  "_id": "john",
  "name": "John Travolta",
  "email": "john@example.com"
}
```
5. **Execute a Requisição:**
 - Clique no botão **Send Request** e veja a resposta retornada da API.
6. **Aba CODE:**
 - Após enviar a requisição, acesse a aba **Code**.
 - Aqui você verá o código gerado automaticamente em várias linguagens, como:
 - **Java (Unirest)**
 - **JavaScript (jQuery)**
 - **JavaScript (AngularJS)**

- Node.js (Axios)
- Shell (cURL)

Exemplo Gerado – Registro de uma Ação (Action Log)

Selecionando na aba Code – JavaScript (jQuery):

```
var settings = {
  "url": "https://service2.funifier.com/v3/action/log",
  "method": "POST",
  "timeout": 0,
  "headers": {
    "Authorization": "Basic YOUR_TOKEN_HERE",
    "Content-Type": "application/json"
  },
  "data": JSON.stringify({
    "actionId": "sell",
    "userId": "john",
    "attributes": {
      "product": "bike",
      "price": 1200
    }
  })
};

$.ajax(settings).done(function (response) {
  console.log(response);
});
```

Selecionando na aba Code – Java (Unirest):

```
HttpResponse<String> response = Unirest.post("https://service2.funifier.com/v3/action/log")
  .header("Authorization", "Basic YOUR_TOKEN_HERE")
  .header("Content-Type", "application/json")
  .body("{\"actionId\":\"sell\",\"userId\":\"john\",\"attributes\":{\"product\":\"bike\",\"price\":1200}}")
  .asString();
```

Vantagens de Usar a Aba Code

- Economiza tempo na construção das integrações.
- Reduz erros de sintaxe e autenticação.
- Serve como documentação viva para os desenvolvedores.
- Perfeito para criar snippets e incluir no seu projeto rapidamente.

DICA PRO

Se você está criando um widget ou uma interface customizada, pode testar todas as chamadas na aba **Request**, validar a resposta, e depois simplesmente copiar o código pronto da aba **Code** para utilizar na sua aplicação.

8.3. Introdução aos Widgets Funifier: O que são e como funcionam

Os **Widgets Funifier** são componentes gráficos prontos que podem ser inseridos diretamente em qualquer página web. Eles são responsáveis por exibir de forma visual informações da gamificação, como rankings, status do jogador, lista de desafios, loja virtual, entre outros elementos interativos.

Cada widget é composto por três partes fundamentais:

- **HTML:** Estrutura visual.
- **CSS:** Estilo e aparência.
- **JavaScript:** Lógica de interação, busca de dados na API e dinâmicas de exibição.

Além disso, o Funifier permite que os widgets tenham integração com recursos externos, como bibliotecas JavaScript (AngularJS, jQuery, etc.), garantindo flexibilidade total para os desenvolvedores.

O que é um Widget no Funifier?

- Um bloco de interface que representa uma técnica de gamificação.
- Pode ser embutido dentro de qualquer página HTML, intranet, plataforma web ou sistema.
- É atualizado dinamicamente pelos dados da API da Funifier.
- Pode ser facilmente alterado diretamente pelo **Studio**, sem precisar acessar ou alterar o código do seu site.

Componentes de um Widget

1. **HTML:** Define a estrutura e os elementos visuais do widget.
2. **CSS:** Responsável pela aparência, como cores, tamanhos, fontes e espaçamentos.
3. **JavaScript:** Faz chamadas à API, processa os dados e exibe os resultados dinamicamente.
4. **Recursos Externos (Opcional):** Dependências como AngularJS, jQuery ou bibliotecas de gráficos.

Biblioteca JavaScript da Funifier

A Funifier fornece uma biblioteca JavaScript que simplifica a comunicação entre os widgets e a API.

Principais métodos:

- **Funifier.config.service:** Retorna a URL base da API REST.

- `Funifier.auth.getAuthorization()`: Retorna o token de autenticação ativo.
- `Funifier.widget.render({widget, selector, bind})`: Renderiza o widget especificado dentro do seletor indicado na página HTML.

Exemplo Simples – Widget de Mensagem

JavaScript:

```
function sayHello() {
  alert("Hello Funifier World!");
}
```

HTML:

```
<button onclick="sayHello()">Say Hello</button>
```

CSS:

```
button {
  padding: 10px 20px;
  border-radius: 5px;
  font-size: 16px;
  cursor: pointer;
}
```

Exemplo – Widget Status do Jogador

Este widget exibe o nome e a imagem do jogador autenticado.

JavaScript:

```
angular.module('status', []).controller('status', ['$scope', '$http', function ($scope, $http) {
  $scope.load = function() {
    $http({
      method: "GET",
      url: Funifier.config.service + '/v3/player/me',
      headers: {"Authorization": Funifier.auth.getAuthorization()}
    }).then(function(response){
      $scope.player = response.data;
    });
  };
  $scope.load();
}]);
```

```
setTimeout(function () {
  var element = angular.element(document.getElementById('status'));
  if (!element.injector()) {
    angular.bootstrap(element, ['status']);
  }
}, 10);
```

HTML:

```
<div id="status" ng-controller="status">
  <h1>
    
    {{player.name}}
  </h1>
</div>
```

CSS:

```
.profile-image {
  width: 100px;
  border-radius: 50%;
}
```

Dependências:

```
["https://ajax.googleapis.com/ajax/libs/angularjs/1.5.7/angular.min.js"]
```

8.4. Como Criar um Widget no Funifier Studio e Publicar: Passo a Passo

Criar um widget no **Funifier Studio** é uma maneira prática e poderosa de gerar componentes visuais para sua gamificação, sem a necessidade de depender exclusivamente de desenvolvedores backend. Esses widgets podem ser embutidos em qualquer site, sistema web, intranet ou plataforma que suporte HTML.

Passo 1 – Acesse a Área de Widgets

1. Acesse <https://my.funifier.com> e entre na sua gamificação.
2. No menu lateral, clique em “**Widgets**”.
3. Você verá a lista de widgets já existentes na sua gamificação.

Passo 2 – Criar um Novo Widget

1. Clique no botão “**+ Novo Widget**” no canto superior direito.
2. Preencha as informações básicas do widget:
 - **ID:** Nome único e sem espaços. Ex.: **rankingSales**.
 - **Título:** Nome descritivo. Ex.: **Ranking de Vendas**.
 - **Descrição:** (Opcional) Explique o que o widget faz.
 - **Imagem:** (Opcional) Adicione uma imagem ilustrativa do widget.

- **Técnicas de Jogo:** Relacione técnicas como **Leaderboard**, **Status**, **Store**, etc.

Passo 3 – Escrevendo o Código

O widget possui três áreas principais de desenvolvimento:

- **HTML:** Estrutura do conteúdo visual.
- **CSS:** Estilo visual do conteúdo.
- **JavaScript:** Lógica de comportamento e conexão com a API.

DICA: Você pode gerar parte do código JavaScript diretamente usando a ferramenta **Request**, na aba **“Code”**, escolhendo linguagens como AngularJS, JQuery, Java (Unirest) ou outras.

Passo 4 – Adicionar Recursos Externos (Se Necessário)

Se seu widget depende de bibliotecas externas (ex.: AngularJS, ChartJS, JQuery), inclua os links no campo **“Resources”**.

Exemplo:

```
[ "https://ajax.googleapis.com/ajax/libs/angularjs/1.5.7/angular.min.js" ]
```

Passo 5 – Adicionar Traduções (i18n)

Se seu widget possui textos fixos (como “Pontos”, “Desafios”, “Ranking”), você pode criar traduções diretamente na aba de internacionalização:

- Clique em **“+ Novo Locale”**, adicione idiomas como **en-US**, **pt-BR** ou **es-ES**.
- Informe as **chaves** e os textos em cada idioma.

Exemplo:

Key	pt-BR	en-US
title	Ranking	Ranking
pointsLabel	Pontos	Points

Passo 6 – Teste e Preview

1. Clique na aba **“Preview”**.
2. Clique em **“Run”** para ver como seu widget será renderizado.
3. Valide se tudo está funcionando corretamente, incluindo autenticação, API, dados dinâmicos e responsividade.

Passo 7 – Publicar e Instalar

Depois de salvar, seu widget está pronto para ser instalado em qualquer página web.

Exemplo de Instalação – Html Simples:

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <meta charset="UTF-8">
  <title>Ranking Vendas</title>
  <script src="https://client2.funifier.com/v3/funifier.js"></script>
</head>
<body>
  <div id="widget"></div>
  <script>
    Funifier.init({apiKey:"YOUR_API_KEY", service:"https://service2.funifier.com"}, function(){
      Funifier.auth.setAuthorization("Basic YOUR_TOKEN");
      Funifier.widget.render({widget:"rankingSales", selector:"#widget", bind:"replace"});
    });
  </script>
</body>
</html>
```

Exemplo de Instalação – Script Inline Assíncrono

```
(function() {
  var s = document.createElement('script');
  s.type = 'text/javascript';
  s.async = true;
  s.src = 'https://client2.funifier.com/v3/funifier.js';
  s.onload = function() {
    Funifier.init({apiKey:"YOUR_API_KEY", service:"https://service2.funifier.com"}, function(){
      Funifier.auth.setAuthorization("Basic YOUR_TOKEN");
      Funifier.widget.render({widget:"RankingRicardo", selector:"#widget", bind:"replace"});
    });
  };
  document.head.appendChild(s);
})();
```

Vantagens de Usar Widgets

- **Portabilidade:** Atualize o widget no Studio e ele se atualiza automaticamente em todos os sistemas onde estiver instalado.
- **Sem dependência do backend:** Toda a lógica é frontend, consumindo diretamente a API Funifier.
- **Rápida implementação:** Ideal para MVPs, testes e produção.
- **Internacionalização:** Nativo, com suporte a múltiplos idiomas.

8.5. Construindo Interfaces com Chamadas Diretas na API

Embora os **Widgets Funifier** sejam uma maneira prática e rápida de criar interfaces gráficas, eles não são obrigatórios. Você pode construir suas próprias interfaces do zero, utilizando qualquer framework, biblioteca ou linguagem de programação de sua preferência — como React, Angular, Vue.js, Next.js, Node.js, PHP, Python, ou até mesmo HTML + JavaScript puro.

Ao construir a interface diretamente, toda a comunicação com a gamificação será feita utilizando requisições HTTP para a **Funifier API (REST API)**. Isso garante total liberdade de design, comportamento e integração com seus sistemas internos ou externos.

Quando Usar Interfaces sem Widgets

- Quando deseja criar uma interface totalmente personalizada.
- Quando o projeto exige controle total sobre o design, interatividade ou experiência do usuário.
- Para integrar diretamente a gamificação em aplicativos, plataformas proprietárias, dashboards internos ou sites externos.
- Quando quer evitar dependências visuais ou técnicas do sistema de widgets.

Exemplo Prático – Registrar uma Ação

Código HTML + AngularJS

```
<!DOCTYPE html>
<html>
<head>
  <title>Funifier - Post Action Log</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0-rc.2/angular.js"></script>
  <script>
    angular.module('app', []).controller('cont', ['$scope', '$http', function ($scope, $http) {
      $scope.log = {actionId: 'sell', attributes: {}};
      $scope.result = {};

      $scope.save = function(){
        var req = {
          method: "POST",
          url: "https://service2.funifier.com/v3/action/log",
          headers: {
            "Authorization": "Basic YOUR_TOKEN_HERE",
            "Content-Type": "application/json"
          },
          data: $scope.log
        };
      };
    }];
  </script>
```

```

        $http(req).then(function(data){
            $scope.result = data.data;
        }, function(err){
            console.error(err);
        });
    };
    });
</script>
</head>
<body ng-app="app" ng-controller="cont">

    <h3>Registrar Venda</h3>
    <label>Jogador:</label> <input ng-model="log.userId" type="text"><br/>
    <label>Produto:</label> <input ng-model="log.attributes.product" type="text"><br/>
    <label>Preço:</label> <input ng-model="log.attributes.price" type="number"><br/>

    <button ng-click="save()">Enviar</button>

    <h4>Resultado</h4>
    <pre>{{result | json}}</pre>

</body>
</html>

```

Exemplo – Autenticação de Jogador

```

<!DOCTYPE html>
<html>
<head>
    <title>Funifier - Autenticação</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0-rc.2/angular.js"></script>
    <script>
        angular.module('app', []).controller('auth', ['$scope', '$http', function ($scope, $http) {
            $scope.auth = {};
            $scope.result = {};

            $scope.login = function(){
                var req = {
                    method: "POST",
                    url: "https://service2.funifier.com/v3/auth/token",
                    headers: {"Content-Type": "application/x-www-form-urlencoded"},
                    data: "apiKey=YOUR_API_KEY&grant_type=password&username="
                        + encodeURIComponent($scope.auth.username)
                        + "&password=" + encodeURIComponent($scope.auth.password)
                };
            };
        }]);
    </script>

```

```

        $http(req).then(function(data){
            $scope.result = data.data;
        }, function(err){
            console.error(err);
        });
    };
    });
</script>
</head>
<body ng-app="app" ng-controller="auth">

    <h3>Autenticação</h3>
    <label>Jogador:</label> <input ng-model="auth.username" type="text"><br/>
    <label>Senha:</label> <input ng-model="auth.password" type="password"><br/>

    <button ng-click="login()">Login</button>

    <h4>Resultado</h4>
    <pre>{{result | json}}</pre>

</body>
</html>

```

O que Você Pode Construir com Chamadas Diretas na API

- Dashboards administrativos customizados.
- Portais de jogadores com status, ranking, loja e conquistas.
- Telões de pontuação em eventos.
- Interfaces mobile utilizando React Native, Flutter, Ionic ou outra stack mobile.
- Integrações com plataformas de e-commerce, ERPs, CRMs e intranets corporativas.
- Experiências personalizadas em jogos, simuladores ou ambientes educacionais.

Vantagens de Usar Chamadas Diretas

- Total liberdade no design e UX.
- Integração mais próxima com seus sistemas existentes.
- Controle sobre otimizações, carregamento assíncrono e performance.
- Mais segurança para ambientes internos, onde o uso de widgets públicos pode não ser desejado.

8.6. Utilizando a Biblioteca JavaScript da Funifier para Facilitar Integrações

A **Funifier JavaScript API** é uma biblioteca oficial que simplifica o desenvolvimento de interfaces web integradas com a gamificação. Ela encapsula as chamadas REST da API Funifier e oferece métodos prontos para realizar tarefas comuns, como autenticar jogadores, registrar ações, consultar dados e renderizar widgets.

Com ela, você escreve menos código, acelera o desenvolvimento e mantém uma integração mais limpa, especialmente em páginas HTML, sistemas web ou SPAs (Single Page Applications).

Vantagens de Usar a Biblioteca Funifier.js

- Elimina a necessidade de configurar manualmente headers, autenticação e endpoints.
- Simplifica a autenticação dos jogadores e o controle de tokens.
- Possibilita o uso de widgets prontos ou personalizados.
- Permite registrar ações e consumir dados da gamificação de forma rápida e segura.
- Funciona em qualquer site ou sistema web, com ou sem frameworks (React, Angular, Vue, etc.).

Como Instalar a Biblioteca

Para usar, basta importar o arquivo JavaScript diretamente no HTML da sua página:

```
<script src="https://client2.funifier.com/v3/funifier.js"></script>
```

Ou, se quiser utilizar a versão minificada:

```
<script  
src="https://client2.funifier.com/v3/funifier.min.js"></script>
```

Inicializando a Biblioteca

O primeiro passo após importar é inicializar a biblioteca com sua **API Key** e o endereço do serviço Funifier.

```
Funifier.init({  
    apiKey: "YOUR_API_KEY",  
    service: "https://service2.funifier.com"  
}, function() {  
    console.log("Funifier Initialized!");  
});
```

Parâmetros opcionais:

- **language**: Define o idioma, exemplo "en-US" ou "pt-BR".
- **service**: URL do servidor Funifier.

- Callback: Executado após a inicialização.

Autenticação com Funifier.auth

A biblioteca oferece três formas de autenticação:

Método	Descrição
Basic	Token sem expiração, usando <code>client_secret</code> . Ideal para sistemas backend ou widgets restritos.
Password	Autenticação do próprio jogador, usando usuário e senha.
Client Credentials	Token com expiração, autenticando um aplicativo, ideal para integrações seguras.

Exemplo – Autenticação com Password:

```
Funifier.auth.authenticate({
  grant_type: "password",
  player: "john",
  password: "123"
}, function(err, data) {
  console.log(data); // Retorna o token
});
```

Exemplo – Autenticação com Basic:

```
Funifier.auth.authenticate({
  grant_type: "basic",
  client_secret: "YOUR_CLIENT_SECRET"
}, function(err, data) {
  console.log(data); // Retorna o token
});
```

Exemplo – Autenticação com Client Credentials:

```
Funifier.auth.authenticate({
  grant_type: "client_credentials",
  client_secret: "YOUR_CLIENT_SECRET"
}, function(err, data) {
  console.log(data); // Retorna o token
});
```

Registrando Logs de Ações – Funifier.track()

Permite enviar um registro de ação que um jogador executou. É equivalente a uma requisição POST no endpoint `/v3/action/log`.

Exemplo – Logar uma venda:

```
Funifier.track({
  action: "sell",
  player: "john", // Necessário se não estiver autenticado como
jogador
  attributes: {
    product: "bike",
    price: 1500
  }
}, function(err, data) {
  console.log(data);
});
```

Renderizando Widgets com Funifier.widget

A biblioteca permite inserir widgets diretamente na sua página com uma única linha de código:

```
Funifier.widget.render({
  widget: "RankingRicardo",
  selector: "#widget",
  bind: "replace"
}, function(err, data) {
  console.log(data);
});
```

O seletor pode ser qualquer elemento CSS (`#id`, `.class`).

Exemplo Completo – Página com Widget de Ranking

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Ranking Funifier</title>
  <script
src="https://client2.funifier.com/v3/funifier.js"></script>
</head>
<body>
```



```

<div id="ranking"></div>

<script>
  Funifier.init({
    apiKey: "YOUR_API_KEY",
    service: "https://service2.funifier.com"
  }, function() {
    Funifier.auth.setAuthorization("Basic YOUR_BASIC_TOKEN");

    Funifier.widget.render({
      widget: "RankingRicardo",
      selector: "#ranking",
      bind: "replace"
    }, function(err, data) {
      if (err) console.error(err);
      else console.log(data);
    });
  });
</script>
</body>
</html>

```

Exemplo de Upload de Arquivo

```

Funifier.upload({
  input: '#fileUpload',
  type: 'file',
  extra: {session: "docs"}
}, function (err, data) {
  if (err) {
    console.error("Erro ao fazer upload:", err);
  } else {
    console.log("Upload realizado com sucesso:", data);
  }
});

```

Métodos Existentes na Funifier JavaScript API

Objeto	Método	Descrição
--------	--------	-----------

Funifier	<code>init(options, callback)</code>	Inicializa a API, definindo API Key, service, idioma e executa callback após configuração.
Funifier.auth	<code>authenticate(options, callback)</code>	Realiza autenticação nos modos <code>basic</code> , <code>password</code> , <code>client_credentials</code> , <code>facebook</code> e <code>google</code> .
Funifier.auth	<code>getAuthorization()</code>	Recupera o token de autenticação salvo.
Funifier.auth	<code>setAuthorization(token)</code>	Define manualmente um token de autenticação.
Funifier.auth	<code>logout()</code>	Remove o token de autenticação salvo (logout local).
Funifier.track	<i>(função direta)</i>	Envia um Action Log para o endpoint <code>/v3/action/log</code> .
Funifier.upload	<i>(função direta)</i>	Envia arquivos ou imagens para a API Funifier via <code>/v3/upload/file</code> ou <code>/v3/upload/image</code> .
Funifier.widget	<code>render(options, callback)</code>	Carrega e exibe um widget na página, com HTML, CSS, JS e dependências externas.
Funifier.widget	<code>init(callback)</code>	Carrega automaticamente todos os elementos com <code>data-funifier-widget</code> na página.
Funifier.i18n	<code>get(resource, key)</code>	Busca uma chave de internacionalização dentro de um recurso (widget).
Funifier.i18n	<code>getValue(obj, key)</code>	Busca uma chave dentro de um objeto que possui o campo <code>i18n</code> .
Funifier.i18n	<code>getResource(resource)</code>	Retorna o dicionário completo de um recurso no idioma atual.
Funifier.range	<code>parse(content_range)</code>	Interpreta o header <code>Content-Range</code> das respostas para controle de paginação.

<code>Funifier.range</code>	<code>paginate(page, content_range)</code>	Gera o valor correto para o header <code>Range</code> para acessar uma página específica da API.
-----------------------------	--	--

Resumo Importante

- **Autenticação:** `Funifier.auth`
- **Registro de ações:** `Funifier.track`
- **Renderização de widgets:** `Funifier.widget`
- **Upload de arquivos e imagens:** `Funifier.upload`
- **Internacionalização:** `Funifier.i18n`
- **Paginação:** `Funifier.range`

8.7. Cuidados e Boas Práticas no Desenvolvimento de Interfaces com Funifier

Quando desenvolvemos interfaces gráficas integradas com a Funifier, seja utilizando widgets, a biblioteca JavaScript ou chamadas diretas na API, é fundamental adotar algumas boas práticas que garantem segurança, performance, manutenibilidade e uma ótima experiência para os usuários.

1. Segurança da Informação

- **Nunca exponha credenciais sensíveis no front-end.**
Se for usar autenticação do tipo **Basic** ou **Client Credentials**, limite as permissões do App nas configurações de segurança do Funifier Studio.
Se a operação for sensível (como consultar dados de muitos jogadores ou alterar registros), prefira realizar via backend.
- **Use autenticação do tipo "Password" sempre que possível para o jogador.**
Isso protege melhor os dados individuais e evita exposição indevida de tokens globais.
- **Restrinja permissões dos Apps no Funifier Studio.**
Defina exatamente quais endpoints cada app pode acessar. Isso mitiga riscos em caso de exposição do token.

2. Performance e Eficiência

- **Evite consultas desnecessárias ou repetitivas.**
Sempre que possível, utilize cache local ou sessionStorage/localStorage para

armazenar dados que não mudam com frequência, como nome do jogador, nível, status atual.

- **Utilize aggregates inteligentes.**

Ao construir widgets ou dashboards, prefira usar o endpoint `/aggregate` com filtros, `$match`, `$project` e `$limit` para receber apenas os dados necessários.

- **Atenção aos Leaderboards.**

Rankings muito extensos podem gerar sobrecarga no carregamento da interface. Utilize filtros como período (semana, mês, ano) e limite (`$limit:10`).

- **Se possível, use a geração de cache dos leaderboards no Funifier em vez de builds em tempo real.**

Isso reduz o número de consultas complexas.

3. Organização e Padronização de Código

- **Padronize IDs e nomes de widgets, variáveis e coleções.**

Use nomes claros e consistentes, evitando espaços, acentos e caracteres especiais.

Ex.: `ranking_sales_month` ao invés de `Ranking de Vendas do Mês`.

- **Mantenha o CSS dos widgets isolado.**

Utilize seletores específicos para não interferir no CSS da sua aplicação principal.

Ex.: `.fun-widget-ranking .title { }` em vez de `.title { }`.

- **Separe o JavaScript de lógica do HTML sempre que possível.**

Isso melhora a manutenção e a escalabilidade dos projetos.

4. Dependências e Recursos Externos

- **Declare corretamente todas as dependências externas.**

No campo `resources` do widget, inclua as URLs de bibliotecas como Angular, jQuery, Chart.js, etc. Isso evita erros de carregamento.

- **Evite carregar múltiplas vezes a mesma biblioteca.**

Isso pode gerar conflitos. Verifique se sua aplicação já possui uma versão do Angular, jQuery ou outra biblioteca antes de incluir novamente.

5. Controle de Erros e Logs

- **Implemente tratamento de erros nas chamadas da API.**

Sempre verifique se ocorreu erro nas funções de callback.

```
Funifier.track({action:"sell", attributes:{product:"bike"}}, function(err, data){
  if(err) {
    console.error("Erro ao registrar ação:", err);
  } else {
    console.log("Ação registrada com sucesso:", data);
  }
});
```

- **Exiba mensagens amigáveis para o usuário.**
Se uma chamada falhar, informe o usuário de forma clara, como “Erro ao carregar ranking. Tente novamente mais tarde.”

6. Responsividade e Mobile First

- Garanta que seus widgets e interfaces funcionem bem em diferentes tamanhos de tela.
- Use `flexbox` ou `grid` no CSS para criar layouts adaptáveis.
- Imagens e elementos devem ser fluidos (`max-width: 100%`).

7. Internacionalização

- Utilize o sistema de `i18n` dos widgets para criar versões em múltiplos idiomas.
- Defina chaves e traduções diretamente no Funifier Studio na aba de internacionalização do widget.

8. Documentação e Manutenção

- Documente como cada widget ou interface foi construído.
- Crie um README interno com informações como:
 - API Keys utilizadas.
 - URLs de ambiente de produção e desenvolvimento.
 - Responsáveis pela manutenção.
 - Observações importantes sobre autenticação e permissões.

9. Limitações Importantes a Considerar

- **Limite de dados nas consultas:** Evite fazer aggregates sem `$limit` ou sem filtros. Isso pode gerar consultas muito grandes.
- **Evite usar tokens do tipo Basic em ambientes públicos sem as devidas restrições de permissão.**
- **Dependência de terceiros:** Cuidado com a utilização de bibliotecas externas hospedadas em terceiros. Se a CDN cair, seu widget pode parar de funcionar.

Resumo de Boas Práticas

Categoria	Boas Práticas
Segurança	Tokens protegidos, permissões restritas, uso de autenticação adequada
Performance	Uso de cache, aggregates otimizados, limites nos dados

Organização	CSS isolado, nomes padronizados, código separado
UX/UI	Tratamento de erros, responsividade, mensagens claras
Internacionalização	Uso de i18n nos widgets
Manutenção	Documentação interna, controle de dependências e versões

9. Automação com Triggers Simples

Triggers são uma poderosa funcionalidade dentro da plataforma Funifier, permitindo automatizar processos de negócios e comportamentos da gamificação sem a necessidade de desenvolvimento externo. Elas funcionam como pequenos scripts executados automaticamente em resposta a eventos específicos do sistema, como quando um jogador realiza uma ação, conquista um desafio, compra um item ou altera seus dados no sistema.

Esses scripts são escritos em Java e são executados no servidor da Funifier, no momento em que o evento acontece, garantindo que as regras de negócio sejam aplicadas de forma consistente e imediata.

Por exemplo, você pode configurar uma trigger para dobrar os pontos de jogadores do departamento de tecnologia, criar códigos de convite automaticamente para novos cadastros, enviar notificações externas via Zapier ou até gerar bônus para amigos de quem completou um desafio importante.

Ao longo deste tópico, você aprenderá como funcionam as triggers, como configurá-las corretamente, como estruturar seus scripts, quais objetos podem ser manipulados e verá diversos exemplos práticos de aplicação no mundo real.

9.1. O que são Triggers no Funifier e Para que Servem

Triggers são trechos de código executados automaticamente em resposta a eventos que ocorrem dentro da sua gamificação. Esses eventos podem ser ações como:

- Criação de um jogador (`before_create` ou `after_create`).
- Compra de um item na loja (`after_win` em um `catalog_item`).
- Registro de uma venda (`after_win` em uma `action`).
- Conclusão de um desafio ou ganho de pontos (`after_win` em `challenge` ou `achievement`).

Quando o evento definido ocorre, o código da trigger é executado antes ou depois da operação ser concluída no banco de dados, dependendo do tipo de evento (`before` ou `after`).

Para que Servem as Triggers?

- **Automatizar regras de negócio:** como bônus especiais, ajustes de dados, bloqueios, validações e atualizações automáticas.
- **Economizar tempo de desenvolvimento externo:** boa parte da lógica pode ser feita direto no Funifier.
- **Manter a gamificação inteligente e responsiva:** reagindo a eventos sem precisar de sistemas terceiros.
- **Integrar com outros sistemas:** como enviar dados para APIs externas, Zapier, ERPs, CRMs ou enviar e-mails automáticos.

Funcionamento Básico

Uma trigger sempre segue este modelo de função em Java:

```
void trigger(event, entity, player, database) {  
    // Seu código aqui  
}
```

Onde:

- **event** → Informações sobre o evento que disparou a trigger.
- **entity** → O objeto afetado (jogador, action log, achievement, etc.).
- **player** → O ID do jogador relacionado ao evento.
- **database** → Permite acessar o banco de dados para consultar ou atualizar outras informações.

Exemplo Simples:

Imagine que você quer garantir que o nome dos jogadores sempre fique salvo em letras maiúsculas. Você cria uma trigger assim:

- **Evento:** `before_create`
- **Entidade:** `player`
- **Script:**

```
void trigger(event, entity, player, database) {  
    entity.name = entity.name.toUpperCase();  
}
```

Sempre que um novo jogador for criado, o nome será convertido para maiúsculas antes de ser salvo no banco.

Resumo deste Subtópico:

- Triggers permitem executar código automático quando algo acontece na gamificação.
- São muito úteis para automações internas e até integrações externas.

- Escrevemos essas triggers usando Java, diretamente dentro do Funifier Studio.

9.2. Entendendo os Eventos e Entidades das Triggers

Para criar uma trigger eficiente no Funifier, é essencial compreender dois conceitos fundamentais: os **Eventos** e as **Entidades**.

Esses dois elementos definem **quando** e **sobre o quê** sua trigger irá atuar.

O Que São Eventos?

Eventos definem **em que momento a trigger será executada** em relação à operação no banco de dados. Eles são compostos de um **prefixo** e um **sufixo**.

Prefixos disponíveis:

- **before_** → Executa **antes** da operação ser realizada no banco de dados.
- **after_** → Executa **após** a operação ser concluída no banco.

Sufixos (ações do banco ou conquistas):

- **create** → Quando um objeto é criado.
- **update** → Quando um objeto é alterado.
- **delete** → Quando um objeto é excluído.
- **win** → Quando um jogador **ganha** uma conquista, ponto, nível, item, ou conclui uma ação/desafio.
- **lose** → (Opcional) Quando um jogador **perde** uma conquista ou item (não muito comum).

Exemplos práticos de eventos:

Evento	Quando é disparado
before_create	Antes de um jogador, ponto ou item ser criado no banco.
after_create	Depois da criação ser concluída no banco.
before_update	Antes de atualizar uma entidade.
after_update	Depois de atualizar uma entidade.
after_win	Quando uma conquista ocorre: ganhar ponto, nível, item, etc.

O Que São Entidades?

A entidade define **qual objeto do Funifier** está sendo observado pela trigger. Ela informa qual tipo de dado ou operação será monitorada.

Principais Entidades Disponíveis:

Entidade	Descrição
player	Jogador
team	Equipe
action	Definição de uma ação (ex.: vender, assistir vídeo)
action_log	Registro de uma ação realizada por um jogador
challenge	Desafio
achievement	Conquista (pontos, nível, item, desafio concluído)
leaderboard	Ranking
catalog_item	Item da loja virtual
lottery	Sorteio
lottery_ticket	Cupom de sorteio
mystery_box	Caixa surpresa
question	Pergunta
quiz	Questionário
(custom)	Qualquer coleção customizada do banco (ex.: car__c, req__c)

Você pode escolher uma entidade padrão do Funifier ou até uma entidade personalizada da sua própria gamificação.

Exemplos de Combinação de Evento + Entidade:

Evento + Entidade	O que faz
before_create + player	Antes de criar um jogador no banco.
after_create + challenge	Após criar um desafio no sistema.

<code>after_win + catalog_item</code>	Após um jogador comprar um item na loja.
<code>after_win + challenge</code>	Após um jogador completar um desafio.
<code>after_win + action</code>	Após um jogador executar uma ação (ex.: venda registrada).
<code>before_win + achievement</code>	Antes de registrar um ganho de pontos, nível ou outro achievement.

Eventos Especiais "WIN" e "LOSE"

Quando você usa o evento `after_win` ou `before_win`, a trigger é executada quando o jogador:

- Ganha um ponto
- Conclui um desafio
- Sobe de nível
- Compra um item
- Ganha um sorteio
- Registra um action log

Nesses casos, o objeto manipulado dentro da trigger pode ser um **Achievement** (ganho de ponto, nível, item, desafio) ou um **ActionLog** (se for uma ação registrada, como uma venda, por exemplo).

Resumo Visual — Combinação de Evento + Entidade:

Formato: [prefixo]_[sufixo] + entidade

Exemplos:

- `before_create + player` → Antes de cadastrar jogador.
- `after_win + catalog_item` → Após comprar um item.
- `after_win + challenge` → Após concluir um desafio.
- `before_create + achievement` → Antes de criar um registro de pontos ou conquista.

Entender claramente a combinação de **evento** e **entidade** é o primeiro passo para dominar a criação de triggers na Funifier. Essa configuração define **quando** e **sobre o quê** sua automação será executada, permitindo construir regras de negócios poderosas e flexíveis.

9.3. Estrutura do Script de uma Trigger

Toda trigger na plataforma Funifier é composta por um código escrito em **Java**, que será executado no backend da gamificação sempre que o evento configurado ocorrer.

A estrutura do script é simples, porém poderosa, permitindo desde manipulações básicas até automações complexas, integrações externas e cálculos dinâmicos.

Estrutura Base da Trigger

O ponto de entrada obrigatório de todo script de trigger é o método chamado:

```
void trigger(event, entity, player, database) {  
    // Seu código aqui  
}
```

Parâmetros do Método

Parâmetro	Descrição
<code>event</code>	Representa o evento que disparou a trigger. Ex.: <code>after_win</code> , <code>before_create</code> .
<code>entity</code>	Objeto da entidade afetada. Por exemplo, se for um jogador, é o objeto Player.
<code>player</code>	ID do jogador relacionado ao evento, se aplicável.
<code>database</code>	Permite acessar diretamente o banco de dados MongoDB usando a biblioteca Jongo .

Estrutura do Objeto `entity`

O objeto `entity` representa os dados que estão sendo manipulados no evento. Ele muda de acordo com a entidade definida na configuração da trigger.

Entidade	Objeto Manipulado
<code>player</code>	Player
<code>action</code>	Action
<code>action_log</code>	ActionLog
<code>challenge</code>	Challenge
<code>achievement</code>	Achievement
<code>catalog_item</code>	Achievement (compra)
<code>lottery</code>	Achievement (sorteio)
<code>mystery_box</code>	Achievement (mystery)

Exemplo Simples de Trigger

Transformar o nome de um jogador para maiúsculas antes de criar:

```
void trigger(event, entity, player, database) {  
    entity.name = entity.name.toUpperCase();  
}
```

O Que Você Pode Fazer Dentro do Script

- Modificar dados do próprio objeto (**entity**).
- Ler informações de outros objetos no banco (**database**).
- Interagir com outros jogadores (ex.: dar pontos para amigos).
- Executar chamadas HTTP externas (ex.: notificar um webhook).
- Gerar códigos, tokens ou dados dinâmicos no momento do evento.
- Aplicar regras de negócio personalizadas.

Acesso ao Banco de Dados

Você pode acessar qualquer coleção do banco utilizando o parâmetro **database** com a biblioteca **Jongo**, nativamente integrada na plataforma.

Buscar um objeto:

```
Player p = database.getCollection("player")  
    .findOne("{_id: #}", "john")  
    .as(Player.class);
```

Atualizar:

```
p.name = "John Travolta Updated";  
database.getCollection("player").save(p);
```

Acesso aos Managers da Plataforma

Dentro de uma trigger, você também tem acesso ao objeto **manager**, que fornece métodos prontos para lidar com:

- Jogadores (**PlayerManager**)
- Ações e Logs (**ActionManager**)
- Conquistas (**AchievementManager**)
- Desafios (**ChallengeManager**)
- Itens e Lojas (**CatalogManager**)
- Sorteios (**LotteryManager**)

Exemplo:

```
Player p = manager.getPlayerManager().findById(player);
```

Exemplo Prático — Gerar Código de Convite Antes de Criar um Jogador

```
void trigger(event, entity, player, database) {
    String code = Guid.shortTimeMillis();
    entity.extra.put("invitation_code", code);
}
```

Neste exemplo, todo novo jogador criado recebe automaticamente um código de convite único salvo no campo `extra.invitation_code`.

Regras Importantes da Estrutura

- O método sempre se chama **`void trigger(event, entity, player, database)`**.
- Toda trigger deve ser escrita em Java (versão padrão suportada pela Funifier).
- O retorno é sempre **`void`** (não retorna valor, apenas executa ações e manipula dados).
- É possível fazer persistência de dados no banco via `database.save()` ou via managers (`manager.getPlayerManager().insert()`, por exemplo).

9.4. Manipulando Objetos na Trigger

Ao criar uma trigger, o primeiro nível de automação disponível é diretamente sobre o **objeto da entidade manipulada**, recebido no parâmetro `entity` do método `trigger`.

Quando a trigger é ativada, esse objeto já está carregado e pode ser alterado diretamente antes de ser salvo (em eventos `before_*`) ou após (em eventos `after_*`).

Assinatura do Script da Trigger

```
void trigger(event, entity, player, database) {
    // Seu código aqui
}
```

Exemplo 1 – Alterar o Nome do Jogador para Maiúsculo Antes de Cadastrar

Entidade	Evento
player	before_create

```
void trigger(event, entity, player, database) {
    entity.name = entity.name.toUpperCase();
}
```

Exemplo 2 – Adicionar um Código de Convite no Cadastro do Jogador

Entidade	Evento
----------	--------

player	before_create
--------	---------------

```
void trigger(event, entity, player, database) {
    String code = Guid.shortTimeMillis();
    entity.extra.put("invite_code", code);
}
```

Exemplo 3 – Dobrar Pontos Ganhados se o Jogador for do Departamento DITEC

Entidade	Evento
achievement	before_create

```
void trigger(event, entity, player, database) {
    if(entity.type == Achievement.TYPE_POINT) {
        Player p = manager.getPlayerManager().findById(player);
        if("DITEC".equals(p.extra.department)) {
            entity.total = entity.total * 2;
        }
    }
}
```

Esse tipo de manipulação é extremamente eficiente quando se deseja fazer ajustes simples no próprio objeto que está passando pelo fluxo de criação, atualização ou conquista.

Tabela de Objetos Recebidos nas Triggers e Suas Estruturas

Evento	Entidade	Exemplo de Objeto entity Recebido no Script
before_create	player	<pre>{ "_id": "player@funifier.com", "name": "Player Name", "email": "player@funifier.com", "image": { "small": {"url": "http://me.com/photo.png"}, "medium": {"url": "http://me.com/photo.png"}, "original": {"url": "http://me.com/photo.png"} }, "friends": ["ricardo", "sandra"], "teams": ["funifier", "octalysis"], "extra": { "gender": "male", "mobile": "+18895687914" } }</pre>

after_win	action	{ "_id": "5a3a6cb0d5327c7c0ba69f27", "actionId": "sell", "time": 1513778352559, "userId": "jerry.mouse@funifier.com", "attributes": { "product": "bicycle", "price": 350 } }
after_win	point_category (achievement)	{ "_id": "5a3a6cb0d5327c7c0ba69f27", "player": "player@funifier.com", "total": 100, "type": 0, "item": "xp", "time": 1513778352559 }
after_win	challenge (achievement)	{ "_id": "5a3a6cb0d5327c7c0ba69f27", "player": "player@funifier.com", "total": 1, "type": 1, "item": "challenge_id", "time": 1513778352559 }
after_win	level (achievement)	{ "_id": "5a3a6cb0d5327c7c0ba69f27", "player": "player@funifier.com", "total": 1, "type": 3, "item": "level_id", "time": 1513778352559 }
after_win	catalog_item (achievement)	{ "_id": "5a3a6cb0d5327c7c0ba69f27", "player": "player@funifier.com", "total": 1, "type": 2, "item": "item_id", "time": 1513778352559 }
after_win	lottery (achievement)	{ "_id": "5a3a6cb0d5327c7c0ba69f27", "player": "player@funifier.com", "total": 1, "type": 5, "item": "lottery_id", "time": 1513778352559 }

9.5. Manipulando Dados via Managers

Os **Managers** são objetos de serviço que oferecem métodos prontos para buscar, inserir, atualizar ou excluir dados de entidades padrão do Funifier, como jogadores, ações, conquistas, catálogos e sorteios.

Exemplo 1 – Dar Pontos Para Amigos Quando Alguém Faz 100 ou Mais Pontos

Entidade	Evento
achievement	before_create

```

void trigger(event, entity, player, database) {
    if(entity.type == Achievement.TYPE_POINT && entity.total >= 100) {
        Player p = manager.getPlayerManager().findById(player);
        for(String friend : p.friends) {
            Achievement a = new Achievement();
            a.id = Guid.newShortGuid();
            a.player = friend;
            a.type = 0; // Tipo ponto
            a.item = entity.item;
            a.total = 1;
            a.time = new Date();
            manager.getAchievementManager().addAchievement(a);
        }
    }
}

```

Exemplo 2 – Enviar E-mail de Boas-Vindas ao Jogador

Entidade	Evento
player	after_create

```

void trigger(event, entity, player, database) {
    long total = manager.getPlayerManager().findTotal();

    Email email = EmailBuilder
        .startingBlank()
        .from("Company", "your@company.com")
        .to(entity.getName(), entity.email)
        .withSubject("Welcome to Funifier!")
        .withPlainText("Welcome " + entity.name + ", you are the member number " + total)
        .buildEmail();

    MailerBuilder
        .withSMTPServer("smtp.server.com", 587, "login", "password")
        .buildMailer()
        .sendMail(email);
}

```


Exemplo 3 – Executar um Sorteio Quando Alguém Comprar um Ticket

Entidade	Evento
lottery_ticket	after_create

```
void trigger(event, entity, player, database) {  
    manager.getLotteryManager().execute(entity.lottery);  
}
```

Exemplo 4 – Registrar uma Ação Manualmente Dentro da Trigger

Entidade	Evento
achievement	after_create

```
void trigger(event, entity, player, database) {  
    ActionLog log = new ActionLog();  
    log.id = Guid.newShortGuid();  
    log.actionId = "bonus";  
    log.userId = player;  
    log.time = new Date();  
    log.attributes.put("reason", "Achievement reward");  
  
    manager.getActionManager().track(log);  
}
```

Tabela de Managers, Entidades e Operações Principais

Entidade	Manager	Operações Principais
Player	getPlayerManager()	findById(id)insert(player)delete(id) findTotal()
Achievement	getAchievementManager()	addAchievement(achievement)deleteAc hievment(id)undoAchievement(id)
Action	getActionManager()	findActionById(id)addAction(action) deleteAction(id)

ActionLog	<code>getActionManager()</code>	<code>track(actionLog)trackSynchronous(actionLog)</code>
Challenge	<code>getChallengeManager()</code>	<code>findById(id)add(challenge)delete(id)</code>
Lottery	<code>getLotteryManager()</code>	<code>find(id)insert(lottery)delete(id)execute(id)undoExecute(id)</code>
LotteryTicket	<code>getLotteryManager()</code>	<code>insertTicket(ticket)deleteTicket(ticket)</code>
Catalog	<code>getCatalogManager()</code>	<code>findById(id)add(catalog)delete(id)</code>
CatalogItem	<code>getCatalogManager()</code>	<code>findItemById(id)addItem(item)deleteItem(id)purchase(achievement, async)undoPurchase(id)</code>

9.6. Manipulando Dados via Database

O acesso via `database.getCollection()` permite trabalhar diretamente com qualquer coleção do banco, seja ela padrão do Funifier (como `player`, `achievement`, `action_log`), ou coleções customizadas (como `product__c`, `feedback__c`, etc.).

Esse tipo de acesso é extremamente poderoso para leituras, agregações ou manipulações diretas, especialmente em dados personalizados.

Assinatura do Script da Trigger

```
void trigger(event, entity, player, database) {
    // Acesso direto ao banco MongoDB
}
```

Exemplo 1 – Consultar um Produto Customizado no Banco

Entidade	Evento
<code>action</code>	<code>before_win</code>

```
void trigger(event, entity, player, database) {
    String id = entity.attributes.product;
    Object product = database.getCollection("product__c")
```

```

        .findOne("{_id: #}", id)
        .as(Object.class);

    entity.attributes.put("price", product.price);
}

```

Exemplo 2 – Criar um Registro Customizado no Banco

Entidade	Evento
player	after_create

```

void trigger(event, entity, player, database) {
    Map<String, Object> log = new HashMap<>();
    log.put("_id", Guid.newShortGuid());
    log.put("playerId", entity._id);
    log.put("event", "created");
    log.put("date", new Date());
    database.getCollection("player_log__c").save(log);
}

```

Exemplo 3 – Executar Aggregate Para Obter o Jogador com Mais XP

Entidade	Evento
player	after_create

```

void trigger(event, entity, player, database) {
    List<Object> top = Arrays.asList(
        database.getCollection("achievement")
            .aggregate("{ $match: {type: 0, item: 'xp'} }")
            .and("{ $group: { _id: '$player', total: { $sum: '$total' } } }")
            .and("{ $sort: { total: -1 } }")
            .and("{ $limit: 1 }")
            .as(Object.class)
    );
    if(!top.isEmpty()) {
        Object topPlayer = top.get(0);
        entity.extra.put("top_player", topPlayer._id);
    }
}

```

Exemplo 4 – Atualizar um Registro no Banco Dentro da Trigger

Entidade	Evento
player	after_create

```
void trigger(event, entity, player, database) {  
    database.getCollection("player_statistics__c")  
        .update("{_id: #}", entity._id)  
        .with("{$inc: {total_created: 1}}");  
}
```

Resumo Final dos Três Métodos de Manipulação em Triggers:

Método	Quando Usar
Manipulação Direta (entity)	Para alterar o próprio objeto afetado pela trigger (nome, extra, total).
Via Managers	Para criar, buscar ou atualizar objetos padrão do Funifier (players, logs, points, catalog).
Via Database	Para acessar coleções personalizadas, executar queries ou aggregates avançados.

Observação Importante

- No evento **after_win**, o objeto recebido na **entity** é quase sempre um **Achievement**, exceto quando a entidade configurada na trigger for **action**. Nesse caso, a **entity** será um **ActionLog**.
- A manipulação via **Managers** é altamente recomendada quando você está trabalhando com objetos padrões do Funifier, pois garante a aplicação de regras de negócio internas da plataforma.
- A manipulação via **database.getCollection()** deve ser utilizada para dados customizados ou queries mais complexas, inclusive aggregates.

9.7. Exemplos Práticos de Triggers no Dia a Dia

As triggers no Funifier são extremamente poderosas para implementar automações que respondem a comportamentos dos jogadores, entregam bônus, controlam regras de negócio e até integram sistemas externos. Veja abaixo alguns exemplos de como elas são usadas em cenários reais.

Exemplo 1 – Dobrar Pontos para Jogadores de um Departamento Específico

Cenário: Jogadores do departamento "TI" recebem o dobro de pontos XP.

Evento: `before_create`

Entidade: `achievement`

Script:

```
void trigger(event, entity, player, database) {
    if(entity.type == Achievement.TYPE_POINT && "xp".equals(entity.item)) {
        Player p = manager.getPlayerManager().findById(player);
        if("TI".equals(p.extra.department)) {
            entity.total = entity.total * 2;
        }
    }
}
```

Exemplo 2 – Gerar Código de Convite para Jogador no Cadastro

Evento: `before_create`

Entidade: `player`

Script:

```
void trigger(event, entity, player, database) {
    String code = Guid.shortTimeMillis();
    entity.extra.put("invite_code", code);
}
```

Exemplo 3 – Dar Pontos para Amigos Quando um Jogador Ganha 100 Pontos ou Mais

Evento: `before_create`

Entidade: `achievement`

Script:

```
void trigger(event, entity, player, database) {
    if(entity.type == Achievement.TYPE_POINT && entity.total >= 100) {
        Player p = manager.getPlayerManager().findById(player);
        for(String friend : p.friends) {
            Achievement a = new Achievement();
            a.player = friend;
            a.total = 10;
            a.type = Achievement.TYPE_POINT;
            a.item = entity.item;
            a.time = new Date();
            a.id = Guid.newShortGuid();
            manager.getAchievementManager().addAchievement(a);
        }
    }
}
```

```

    }
  }
}

```

Exemplo 4 – Enviar um Email de Boas-Vindas Quando um Jogador Se Cadastra

Evento: `after_create`

Entidade: `player`

Script:

```

void trigger(event, entity, player, database) {
    long totalPlayers = manager.getPlayerManager().findTotal();

    Email email = EmailBuilder
        .startingBlank()
        .from("Funifier", "noreply@funifier.com")
        .to(entity.getName(), entity.email)
        .withSubject("Bem-vindo ao Funifier!")
        .withPlainText("Olá " + entity.name + ", você é o jogador número " + totalPlayers + " da
plataforma!")
        .buildEmail();

    MailerBuilder
        .withSMTPServer("smtp.yourserver.com", 587, "login", "password")
        .buildMailer()
        .sendMail(email);
}

```

Exemplo 5 – Enviar Dados de uma Compra para um Webhook (Zapier, API Externa, etc.)

Evento: `after_win`

Entidade: `catalog_item`

Script:

```

void trigger(event, entity, player, database) {
    Player buyer = manager.getPlayerManager().findById(entity.player);
    CatalogItem item = manager.getCatalogManager().findItemById(entity.item);

    HashMap data = new HashMap();
    data.put("buyer", buyer);
    data.put("item", item);
    data.put("purchase", entity);

    HttpResponse<String> response =
Unirest.post("https://hooks.zapier.com/hooks/catch/xxxxxx")

```

```

        .header("Content-Type", "application/json")
        .body(JsonUtil.toJson(data))
        .asString();
    }

```

Exemplo 6 – Preencher Preço do Produto Automaticamente na Venda

Evento: `before_win`

Entidade: `action`

Script:

```

void trigger(event, entity, player, database) {
    String productId = entity.attributes.product;
    Object product = database.getCollection("product__c")
        .findOne("{_id:?}", productId)
        .as(Object.class);

    entity.attributes.put("price", product.price);
}

```

Esses são apenas alguns exemplos. Triggers podem ser usadas para gerar rankings, controlar regras de elegibilidade, criar experiências mais personalizadas e muito mais.

9.8. Cuidados, Limitações e Boas Práticas no Uso de Triggers

Triggers são extremamente poderosas, mas precisam ser usadas com responsabilidade. Veja os principais cuidados, limitações e boas práticas recomendadas para desenvolvimento com triggers no Funifier.

Performance e Eficiência

- **Cuidado com loops:** Evite loops desnecessários sobre grandes coleções. Cada chamada a `find()` ou `aggregate()` consome recursos do banco.
- **Evite triggers que disparam outras triggers em cascata.** Isso pode gerar sobrecarga e comportamento imprevisível.
- Utilize filtros bem definidos nas queries para não sobrecarregar o banco.

Boas Práticas de Código

- Sempre valide os dados recebidos no `entity` antes de fazer qualquer operação.
- Use condicionais para evitar que uma trigger rode desnecessariamente (Ex.: verifique o tipo da conquista, departamento do jogador, etc.).
- Documente dentro do próprio script o que sua trigger faz.

- Prefira o uso de **Managers** para manipular objetos padrões (players, achievements, actions) — eles garantem integridade de dados.

Uso Correto do Database

- Utilize `database.getCollection()` para acessar dados customizados.
- Prefira usar `aggregate()` quando precisar processar grandes volumes de dados de forma otimizada no próprio MongoDB.

Limitações Técnicas

- Triggers rodam em ambiente controlado. Elas não podem executar comandos externos ao sandbox da plataforma, exceto requisições HTTP.
- Não é possível manipular diretamente arquivos, sistema de arquivos ou executar código externo.
- Requisições HTTP devem ser rápidas. Se a API externa for lenta, isso impactará no desempenho da sua gamificação.

Debug e Testes

- Antes de ativar uma trigger em produção, teste em uma gamificação de desenvolvimento.
- Utilize logs na plataforma para acompanhar erros e execuções.
- Faça testes manuais executando ações que disparam a trigger e valide os efeitos no banco (via Database Explorer ou API).

Triggers são um dos maiores diferenciais da plataforma Funifier. Elas permitem levar sua gamificação a um nível de automação e inteligência muito além de regras pré-definidas. Quando bem utilizadas, podem transformar completamente a experiência dos jogadores e a eficiência dos processos.

10. Princípios Básicos de Segurança

A segurança dos dados na Funifier é uma prioridade fundamental tanto para a plataforma quanto para você, administrador ou desenvolvedor de gamificações. A proteção das informações dos jogadores, dos dados operacionais e dos sistemas integrados é essencial para garantir a privacidade, a confiabilidade e a integridade das suas soluções.

A Funifier oferece um conjunto robusto de recursos de segurança que incluem autenticação, controle de acesso granular, criptografia de dados sensíveis e práticas recomendadas para desenvolvimento seguro. No entanto, a segurança é uma responsabilidade compartilhada: enquanto a plataforma oferece as ferramentas, cabe a você configurá-las corretamente e seguir as melhores práticas.

Neste tópico, você aprenderá:

- Como funciona o modelo de autenticação e controle de acesso da Funifier.
- Como proteger seus dados usando roles, scopes e session timeout.
- Como funciona a autenticação de jogadores e aplicativos.
- Como implementar criptografia de dados sensíveis.
- Como realizar operações seguras no ciclo de vida dos jogadores, incluindo troca de senha, autenticação com email e proteção de dados pessoais (PII).
- Quais são as boas práticas, limitações e recomendações para garantir máxima segurança na sua gamificação.

10.1. Modelos de Autenticação no Funifier

A autenticação é o primeiro pilar da segurança no Funifier. Ela garante que somente usuários, jogadores ou aplicativos autorizados possam acessar sua gamificação e os dados nela contidos.

O Funifier oferece diferentes modelos de autenticação para atender às necessidades tanto de aplicações client-side (front-end) quanto server-side (back-end), proporcionando flexibilidade e segurança.

Tipos de Autenticação Disponíveis

Autenticação de Jogadores (Player Authentication)

- **Uso:** Ideal para quando jogadores precisam acessar suas próprias informações na gamificação, registrar ações e interagir com sistemas de forma segura.
- **Método:** Via login e senha (se configurado na área de segurança) ou métodos OAuth personalizados.
- **Token:** Gera um token do tipo **Bearer**, nomeado (associado ao jogador).
- **Validade:** Possui tempo de expiração (configurável, padrão 7 dias).
- **Permissões:** Limitadas por scopes definidos na role **player** ou roles personalizadas.

Exemplo de geração de token via senha:

POST <https://service2.funifier.com/v3/auth/token>

Content-Type: application/x-www-form-urlencoded

[apiKey=API_KEY&username=player_login&password=player_password&grant_type=password](#)

Autenticação de Aplicativos (App Authentication)

- **Uso:** Ideal para integrações entre servidores, automações ou sistemas administrativos.
- **Métodos disponíveis:**
 - **Basic Authentication:** Usa ApiKey + Secret. Token **nunca expira**.

- **Client Credentials (OAuth2):** Usa ApiKey + Secret. Token do tipo **Bearer** com validade configurável (padrão 7 dias).

Exemplo de autenticação via Client Credentials:

POST <https://service2.funifier.com/v3/auth/token>

Content-Type: application/x-www-form-urlencoded

`client_id=API_KEY&client_secret=APP_SECRET&grant_type=client_credentials`

Características dos Tokens

Tipo	Expira	Nominal (associa ao jogador)	Usa "me" nas rotas	Atualiza scope em tempo real
Player Password (Bearer)	✓	✓	✓	✗
Client Credentials	✓	✗	✗	✗
Basic Token	✗	✗	✗	✓

Quando Usar Cada Tipo

Cenário	Melhor Autenticação
Jogador logando no app (client-side)	Password (Bearer)
API server comunicando com Funifier	Client Credentials ou Basic
Dashboard administrativo personalizado	Basic
Integrações que não podem expirar (webhook)	Basic
Jogos, apps móveis com usuários autenticados	Password (Bearer)

Boas Práticas para Autenticação

- **Nunca compartilhe sua App Secret publicamente.**
- Prefira **Client Credentials** ou **Basic** para integrações backend.
- Sempre use HTTPS nas requisições à API.
- Evite expor tokens em front-end, especialmente o Basic Token que **não expira**.
- Use roles e scopes para controlar o que cada token pode acessar.

10.2. Controle de Acesso com Roles e Scopes

O controle de acesso no Funifier é baseado em dois pilares fundamentais:

1. **Roles (Papéis):** Grupos de permissões que podem ser atribuídos a jogadores ou aplicativos.
2. **Scopes (Escopos):** Definem exatamente quais operações podem ser realizadas em quais endpoints da API.

A combinação desses dois elementos permite que você configure desde acessos extremamente restritos até permissões administrativas completas, garantindo segurança, flexibilidade e aderência às necessidades específicas da sua gamificação.

Como Funciona

- Quando um **jogador** ou um **aplicativo** se autentica, ele recebe um token com as permissões definidas nas roles atribuídas.
- A API valida essas permissões a cada requisição.
- O controle é feito por:
 - **Operação:** `read`, `write`, `delete`
 - **Endpoint:** Ex.: `player`, `action_log`, `challenge`, ou `all` para todos.

Exemplo de Declarações de Scope

Declaração	Descrição
<code>read_all</code>	Permite ler todos os endpoints
<code>write_action_log</code>	Permite gravar logs de ação
<code>read_player_me</code>	Permite ler os próprios dados do jogador
<code>read_challenge</code>	Permite ler os desafios
<code>write_player</code>	Permite criar ou atualizar jogadores
<code>delete_player</code>	Permite excluir jogadores
<code>write_database_registration__c</code>	Permite criar registros na coleção customizada <code>registration__c</code>
<code>read_leaderboard</code>	Permite consultar rankings

Composição do Scope

Cada entrada de scope é composta por:

<operação>_<endpoint>

- **Operações suportadas:** `read`, `write`, `delete`
- **Wildcard:** O sufixo `_all` permite aplicar para todos os subníveis daquele endpoint.
Ex.:
 - `read_player_all` → Acesso a `/player`, `/player/me`, `/player/status`, etc.

Exemplo Real de Role

Role: `player` (default)

Campo	Valor
Name	player
Session Timeout	7d
Scope	read_all, write_action_log

Permite que jogadores leiam qualquer informação pública e registrem ações (action logs), mas **não podem criar nem excluir outros objetos**.

Exemplo Real de Role: `admin`

Campo	Valor
Name	admin
Session Timeout	30d
Scope	read_all, write_all, delete_all

Permissão total sobre todos os endpoints.

Como Atribuir Roles a Jogadores

Endpoint:

`POST /v3/role/assign`

Exemplo:

```
{  
  "player": "tom",  
  "role": "admin"  
}
```

Headers:

Authorization: Basic {APIKEY}:{APPSECRET}

Content-Type: application/json

Para remover:

DELETE /v3/role/assign

Hierarquia na Definição de Role

Ao autenticar, o Funifier segue essa lógica para determinar as permissões do token:

1. **Role com ID igual ao jogador.** (Se existir, tem prioridade total.)
2. Se não existir, então:
 - Verifica roles associadas ao jogador.
 - As permissões são a soma dos scopes de todas as roles.
 - O **session timeout** adotado será o menor entre todas as roles vinculadas.
3. Se não existir role vinculada:
 - Usa a role padrão chamada **player**.
4. Se não existir role **player**:
 - Usa uma role default mínima criada automaticamente pelo sistema.

Observação Importante

- **Tokens do tipo Basic:** Sempre validam o scope no momento da requisição, ou seja, se você alterar o scope da role ou app, o efeito é imediato.
- **Tokens do tipo Password ou Client Credentials:** Carregam o scope no momento da geração do token. Se você alterar o scope depois, o token antigo **não é afetado** e mantém as permissões até expirar.

Boas Práticas

- Sempre restrinja tokens do front-end para apenas **write_action_log** e **read_player_me**.
- Utilize roles diferentes para administradores e usuários comuns.
- Crie roles específicas para tarefas sensíveis, como gestão de sorteios (**lottery**) ou manipulação de dados (**database**).
- Acompanhe o uso dos tokens e rotacione as App Secrets periodicamente.
- Tokens do tipo Basic nunca expiram, use apenas para back-end e APIs seguras.

10.3. Sessão, Expiração de Tokens e Políticas de Timeout

O que é uma Sessão?

Na plataforma Funifier, uma sessão representa o período de tempo durante o qual um token de acesso é considerado válido. Durante essa sessão, o jogador ou aplicativo pode fazer requisições à API utilizando aquele token, sem necessidade de se autenticar novamente.

Tipos de Tokens e Suas Características

Tipo de Token	Expira?	É Nominal?	Scope é Validado em Tempo Real?	Recomendações
Password Token	✓	✓	✗	Frontend, apps públicos, mobile, web
Client Credentials	✓	✗	✗	Backend, automações, integrações
Basic Token	✗	✗	✓	Backend, APIs privadas, uso interno

Políticas de Expiração (Session Timeout)

Quando você gera um token (Password ou Client Credentials), ele terá um tempo de expiração configurado pelas **Roles** associadas ao jogador ou ao aplicativo.

Como é configurado?

- Acesse:
[Studio](#) → [Security](#) → [Roles](#)
- Campo:
[Session Timeout](#)
- Formato:
[<quantidade><unidade>](#)

Unidade	Descrição
y	Ano
M	Mês
w	Semana
d	Dia

h	Hora
m	Minuto
s	Segundo

Exemplos:

Valor	Interpretação
7d	7 dias
12h	12 horas
30m	30 minutos
90s	90 segundos (1,5 min)

Como Funciona na Prática?

- No momento da geração do token, o tempo de expiração é calculado com base no session timeout das roles.
- Se o jogador ou app tiver múltiplas roles, o **menor session timeout** será aplicado.
- Quando o token expira, todas as requisições passam a receber a resposta:

```
{
  "message": "token expired or format invalid",
  "code": 401,
  "type": "Unauthorized"
}
```

O Que Acontece Quando o Token Expira?

- O acesso à API é imediatamente bloqueado.
- A aplicação precisa:
 - Fazer o processo de autenticação novamente.
 - Ou, em ambientes client-side, exibir mensagem solicitando novo login.

Tokens do Tipo Basic: Atenção

- **Nunca expiram.**
- Sempre respeitam o scope atual da role ou app, mesmo que este seja alterado após a emissão do token.
- Por isso, use Basic Tokens **apenas em integrações backend seguras**, nunca em aplicações públicas.

Estratégias Recomendadas

- **Front-end:** Use Password Tokens. Crie session timeouts curtos (ex.: 1d ou 2h) e force o jogador a se autenticar novamente após expirar.
- **Back-end:** Use Client Credentials ou Basic Tokens, com roles bem definidas, escopos limitados e segredos rotacionados periodicamente.
- Implemente uma rotina que detecte respostas 401 e faça logout automático na interface do usuário.

Exemplo de Validação no Frontend (Logout Automático)

```
$http({
  method: "GET",
  url: Funifier.config.service + '/v3/player/me',
  headers: {"Authorization": Funifier.auth.getAuthorization()}
}).then(function(response){
  console.log(response.data);
}, function(err){
  if(err.status === 401){
    alert("Sua sessão expirou, por favor faça login novamente.");
    Funifier.auth.logout();
    window.location.href = "/login.html";
  }
});
```

10.4. Criptografia de Dados no Funifier

O que é Criptografia no Funifier?

A criptografia no Funifier é uma funcionalidade que permite proteger dados sensíveis armazenados na sua gamificação, garantindo que apenas usuários, aplicativos ou APIs autorizadas possam visualizar determinados campos.

Os campos criptografados são protegidos com chaves AES (Advanced Encryption Standard) de 128 bits, garantindo alto nível de segurança e aderência às principais legislações de proteção de dados, como **LGPD**, **GDPR**, entre outras.

Como Funciona?

- Dados sensíveis, como **email**, **telefone**, **CPF**, **informações bancárias**, **segredos**, podem ser configurados para serem criptografados diretamente no banco de dados.
- Quem não tiver permissão específica verá esses campos como textos cifrados (não legíveis).

- Usuários e APIs com a permissão correta podem visualizar e manipular esses dados de forma descriptografada.

Habilitando a Criptografia

1. Requisito:

- Sua conta precisa ter o módulo de criptografia ativado. Se não estiver, solicite à equipe da Funifier.



2. Configurando a Chave de Criptografia:

- Acesse: [Studio](#) → [Settings](#) → [Security Settings](#) → [Field Encryption](#) → [Manage Encryption](#)
- Clique em **"Key Management"** → **"Create"** para gerar sua chave de segurança.
- A chave deve ser forte, única e mantida em sigilo absoluto.

3. Escolhendo os Campos para Criptografar:

- Acesse a aba **"Field Encryption"**.
- Informe:
 - **Entity:** Nome da coleção (ex.: [player](#)).
 - **Fields:** Quais campos serão criptografados (ex.: [email](#), [extra.secret](#)).

Visualizando Dados Criptografados

 Com Permissão	 Sem Permissão
Dados aparecem descriptografados.	Dados aparecem como texto cifrado ilegível.

Permissão necessária:

[read_encrypted_field_values](#)

Regras e Restrições

- Campos criptografados:
 - Devem ser do tipo **String**.
 - Não podem ser **únicos**, **ID externo**, ou **objetos**.
 - Não são utilizáveis em **filtros**, **relatórios**, **sumários** ou **buscas**.

Rotação de Chaves

- Você pode gerar uma nova chave de criptografia periodicamente.
- Dados antigos continuam protegidos com a chave anterior até serem reprocessados.

- A Funifier oferece uma rotina de reprocessamento que descriptografa os dados antigos e os reescreve usando a nova chave, garantindo segurança contínua.

Exemplos Práticos

Definindo Criptografia no Campo **extra.secret** do Jogador:

Entity: player

Fields: extra.secret

Leitura de Dados Criptografados no Código Java de Trigger:

```
void trigger(event, entity, player, database) {  
    Player encrypted = manager.getPlayerManager().findById(player);  
    Player decrypted = (Player) manager.getCryptManager().decryptFields("player", encrypted);  
    println("Criptografado: " + encrypted.extra.get("secret"));  
    println("Descriptografado: " + decrypted.extra.get("secret"));  
}
```

Visibilidade de Senhas

- As senhas dos jogadores são **sempre criptografadas** e **nunca podem ser descriptografadas**, nem por usuários administradores.
- Porém, você pode controlar quem pode visualizar o campo **password** (sempre criptografado) através da permissão:

read_encrypted_player_password

Boas Práticas

- Criptografe **apenas o que for realmente sensível**. A criptografia gera sobrecarga de processamento e limita funcionalidades como filtros e buscas.
- Sempre mantenha sua chave de criptografia protegida.
- Implemente rotinas periódicas de rotação de chave, especialmente em ambientes com requisitos de compliance.

10.5. Proteção de Dados Pessoais (PII) e Conformidade com LGPD e GDPR

O que são PII, LGPD e GDPR?

- **PII (Personally Identifiable Information)**
São dados que permitem identificar uma pessoa específica, como: nome, CPF, e-mail, telefone, endereço, entre outros.
- **LGPD (Lei Geral de Proteção de Dados – Brasil)**
Legislação brasileira que regula o tratamento de dados pessoais, exigindo que empresas adotem medidas de segurança, transparência e consentimento dos titulares.
- **GDPR (General Data Protection Regulation – Europa)**
Legislação europeia que protege os dados dos cidadãos da União Europeia, com regras rígidas sobre coleta, armazenamento e uso de dados pessoais.

Como a Funifier ajuda na Proteção de Dados?

A Funifier foi construída com ferramentas e recursos que permitem implementar práticas seguras no tratamento de dados, ajudando sua empresa a estar em conformidade com legislações como LGPD e GDPR.

Recursos de Segurança Disponíveis:

- **Criptografia de Campos Sensíveis**
Permite armazenar informações pessoais de forma cifrada no banco de dados.
- **Controle de Acesso com Roles e Scopes**
Restringe quais dados e endpoints cada jogador, app ou usuário pode acessar.
- **Autenticação Segura e Tokens Controlados**
Tokens temporários, autenticação por senha, OAuth2 (Client Credentials) e Basic Auth.
- **Registro e Auditoria**
Todos os acessos e operações ficam registrados via logs, permitindo auditoria de acessos.
- **Anonimização e Pseudonimização (Via Triggers ou Processos)**
Possível implementar rotinas que anonimizam ou removem dados sensíveis de jogadores, se necessário.

Como Proteger Dados PII na Prática?

1. Criptografar Dados Sensíveis

- Email, telefone, CPF e outros dados podem ser criptografados utilizando o módulo de Field Encryption da Funifier.

2. Evitar Uso de Dados Pessoais como ID

- O campo `_id` do jogador, por padrão, é usado como login. Caso use email ou CPF, recomenda-se utilizar **login alternativo (alternative_logins)** e não no `_id`.

3. Criar Triggers para Gerenciar PII

- Triggers podem ser usadas para:

- Criptografar dados no momento do cadastro.
- Gerar IDs aleatórios.
- Sincronizar dados sensíveis de forma segura.

4. Controle Estrito de Permissões

- Defina roles específicas:
 - Ex.: `admin`, `data_protection_officer`.
- Permissão obrigatória para visualizar dados sensíveis: `read_encrypted_field_values`

5. Implemente Processo de Exclusão ou Anonimização

- Caso o usuário solicite a remoção dos dados (direito previsto na LGPD e GDPR), implemente:
 - Trigger para limpar ou anonimizar dados.
 - Remoção de registros sensíveis de coleções customizadas.

Exemplo Prático: Cadastro Protegido com Email Criptografado

- O email é criptografado e armazenado no campo `alternative_logins`.
- O `_id` do jogador é gerado de forma aleatória (não é mais o email).

Trigger Exemplo:

```
void trigger(event, entity, player, database){
    entity._id = Guid.shortTimeMillis();
    Object encrypt = manager.getCryptManager().encryptFields("player", entity);
    entity.email = encrypt.email;
    entity.addAlternativeLogin(encrypt.email);
}
```

Importante:

A palavra reservada `me` não funciona com login alternativo criptografado. Nesse caso, o sistema utiliza internamente o `_id` do jogador para identificar o usuário autenticado.

Benefícios de Estar em Conformidade

- Atende às exigências legais (LGPD, GDPR).
- Protege seus jogadores e seus dados sensíveis.
- Melhora a reputação da sua empresa no mercado.
- Evita riscos de multas e sanções.

11. Logística e Políticas do Exame

Objetivo da Certificação

A certificação **Funifier Gamification Developer I** valida a sua capacidade de implementar gamificações utilizando a API da Funifier, manipular dados, criar integrações, desenvolver interfaces com widgets e realizar automações utilizando triggers no Funifier Studio. Esta é uma certificação essencial para profissionais que desejam dominar o desenvolvimento técnico de soluções gamificadas dentro do ecossistema Funifier.

Público-Alvo

- Desenvolvedores, programadores, consultores técnicos e profissionais de tecnologia que trabalham na implementação de gamificações.
- Profissionais interessados em criar integrações, personalizar front-ends, desenvolver automações e construir soluções robustas com a plataforma Funifier.

Pré-Requisito

- Não há pré-requisitos formais para esta certificação.
- Conhecimentos básicos de lógica de programação e requisições API são recomendados para melhor desempenho no exame.

Principais Competências Avaliadas

- Manipulação da API REST Funifier (endpoints principais e database)
- Criação e customização de widgets (HTML, CSS, Javascript)
- Desenvolvimento de interfaces conectadas à API
- Implementação de triggers e automações via scripts Java no Funifier
- Manipulação de dados com aggregates
- Segurança, autenticação e controle de acesso na API Funifier

Detalhes do Exame

- **Tipo de Avaliação:** Prova objetiva (múltipla escolha)
- **Número de Questões:** 60 perguntas
- **Nível de Dificuldade:** Intermediário
- **Duração:** 90 minutos
- **Custo:** R\$ 800
- **Local:** Online ou em centros de teste credenciados
- **Pontuação para Aprovação:** 65% de acerto
- **Resultados:** Disponíveis imediatamente após a conclusão

Políticas do Programa de Certificação

- **Material:** Não é permitido o uso de materiais impressos, anotações ou acesso a recursos online durante o exame.
- **Termo de Concordância:** Ao participar, você concorda com o **Acordo do Programa de Certificação Funifier**, que estabelece diretrizes sobre:
 - **Confidencialidade:** Proteção do conteúdo da prova e não compartilhamento de perguntas e respostas.
 - **Conduta:** Proibição de qualquer tipo de cola, consulta ou comportamento desonesto.
 - **Integridade:** Compromisso com ética, honestidade e profissionalismo durante todo o processo de avaliação.

12. Manutenção da Certificação

Mantendo sua certificação Funifier Gamification Developer I ativa e válida:

Atualizações Anuais

- É obrigatório concluir módulos de atualização anuais, que incluem:
 - Novas funcionalidades da API e plataforma Funifier
 - Atualizações de segurança, autenticação e melhores práticas de desenvolvimento
 - Novas funcionalidades em widgets, aggregates, triggers e interface

Validade da Certificação

- Caso não seja realizada a atualização no prazo estipulado, a certificação passará ao status de **inativa/expirada**, e será necessário realizar um novo exame completo para reativação.

Conclusão

Este guia foi desenvolvido para fornecer todos os conteúdos necessários para a sua **preparação completa rumo à certificação Funifier Gamification Developer I**. Agora é com você! Revise, pratique, desenvolva protótipos, simule integrações e mergulhe profundamente no universo da Funifier. **Boa sorte na prova! Que você se torne oficialmente um(a) Funifier Developer certificado(a)!**

Anexo I - Módulos Funifier

A plataforma Funifier é composta por diversos módulos que permitem ao designer configurar, personalizar e gerenciar cada aspecto de uma estratégia de gamificação. Cada módulo atende a uma função específica, desde o cadastro de ações e participantes até a definição de regras, recompensas, campanhas e integrações externas. A seguir, apresentamos os principais módulos da plataforma, organizados por assunto, com suas respectivas funções e exemplos de uso para facilitar o entendimento e a implementação no Funifier Studio ou via API.

Módulos de Configuração Básica

A configuração básica é o ponto de partida para qualquer gamificação no Funifier. Aqui você define as principais entidades que serão utilizadas ao longo da estratégia: as ações que os jogadores podem realizar, quem são esses jogadores e como eles se agrupam em equipes.

Action (Ação)

- **Acesso:** `/studio/action`
 - **API:** `/v3/action`
 - **Função:** Cadastro das diferentes ações que os jogadores podem realizar na gamificação.
 - **Descrição:**
Permite criar e configurar todas as ações possíveis dentro da experiência gamificada, como “vender”, “comprar”, “comentar”, “compartilhar”, entre outras. As ações são a base da estratégia de engajamento, pois definem o que é possível fazer na gamificação e servem para disparar regras e recompensas. É possível adicionar atributos extras a cada ação, por exemplo, “produto” e “valor” na ação de venda. Isso possibilita desde interações simples até configurações avançadas, como metas personalizadas.
 - **Exemplo de uso:**
 - Ação de “logar” para iniciar sessão.
 - Ação de “vender” com atributos de produto e valor, permitindo criar regras como “vender 2 carros de R\$50.000 por semana durante 3 meses”.
-

Player (Jogador)

- **Acesso:** `/studio/player`
- **API:** `/v3/player`

- **Função:** Cadastro e gerenciamento dos participantes da gamificação.
 - **Descrição:**
Permite cadastrar e detalhar cada jogador, incluindo nome, login, email, equipes, amigos, foto, avatar e informações adicionais como telefone (para envio de SMS), integração com redes sociais (para contabilizar interações externas) e data de aniversário (para recompensas exclusivas). Também é possível definir senha para cada jogador no momento da configuração.
 - **Exemplo de uso:**
 - Cadastro do jogador João Silva, com login “jsilva”, email “joao.silva@example.com”, equipe “Vendas” e foto de perfil.
-

Team (Equipe)

- **Acesso:** `/studio/team`
 - **API:** `/v3/team`
 - **Função:** Configuração de grupos ou equipes de jogadores.
 - **Descrição:**
Permite criar grupos fixos (como “Equipe de Vendas” ou “Equipe de Tecnologia”) ou equipes dinâmicas, formadas automaticamente com base em critérios como aniversariantes do dia. O designer pode nomear equipes e adicionar imagens ou brasões personalizados, como acontece em times esportivos. Equipes permitem a competição ou colaboração entre grupos dentro da gamificação.
 - **Exemplo de uso:**
 - Equipe de Tecnologia, composta por desenvolvedores e analistas.
 - Equipe dinâmica de aniversariantes do dia, formada automaticamente pela data de nascimento dos jogadores.
-

Módulos de Mecânicas de Jogos

Os módulos de mecânicas de jogos são responsáveis por estruturar toda a lógica de progressão, desafios, recompensas, competição e interação entre jogadores dentro da gamificação. Eles possibilitam criar experiências ricas, dinâmicas e adaptadas a diferentes públicos e objetivos, permitindo ao designer implementar desde mecânicas clássicas, como

pontos e rankings, até experiências inovadoras, como sorteios, avatares personalizados e histórias interativas.

Point (Ponto)

- **Acesso:** `/studio/point`
 - **API:** `/v3/point`
 - **Função:** Configuração dos diferentes tipos de pontuação que os jogadores podem conquistar.
 - **Descrição:**
Permite criar e personalizar unidades de medida de progresso, como XP, moedas virtuais, pontos de conhecimento ou métricas customizadas. Os pontos são fundamentais para recompensar ações, mensurar desempenho e alimentar outros módulos, como rankings, lojas virtuais e desafios.
 - **Exemplo de uso:**
 - XP, moedas, karma, pontos em diferentes áreas de conhecimento ou desempenho.
-

Challenge (Desafio)

- **Acesso:** `/studio/challenge`
 - **API:** `/v3/challenge`
 - **Função:** Configuração de desafios a serem completados por jogadores ou equipes.
 - **Descrição:**
Permite criar desafios personalizados, definindo ações e recompensas envolvidas. Os desafios podem variar de simples tarefas a missões complexas compostas por múltiplas ações ou etapas, exigindo planejamento e engajamento dos participantes.
 - **Exemplo de uso:**
 - Assistir a um vídeo para ganhar pontos; vender produtos por uma semana para ganhar prêmios; completar uma sequência de ações para receber recompensas especiais.
-

Level (Nível)

- **Acesso:** `/studio/level`
 - **API:** `/v3/level`
 - **Função:** Sistema de progressão por níveis.
 - **Descrição:**

Permite definir a progressão dos jogadores com base em pontos acumulados ou outros critérios, atribuindo títulos ou benefícios a cada nível. Pode-se exigir também desafios específicos para desbloquear certos níveis, tornando a progressão mais desafiadora.
 - **Exemplo de uso:**
 - Títulos como Júnior, Pleno, Sênior em ambientes corporativos; faixas coloridas em artes marciais.
-

Leaderboard (Ranking)

- **Acesso:** `/studio/leaderboard`
 - **API:** `/v3/leaderboard`
 - **Função:** Criação e gestão de rankings.
 - **Descrição:**

Permite comparar o desempenho dos jogadores utilizando diferentes critérios, como pontos, tempo, ou métricas customizadas. Rankings podem ser individuais, por equipes, por período ou por status.
 - **Exemplo de uso:**
 - Ranking de vendedores por volume de vendas; funcionários com mais pontos; alunos com melhores notas.
-

Virtual Good (Loja Virtual)

- **Acesso:** `/studio/catalog`
- **API:** `/v3/catalog`
- **Função:** Cadastro e gestão de objetos e benefícios que podem ser adquiridos pelos jogadores.

- **Descrição:**
Permite cadastrar produtos físicos, virtuais ou benefícios, estipular preços, quantidades, limites de compra e atributos. Os jogadores podem trocar pontos ou itens por esses objetos, que também podem ser concedidos como recompensas de desafios.
 - **Exemplo de uso:**
 - Camiseta de evento por 100 pontos; resgate de drone com peças colecionáveis; gift cards; itens virtuais para avatares.
-

Lottery (Sorteio)

- **Acesso:** `/studio/lottery`
 - **API:** `/v3/lottery`
 - **Função:** Configuração de sorteios e concursos.
 - **Descrição:**
Permite criar sorteios nos quais os jogadores participam com cupons obtidos em outras mecânicas, como desafios. É possível definir datas, limites de ganhadores e prêmios para os sorteados.
 - **Exemplo de uso:**
 - Sorteio de viagem; campanhas promocionais com distribuição de cupons.
-

Competition (Competição)

- **Acesso:** `/studio/competition`
- **API:** `/v3/competition`
- **Função:** Configuração de disputas diretas entre jogadores ou equipes.
- **Descrição:**
Permite criar competições com regras próprias, prêmios para diferentes posições e critérios de apuração customizáveis. Geralmente, o jogador ou equipe precisa se inscrever para participar.
- **Exemplo de uso:**

- Competição de vendas semanais com premiação para os melhores colocados.

Swap (Troca)

- **Acesso:** `/studio/swap`
- **API:** `/v3/swap`
- **Função:** Negociação e troca de objetos ou pontos entre jogadores.
- **Descrição:**
Permite que jogadores façam ofertas públicas de troca, negociando objetos ou pontos. Outros jogadores podem aceitar ou propor novas condições, incentivando a colaboração e interação.
- **Exemplo de uso:**
 - Troca de figurinhas repetidas, objetos de loja ou moedas.

Question (Pergunta)

- **Acesso:** `/studio/question`
- **API:** `/v3/question`
- **Função:** Configuração de perguntas e respostas para os jogadores.
- **Descrição:**
Permite criar perguntas em diferentes formatos (múltipla escolha, verdadeiro/falso, com mídia), definir respostas corretas ou colher opiniões, e aplicar pesos diferentes para as perguntas.
- **Exemplo de uso:**
 - Quiz de conhecimentos gerais; perguntas de opinião para pesquisas internas.

Quiz (Quiz)

- **Acesso:** `/studio/quiz`
- **API:** `/v3/quiz`

- **Função:** Agrupamento de perguntas em formato de prova ou simulado.
 - **Descrição:**
Permite criar provas compostas por perguntas, definir nota total, controlar tentativas e embaralhar questões para cada jogador. Ideal para avaliações, simulados ou concursos internos.
 - **Exemplo de uso:**
 - Prova de matemática; simulado de história com embaralhamento de perguntas.
-

Mystery (Mistério)

- **Acesso:** </studio/mystery>
 - **API:** </v3/mystery>
 - **Função:** Jogos de probabilidade e recompensas aleatórias.
 - **Descrição:**
Permite criar experiências como caixas surpresa, rodas da fortuna, raspadinhas e outros jogos baseados em sorte, definindo as chances e prêmios disponíveis.
 - **Exemplo de uso:**
 - Roda da Fortuna com prêmios variados; raspadinhas premiadas.
-

Crossword (Palavras Cruzadas)

- **Acesso:** </studio/crossword>
- **API:** </v3/crossword>
- **Função:** Configuração de jogos de palavras cruzadas.
- **Descrição:**
Permite criar jogos de palavras cruzadas customizados, com dicas, posições e temas específicos para diferentes objetivos educacionais ou promocionais.
- **Exemplo de uso:**

- Palavras cruzadas sobre termos técnicos para onboarding de funcionários; desafios temáticos para campanhas de marketing.

Story (História)

- **Acesso:** `/studio/story`
- **API:** `/v3/story`
- **Função:** Criação de histórias interativas com múltiplos caminhos e finais.
- **Descrição:**
Permite criar narrativas gamificadas nas quais o jogador faz escolhas que afetam o desenrolar da trama. Cenários, personagens, imagens e vídeos podem ser utilizados para enriquecer a experiência.
- **Exemplo de uso:**
 - Treinamentos baseados em storytelling; campanhas de marketing com desfechos alternativos; jogos educacionais interativos.

Avatar (Avatar)

- **Acesso:** (Módulo não tem caminho específico no Studio, pode ser acessado em diversos pontos)
- **Função:** Criação e customização de avatares para os jogadores.
- **Descrição:**
Permite aos jogadores criarem avatares personalizados, desbloqueando acessórios e itens de acordo com o desempenho na gamificação ou por compras na loja. Pode ser usado em ambientes 2D ou 3D.
- **Exemplo de uso:**
 - Avatares com roupas e acessórios de uma marca; avatares para perfis em ambientes virtuais.

LastMile (Última Milha)

- **Acesso:** `/studio/lastmile`
 - **API:** `/v3/lastmile`
 - **Função:** Envio de mensagens motivacionais quando o jogador está próximo de uma meta.
 - **Descrição:**
Permite configurar mensagens automáticas para engajar e lembrar os jogadores sobre metas e desafios que estão prestes a serem concluídos, aumentando a taxa de finalização.
 - **Exemplo de uso:**
 - Lembrete para quem completou 90% de um desafio; mensagem motivacional para vendedores próximos da meta.
-

Folder (Pasta)

- **Acesso:** `/studio/folder`
 - **API:** `/v3/folder`
 - **Função:** Organização de conteúdos e monitoramento do progresso em cursos ou trilhas.
 - **Descrição:**
Permite criar pastas virtuais para agrupar conteúdos como quizzes, desafios, materiais de estudo e jogos, facilitando o acompanhamento do progresso dos jogadores em jornadas de aprendizagem ou trilhas gamificadas.
 - **Exemplo de uso:**
 - Estruturação de cursos online, trilhas de capacitação, acompanhamento de progresso em treinamentos.
-

Módulos de Registro de Atividades e Conquistas

Esses módulos são responsáveis por monitorar, registrar e comunicar cada passo dos jogadores dentro da gamificação. Eles viabilizam o acompanhamento do progresso, o reconhecimento de conquistas e a geração de feedbacks automáticos, além de integrarem as mecânicas do Funifier às principais técnicas de jogos e motivações dos usuários. Utilizando

esses módulos, o designer garante rastreabilidade, transparência e engajamento contínuo para todos os participantes.

Action Log (Registro de Ação)

- **API:** `/v3/action/log`
 - **Função:** Registro detalhado de todas as ações executadas pelos jogadores.
 - **Descrição:**
Armazena cada ação realizada, com detalhes como quem executou, qual ação foi feita e quando ocorreu. Esse registro é fundamental para alimentar todos os outros módulos da plataforma, permitindo calcular impactos, atualizar conquistas, rankings e enviar notificações relevantes.
 - **Exemplo de uso:**
 - João realizou uma venda de um livro de 120 reais às 11:30 do dia 01/09/2024.
-

Progress Log (Registro de Progresso)

- **Função:** Monitoramento do progresso dos jogadores em relação a objetivos, desafios ou jornadas.
 - **Descrição:**
Mantém o acompanhamento em tempo real do quanto cada jogador já avançou em suas metas, desafios, quizzes ou cursos. O registro é atualizado toda vez que uma nova ação é registrada, permitindo feedbacks constantes e transparentes ao participante.
 - **Exemplo de uso:**
 - Jogador completou 50% de um desafio “Faça 10 vendas”.
 - Jogador avançou 85% no curso de programação Java.
-

Achievement (Conquista)

- **API:** `/v3/achievement`

- **Função:** Registro das conquistas dos jogadores ao atingirem marcos relevantes.
 - **Descrição:**
Todas as conquistas – como completar desafios, subir de nível, ganhar pontos ou adquirir itens – são registradas nesse módulo. Ele automatiza o reconhecimento dos marcos alcançados, mas também permite ajustes manuais sempre que necessário.
 - **Exemplo de uso:**
 - Sandra completou o desafio “Faça 10 vendas”.
 - Beatriz subiu para o nível Princesa Valorosa.
-

Notification (Notificação)

- **API:** [/v3/notification](#)
 - **Função:** Envio de mensagens automáticas e feedback instantâneo aos jogadores.
 - **Descrição:**
Permite configurar e disparar notificações personalizadas quando eventos importantes acontecem, como a conclusão de um desafio, o ganho de uma competição ou uma compra na loja virtual. Notificações podem ser configuradas diretamente em diferentes módulos.
 - **Exemplo de uso:**
 - “Parabéns Sandra! Você acaba de completar o desafio Faça 10 vendas.”
 - “Parabéns, Ricardo! Você acabou de ganhar a competição de vendas em 1º lugar e ganhou 1 diamante.”
-

Technique Link (Link de Técnica)

- **Função:** Conexão entre módulos da plataforma e técnicas de jogo.
- **Descrição:**
Atua como ponte entre cada configuração de módulo e as técnicas de jogos associadas, facilitando o rastreo das motivações principais dos jogadores e oferecendo insights sobre o engajamento. Isso possibilita análises e ajustes mais estratégicos nas mecânicas do sistema.

- **Exemplo de uso:**
 - Vincular o item “moeda” do módulo Pontos à técnica de jogo “Pontos Trocáveis”.
-

Módulos de Configurações Avançadas

Os módulos de configurações avançadas da plataforma Funifier oferecem flexibilidade e poder para personalizar o comportamento do sistema, automatizar tarefas e implementar lógicas complexas que vão além das funcionalidades padrão. Esses recursos são especialmente úteis para desenvolvedores e administradores que precisam adaptar a gamificação a regras de negócio, integrações e dinâmicas sofisticadas.

Trigger (Trigger)

- **Acesso:** `/studio/trigger`
 - **API:** `/v3/trigger`
 - **Função:** Execução automática de códigos JAVA quando eventos específicos acontecem na gamificação.
 - **Descrição:**

Permite programar ações customizadas disparadas por eventos, como o cadastro de um jogador ou a realização de uma compra. As triggers dão liberdade para alterar o comportamento padrão dos módulos, integrar com sistemas externos ou implementar automações em tempo real.
 - **Exemplo de uso:**
 - Enviar um email de boas-vindas quando um novo jogador é cadastrado.
 - Fazer requisição a uma API externa ao completar uma compra.
 - Alterar a quantidade de pontos concedidos ao finalizar um desafio.
-

Scheduler (Scheduler)

- **Acesso:** `/studio/scheduler`
- **API:** `/v3/scheduler`

- **Função:** Execução de códigos JAVA em datas e horários agendados (expressões CRON).
 - **Descrição:**
Permite agendar tarefas e automações para rodar em horários ou intervalos pré-definidos. Ideal para atividades recorrentes, relatórios, campanhas temporais ou aplicação de regras periódicas.
 - **Exemplo de uso:**
 - Gerar relatórios de desempenho toda sexta-feira às 10h.
 - Debitar pontos de jogadores inativos no fim do mês.
 - Enviar lembretes automáticos de metas pendentes.
-

KPI Formulas (Fórmulas de KPI)

- **Função:** Configuração de regras e cálculos avançados para avaliar ações dos jogadores.
 - **Descrição:**
Permite criar fórmulas customizadas para avaliação de desempenho e resultados, indo além das regras padrão dos outros módulos. Essas fórmulas podem ser utilizadas em desafios, rankings e outros pontos estratégicos do sistema. Devem ser usadas com critério, apenas quando os recursos padrão não atendem plenamente à necessidade.
 - **Exemplo de uso:**
 - Fórmula para calcular eficiência de vendas em desafios considerando múltiplos critérios (quantidade, valor, tempo).
 - Regras avançadas para rankings, diferenciando jogadores com base em uma combinação de indicadores.
-

Módulos de Interface Visual e Feedback

Os módulos de interface visual e feedback são responsáveis por transformar as informações e mecânicas da gamificação em experiências visuais acessíveis e motivadoras para os jogadores. Eles também viabilizam interações dinâmicas e feedbacks instantâneos, seja em portais, sistemas corporativos ou áreas administrativas. Com esses recursos, a Funifier garante

que tanto jogadores quanto administradores possam visualizar, interagir e acompanhar o progresso gamificado em tempo real.

Widget (Widget)

- **Acesso:** `/studio/widget`
 - **API:** `/v3/widget`
 - **Função:** Criação de componentes visuais para exibir técnicas de jogo.
 - **Descrição:**

Permite criar e configurar widgets gráficos — como rankings, listas de desafios, status do jogador, lojas virtuais e muito mais — para apresentar informações gamificadas de maneira clara e motivadora. Esses widgets podem ser integrados em sistemas web existentes, como intranets, CRMs ou LMS, tornando a experiência de gamificação visível e acessível ao público-alvo.
 - **Exemplo de uso:**
 - Exibir o ranking semanal de vendedores na intranet.
 - Mostrar uma lista de desafios ativos para toda a equipe.
-

WebSocket (WebSocket)

- **Acesso:** `/studio/websocket`
 - **API:** `/v3/websocket`
 - **Função:** Entrega de feedbacks em tempo real e interação instantânea entre jogadores.
 - **Descrição:**

Permite configurar rotinas para enviar e receber informações instantaneamente entre jogadores e o sistema. É ideal para chats ao vivo, notificações instantâneas, atualização de status ou acompanhamento de desafios sem precisar recarregar a página.
 - **Exemplo de uso:**
 - Comunicação em tempo real entre jogadores durante competições.
 - Feedback instantâneo ao completar um desafio ou conquistar uma recompensa.
-

Static Repo (Repositório Estático)

- **Acesso:** `/studio/static`
 - **API:** `/v3/static`
 - **Função:** Hospedagem de interfaces e conteúdos estáticos em subdomínios Funifier.
 - **Descrição:**

Permite criar repositórios públicos nos servidores da Funifier para hospedar interfaces gráficas customizadas. Os conteúdos podem ser enviados manualmente, sincronizados via Git ou upload de arquivos, facilitando o acesso dos jogadores a uma experiência gamificada personalizada.
 - **Exemplo de uso:**
 - Hospedagem de uma interface exclusiva da gamificação em um subdomínio próprio (ex: seugame.funifier.app).
-

Studio Page (Página do Studio)

- **Acesso:** `/studio/page`
 - **Função:** Criação de páginas personalizadas para administração no Funifier Studio.
 - **Descrição:**

Permite criar páginas customizadas na área administrativa do Funifier Studio, utilizando HTML, CSS e JavaScript (AngularJS e Bootstrap). É ideal para gestores que desejam visualizar relatórios, gráficos, indicadores de desempenho ou criar funcionalidades administrativas sob medida para suas necessidades.
 - **Exemplo de uso:**
 - Página personalizada com gráficos de vendas por vendedor.
 - Painel de acompanhamento de desempenho de equipes ou campanhas.
-

Módulos de Integração e Conectividade

A Funifier oferece uma arquitetura aberta e flexível para integração com sistemas externos, automação de processos e desenvolvimento de experiências personalizadas. Esses módulos facilitam tanto a comunicação em tempo real quanto a preparação e encapsulamento de

comandos, permitindo que empresas expandam e conectem a gamificação a qualquer contexto de negócio.

API Restful (API Restful)

- **Função:** Interface principal para integração de sistemas externos com a gamificação.
 - **Descrição:**
Disponibiliza uma interface robusta para construir integrações, automatizar processos e criar novas interfaces para a gamificação. Todas as operações do motor de gamificação da Funifier podem ser acionadas por API, garantindo máxima flexibilidade para integração com CRMs, ERPs, portais externos e outras soluções.
 - **Exemplo de uso:**
 - Sincronizar dados de clientes e atividades entre o CRM e a gamificação.
 - Desenvolvimento de portais e apps que consomem diretamente os serviços do Funifier.
-

Request (Request)

- **Acesso:** `/studio/request`
 - **Função:** Ferramenta para testar e preparar requisições à API.
 - **Descrição:**
Permite aos desenvolvedores testar, validar e compartilhar requisições de API, garantindo que integrações e automações estejam funcionando conforme o esperado. Facilita o desenvolvimento colaborativo e a documentação dos fluxos de integração.
 - **Exemplo de uso:**
 - Testar requisições para integração com sistemas como CRM, ERPs, e-commerce etc.
 - Compartilhar coleções de requisições com outros membros da equipe.
-

Find (Find)

- **Acesso:** `/studio/prepared`
 - **API:** `/v3/find`
 - **Função:** Cadastro de comandos de consulta encapsulados.
 - **Descrição:**

Permite criar e salvar comandos aggregate complexos no servidor, protegendo a lógica de banco de dados dos usuários finais. Dessa forma, consultas sofisticadas podem ser chamadas de forma simplificada e segura pelas interfaces da gamificação.
 - **Exemplo de uso:**
 - Criar uma consulta agregada que retorna o histórico completo de ações de um jogador, sem expor detalhes do banco de dados ao frontend.
-

SDK (SDK)

- **Função:** Ferramentas para desenvolvedores criarem software e aplicações personalizadas.
 - **Descrição:**

O SDK inclui tudo o que desenvolvedores precisam para criar, estender ou integrar soluções com a Funifier: bibliotecas, depuradores, documentação, tutoriais e exemplos práticos, tornando o processo de desenvolvimento mais ágil e seguro.
 - **Exemplo de uso:**
 - Desenvolvimento de novas funcionalidades, widgets ou integrações utilizando o SDK da Funifier.
-

WebHook (WebHook)

- **API:** `/v3/webhook`
- **Função:** Notificações automáticas para endpoints externos em tempo real.
- **Descrição:**

Permite que a Funifier envie notificações automáticas (payloads) para sistemas externos sempre que eventos importantes ocorrem na gamificação, sem necessidade de polling. Os Webhooks garantem atualizações em tempo real, otimizando recursos e

melhorando a experiência do usuário.

- **Exemplo de uso:**
 - Enviar dados para um sistema de BI ou CRM sempre que um jogador completar um desafio ou atingir uma meta.
-

Módulos de Segurança e Controle de Acesso

Garantir a segurança e o controle de acesso é fundamental para qualquer estratégia de gamificação, especialmente em ambientes corporativos ou que lidam com dados sensíveis. Os módulos abaixo oferecem as ferramentas necessárias para proteger informações, personalizar autenticações, auditar ações e garantir a integridade de todas as operações na plataforma Funifier.

Auth (Autenticação)

- **API:** `/v3/auth`
 - **Função:** Gerenciamento de autenticação e critérios básicos de segurança.
 - **Descrição:**

Permite configurar como será o processo de login dos jogadores, definindo exigência de senha, auto-criação de usuários, geração de tokens de acesso (incluindo tempo de expiração e revalidação), entre outros critérios essenciais para proteção de acessos.
 - **Exemplo de uso:**
 - Exigir senha dos jogadores ao fazer login.
 - Geração de tokens de acesso com expiração definida para sessões seguras.
-

Auth Module (Módulo Customizado de Autenticação)

- **Acesso:** `/studio/auth`
- **API:** `/v3/auth/module`
- **Função:** Criação de rotinas personalizadas de autenticação.

- **Descrição:**
Permite aos desenvolvedores criar processos de autenticação sob medida, adicionando etapas de validação extra, integração com sistemas externos, registro de logs de login, ou a implementação de SSO (Single Sign-On) para unificação de acessos entre plataformas.
 - **Exemplo de uso:**
 - Validação do login no sistema da empresa antes de liberar acesso ao Funifier.
 - Registro de um log sempre que um jogador faz login.
 - Implementação de SSO entre Funifier e outros sistemas corporativos.
-

Audit (Auditoria)

- **Acesso:** `/studio/audit`
 - **API:** `/v3/audit`
 - **Função:** Auditoria e rastreamento de alterações e eventos no sistema.
 - **Descrição:**
Permite definir quais informações e operações devem ser auditadas, garantindo rastreabilidade e transparência. Registra quem alterou um dado, quando e qual foi a modificação, protegendo o sistema contra fraudes, erros e ações não autorizadas.
 - **Exemplo de uso:**
 - Rastreamento de alterações no cadastro de jogadores.
 - Auditoria de exclusão de dados ou mudanças em níveis/pontuações.
-

Crypt (Criptografia)

- **Acesso:** `/studio/crypt`
- **API:** `/v3/crypt`
- **Função:** Criptografia de dados sensíveis.
- **Descrição:**
Protege informações críticas e confidenciais (dados pessoais, senhas, registros estratégicos), garantindo que apenas usuários autorizados possam acessá-las. Esse módulo impede a visualização ou leitura indevida por terceiros não autorizados.

- **Exemplo de uso:**
 - Criptografia de informações pessoais dos jogadores para garantir a privacidade e segurança dos dados.
-

Módulos de Gestão e Otimização de Dados

Para garantir uma operação eficiente, escalável e flexível, a plataforma Funifier oferece recursos avançados de gestão, importação, exportação, organização, backup e otimização dos dados da gamificação. Esses módulos permitem que administradores e desenvolvedores extraiam o máximo valor dos dados, realizem operações complexas, personalizem a experiência e mantenham a performance do sistema.

Database (Database)

- **API:** [/v3/database](#)
 - **Função:** Acesso direto à base de dados para operações avançadas.
 - **Descrição:**

Oferece ferramentas para criar índices, executar comandos aggregate (MongoDB), importar dados em lote, e muito mais. Permite extrair relatórios personalizados e realizar manipulações complexas diretamente no banco de dados da gamificação.
 - **Exemplo de uso:**
 - Criação de índices para otimizar consultas.
 - Geração de relatórios por comandos aggregate.
 - Importação em massa de dados históricos.
-

Custom Object (Objeto Customizado)

- **Função:** Criação de objetos personalizados para necessidades específicas.
- **Descrição:**

Permite criar e gerenciar coleções personalizadas no banco de dados (sufixo `__c`), que podem ser acessadas via API, triggers, schedulers e páginas personalizadas. Ideal para

quando os módulos padrão não atendem a todos os requisitos do projeto.

- **Exemplo de uso:**
 - Cadastro de produtos (carros, imóveis) em uma gamificação de vendas.
 - Listagem de pontos turísticos em uma gamificação de turismo.
-

Csv Data (Dados CSV)

- **API:** [/v3/csv](#)
 - **Função:** Importação e exportação de dados no formato CSV.
 - **Descrição:**

Permite importar ou exportar dados em CSV de forma fácil e segura, seja via upload manual ou FTP. Suporta arquivos criptografados, garantindo segurança no trânsito das informações.
 - **Exemplo de uso:**
 - Importação de listas de jogadores.
 - Exportação de relatórios de desempenho para análise externa.
 - Importação segura de dados criptografados via FTP.
-

Upload (Upload)

- **API:** [/v3/upload](#)
- **Função:** Upload de arquivos para uso na gamificação.
- **Descrição:**

Permite enviar imagens, documentos e outros tipos de arquivos para utilização em avatares, desafios, recursos visuais ou qualquer outra funcionalidade que torne a experiência mais rica.
- **Exemplo de uso:**
 - Upload de imagens para avatares personalizados.
 - Envio de documentos para utilização em treinamentos e desafios.

Backup (Backup)

- **Acesso:** `/studio/backup`
- **API:** `/v3/backup`
- **Função:** Backup e remoção controlada de dados antigos.
- **Descrição:**

Permite criar rotinas automáticas de backup de registros importantes, além de remover dados antigos de maneira controlada para manter a performance da plataforma. Garante segurança, integridade e bom desempenho mesmo em grandes volumes de dados.
- **Exemplo de uso:**
 - Backup de logs antigos antes de uma atualização.
 - Remoção de registros históricos para otimizar consultas.

Compact (Compactação)

- **Acesso:** `/studio/compact`
- **API:** `/v3/compact`
- **Função:** Compactação e otimização do armazenamento de dados.
- **Descrição:**

Configura rotinas para agrupar registros e reduzir o espaço ocupado por dados históricos, como achievements ou logs, melhorando o tempo de resposta das consultas e a performance geral do sistema.
- **Exemplo de uso:**
 - Compactação de registros de achievements para reduzir volume de dados.
 - Otimização de coleções de acesso rápido para performance superior.

Módulos de Ambientes e Publicação

A Funifier permite aos administradores e desenvolvedores gerenciar diferentes ambientes, testar novas funcionalidades com segurança e reutilizar componentes já validados em outras gamificações. Esses módulos são essenciais para projetos de missão crítica, grandes corporações ou qualquer operação que precise garantir qualidade, padronização e agilidade na publicação de soluções gamificadas.

Staging (Staging)

- **Acesso:** </studio/integration/staging>
 - **Função:** Gerenciamento de ambientes de homologação e produção, com clonagem e migração de dados.
 - **Descrição:**

Permite criar e gerenciar ambientes de homologação (staging) separados do ambiente de produção. Possibilita clonar o ambiente produtivo para o de testes e migrar dados entre servidores localizados em diferentes regiões. Dessa forma, é possível testar novas funcionalidades, validar atualizações e garantir que os lançamentos ocorram sem riscos para o ambiente principal.
 - **Exemplo de uso:**
 - Criação de ambiente de testes espelhando o ambiente real para validar novidades sem impactar os usuários.
 - Migração de dados entre servidores em diferentes continentes para atender a necessidades regionais ou compliance.
-

Marketplace (Mercado)

- **Acesso:** </market>
- **Função:** Compartilhamento e reutilização de componentes prontos para gamificações.
- **Descrição:**

Disponibiliza um catálogo de componentes já configurados — como desafios, rankings, widgets, modelos de gamificação e outros — que podem ser reutilizados em diferentes projetos, acelerando a implementação e evitando retrabalho. Usuários podem empacotar e compartilhar soluções para toda a organização ou para o ecossistema Funifier.
- **Exemplo de uso:**

- Compartilhamento de um modelo de gamificação corporativa para uso em diferentes departamentos ou unidades de negócio.
 - Reutilização de conjuntos de desafios e widgets já validados em múltiplos projetos.
-

Anexo II - API Funifier

A plataforma Funifier disponibiliza uma API RESTful que permite a integração e o controle total das funcionalidades da gamificação por sistemas externos. Através desta API, é possível realizar operações como autenticação de jogadores, registro de ações, atribuição de recompensas, consulta e manipulação de dados, entre outras. Abaixo está a documentação dos principais recursos da API, com seus respectivos endpoints, métodos suportados, descrições e exemplos de uso. Essa estrutura permite que desenvolvedores implementem soluções gamificadas de forma flexível, segura e escalável.

Recurso: **Player**

Este recurso representa os jogadores que interagem com a gamificação. Permite criar, consultar, excluir e monitorar o progresso dos jogadores.

Listar Jogadores

Método: GET

Endpoint: /v3/player

Descrição: Retorna todos os jogadores cadastrados na gamificação.

Exemplo de Resposta:

```
[
  {
    "_id": "jerry",
    "name": "Jerry",
    "email": "jerry@yourdomain.com",
    "extra": {
      "company": "Tom & Jerry Inc."
    },
    "created": 1694990893810,
    "updated": 1694990893880
  }
]
```

Criar Jogador

Método: POST

Endpoint: /v3/player

Descrição: Cria um novo jogador com atributos personalizados.

Exemplo de Body da Requisição:

```
{
  "_id": "jerry",
  "name": "Jerry",
  "email": "jerry@yourdomain.com",
  "image": {
    "small": { "url": "https://my.funifier.com/images/funny.png" },
    "medium": { "url": "https://my.funifier.com/images/funny.png" },
    "original": { "url": "https://my.funifier.com/images/funny.png" }
  },
  "teams": ["cartoon"],
  "friends": ["tom", "spike", "quacker"],
  "extra": {
    "country": "USA",
    "company": "Tom & Jerry Inc.",
    "sports": ["soccer", "cycling", "surf"]
  }
}
```

Exemplo de Resposta:

```
{
  "_id": "jerry",
  "name": "Jerry",
  "email": "jerry@yourdomain.com",
  "created": 1695216470091,
  "updated": 1695216470091
}
```

Excluir Jogador

Método: DELETE

Endpoint: /v3/player/:id

Descrição: Remove o jogador e todas as suas informações da gamificação. Substitua **:id** pelo identificador do jogador.

Consultar Status do Jogador

Método: GET

Endpoint: /v3/player/:id/status

Descrição: Retorna as estatísticas do jogador, como total de pontos, nível atual, desafios concluídos e itens adquiridos. Substitua **:id** pelo ID do jogador.

Exemplo de Resposta:

```
{
```

```
"name": "Jerry",
"total_challenges": 0,
"total_points": 0,
"level_progress": {
  "percent_completed": 0,
  "next_level": {
    "level": "Apprentice",
    "minPoints": 10
  }
},
"_id": "jerry"
}
```

Recurso: Team

Este recurso representa as equipes, ou grupos de jogadores, dentro da gamificação. Permite criar, consultar, gerenciar membros e acompanhar o desempenho das equipes.

Listar Equipes

Método: GET

Endpoint: /v3/team

Descrição: Retorna todas as equipes cadastradas na gamificação.

Criar Equipe

Método: POST

Endpoint: /v3/team

Descrição: Cria uma nova equipe, com nome, descrição, imagem e dono.

Exemplo de Body da Requisição:

```
{
  "_id": "sales",
  "name": "Sales",
  "description": "Funifier sales team",
  "image": {
    "small": {"url": "https://my.funifier.com/images/funny.png"},
    "medium": {"url": "https://my.funifier.com/images/funny.png"},
    "original": {"url": "https://my.funifier.com/images/funny.png"}
  },
  "extra": {
    "country": "USA"
  },
  "owner": "john"
}
```

Exemplo de Resposta:

```
{
  "_id": "sales",
```



```
{
  "name": "Sales",
  "description": "Funifier sales team",
  "image": {
    "small": {"url": "https://my.funifier.com/images/funny.png"},
    "medium": {"url": "https://my.funifier.com/images/funny.png"},
    "original": {"url": "https://my.funifier.com/images/funny.png"}
  },
  "extra": {
    "country": "USA"
  },
  "owner": "john",
  "created": 1695135261192,
  "updated": 1695135261192
}
```

Excluir Equipe

Método: DELETE

Endpoint: `/v3/team/:id`

Descrição: Remove a equipe especificada pelo ID.

Adicionar Membro à Equipe

Método: GET

Endpoint: `/v3/team/:id/member/add/:player`

Descrição: Adiciona um jogador à equipe. Substitua `:id` pelo ID da equipe e `:player` pelo ID do jogador.

Exemplo de Resposta:

```
{
  "team": "sales",
  "player": "tom"
}
```

Remover Membro da Equipe

Método: GET

Endpoint: `/v3/team/:team_id/member/remove/:player_id`

Descrição: Remove um jogador da equipe. Substitua os parâmetros pela equipe e jogador desejado.

Listar IDs dos Membros

Método: GET

Endpoint: `/v3/team/:team_id/memberids`

Descrição: Lista apenas os identificadores dos jogadores de uma equipe.

Exemplo de Resposta:

[

```
"tom"  
]
```

Consultar Status da Equipe

Método: GET

Endpoint: /v3/team/:id/status

Descrição: Retorna estatísticas da equipe, como pontos totais, progresso em desafios e itens adquiridos.

Exemplo de Resposta:

```
{  
  "_id": "sales",  
  "name": "Sales",  
  "image": {  
    "small": {"url": "https://my.funifier.com/images/funny.png"},  
    "medium": {"url": "https://my.funifier.com/images/funny.png"},  
    "original": {"url": "https://my.funifier.com/images/funny.png"}  
  },  
  "total_challenges": 0,  
  "total_points": 0,  
  "point_categories": {},  
  "catalog_items": {},  
  "extra": {  
    "country": "USA"  
  }  
}
```

Recurso: Auth

Este recurso é responsável por autenticar jogadores na gamificação, retornando um token de acesso que poderá ser usado nas requisições seguintes.

Autenticar Jogador

Método: POST

Endpoint: /v3/auth/token

Descrição: Autentica o jogador usando nome de usuário e senha, retornando um token de acesso.

Exemplo de Body da Requisição:

```
{  
  "apiKey": "YOUR_API_KEY",  
  "grant_type": "password",  
  "username": "tom",  
  "password": "123"  
}
```

Exemplo de Resposta:

```
{
  "access_token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXZWQlbn0...",
  "token_type": "Bearer",
  "expires_in": 1695751444626
}
```

Recurso: Action

Este recurso representa as ações que os jogadores podem realizar dentro da gamificação. Exemplos comuns de ações são: "vender", "comprar", "comentar", "assistir vídeo", entre outras. Cada ação pode conter atributos personalizados como produto, valor, localização etc.

Listar Ações

Método: GET

Endpoint: /v3/action

Descrição: Retorna a lista de ações configuradas na gamificação.

Exemplo de Resposta:

```
[
  {
    "action": "Sell",
    "attributes": [
      { "name": "product", "type": "String" },
      { "name": "price", "type": "Number" }
    ],
    "active": true,
    "_id": "sell"
  },
  {
    "action": "Watch Video",
    "active": true,
    "_id": "watch_video"
  }
]
```

Criar Ação

Método: POST

Endpoint: /v3/action

Descrição: Cria uma nova ação com nome, atributos e status de ativação.

Exemplo de Body da Requisição:

```
{
  "action": "Sell",
  "attributes": [
    { "name": "product", "type": "String" },
  ]
}
```

```
{ "name": "price", "type": "Number" }
],
"active": true,
"_id": "sell"
}
```

Exemplo de Resposta:

```
{
  "action": "Sell",
  "attributes": [
    { "name": "product", "type": "String" },
    { "name": "price", "type": "Number" }
  ],
  "active": true,
  "_id": "sell"
}
```

Excluir Ação

Método: DELETE

Endpoint: /v3/action/:id

Descrição: Remove uma ação da gamificação. Substitua :id pelo identificador da ação.

Recurso: ActionLog

Este recurso registra quando uma ação é realmente executada por um jogador. Cada log de ação contém o identificador da ação, do jogador, os atributos envolvidos e o horário da execução. É útil para rastrear o comportamento dos jogadores em tempo real.

Listar ActionLogs

Método: GET

Endpoint: /v3/action/log

Descrição: Retorna a lista de ações realizadas pelos jogadores, incluindo atributos como data, jogador e detalhes da ação.

Exemplo de Resposta:

```
[
  {
    "_id": "6509e4e28325771ffaa4506e",
    "actionId": "sell",
    "userId": "jerry",
    "time": 1695147234712,
    "attributes": {
      "product": "bike",
      "price": 1200
    }
  },
  {

```

```
{
  "_id": "6509e4d18325771ffaa4505c",
  "actionId": "sell",
  "userId": "tom",
  "time": 1695147217658,
  "attributes": {
    "product": "book",
    "price": 25
  }
}
```

Registrar ActionLog

Método: POST

Endpoint: /v3/action/log

Descrição: Registra a execução de uma ação por um jogador.

Exemplo de Body da Requisição:

```
{
  "actionId": "sell",
  "userId": "jerry",
  "attributes": {
    "product": "bike",
    "price": 1200
  }
}
```

Exemplo de Resposta:

```
{
  "actionId": "sell",
  "userId": "jerry",
  "time": 1695147341932,
  "attributes": {
    "product": "bike",
    "price": 1200
  },
  "_id": "6509e54d8325771ffaa4509a"
}
```

Registrar Múltiplos ActionLogs

Método: POST

Endpoint: /v3/action/log/bulk

Descrição: Permite registrar múltiplas ações de jogadores em uma única requisição.

Exemplo de Body da Requisição:

```
[
  {
    "actionId": "sell",
```

```

    "userId": "jerry",
    "attributes": {
      "product": "bike",
      "price": 1200
    }
  },
  {
    "actionId": "sell",
    "userId": "tom",
    "attributes": {
      "product": "book",
      "price": 25
    }
  }
]

```

Exemplo de Resposta:

```

{
  "content_size": 2,
  "total_registered": 2,
  "content": [
    {
      "actionId": "sell",
      "userId": "jerry",
      "time": 1695147549965,
      "attributes": {
        "product": "bike",
        "price": 1200
      }
    },
    "_id": "6509e61d8325771ffaa450de"
  ],
  {
    "actionId": "sell",
    "userId": "tom",
    "time": 1695147549969,
    "attributes": {
      "product": "book",
      "price": 25
    }
  },
  "_id": "6509e61d8325771ffaa450df"
]
}

```

Excluir ActionLog

Método: DELETE

Endpoint: /v3/action/log/:id

Descrição: Remove um registro de ação específica, informando o ID do ActionLog.

Recurso: Achievement

Reconhecimentos que os jogadores recebem ao atingir certos marcos, como completar um desafio, subir de nível, ganhar pontos, comprar um item da loja virtual.

Listar Conquistas

Método: GET

Endpoint: `/v3/achievement`

Descrição: Retorna a lista de conquistas obtidas pelos jogadores.

Exemplo de Resposta:

```
[
  {
    "_id": "6509e4e28325771ffaa4506f",
    "player": "jerry",
    "total": 1,
    "type": 1,
    "item": "DTqtTI9",
    "time": 1695147234712
  }
]
```

Recurso: Point

Pontos são unidades de medida do status do jogador. Uma gamificação pode ter vários tipos de pontos diferentes.

Listar Pontos

Método: GET

Endpoint: `/v3/point`

Descrição: Retorna todos os tipos de pontos disponíveis na gamificação.

Exemplo de Resposta:

```
[
  {
    "category": "Exchangeable Coins",
    "shortName": "Coins",
    "extra": {},
    "techniques": ["GT75"],
    "_id": "coin"
  }
]
```

Criar Ponto

Método: POST

Endpoint: `/v3/point`

Descrição: Cria uma nova categoria de pontos.

Exemplo de Body da Requisição:

```
{
  "category": "Experience Points",
  "shortName": "XP",
  "extra": {},
  "techniques": ["GT01"],
  "_id": "xp"
}
```

Excluir Ponto

Método: DELETE

Endpoint: `/v3/point/:id`

Descrição: Exclui um tipo de ponto com base no seu ID.

Recurso: Challenge

Tarefas que os jogadores precisam executar. No desafio são configuradas as ações que o jogador precisa realizar e as recompensas que receberá.

Listar Desafios

Método: GET

Endpoint: `/v3/challenge`

Descrição: Retorna todos os desafios configurados no sistema.

Exemplo de Resposta:

```
[
  {
    "challenge": "Sell 10 books",
    "description": "Sell 10 books to earn 25 xp and 5 coins",
    "rules": [
      {
        "actionId": "sell",
        "filters": [
          { "value": "book", "operator": 1, "param": "product" }
        ],
        "total": 10
      }
    ],
    "points": [
      { "total": 25, "category": "xp" },
      { "total": 5, "category": "coin" }
    ],
    "_id": "DTkhJHV"
  }
]
```

Criar Desafio

Método: POST

Endpoint: `/v3/challenge`

Descrição: Cria um novo desafio.

Exemplo de Body da Requisição:

```
{
  "challenge": "Watch Video",
  "description": "Complete this challenge by watching a video, and earn 10 xp",
  "rules": [
    { "actionId": "watch_video", "operator": 5, "total": 0 }
  ],
  "points": [
    { "total": 10.0, "category": "xp", "operation": 0 }
  ],
  "techniques": ["GT35"]
}
```

Excluir Desafio

Método: DELETE

Endpoint: `/v3/challenge/:id`

Descrição: Exclui um desafio com base no ID informado.

Recurso: Level

Sistema de evolução do jogador, com múltiplos níveis baseados em pontos acumulados e critérios definidos.

Listar níveis

Método: GET

Endpoint: `/v3/level`

Retorna a lista de todos os níveis configurados no sistema.

Criar nível

Método: POST

Endpoint: `/v3/level`

Corpo do JSON:

```
{
  "level": "Apprentice",
  "position": 0,
  "description": "It indicates that you are learning, growing and striving to reach your goals.",
  "minPoints": 10,
  "notifications": [],
  "requirements": [],
  "i18n": {},
  "extra": {}
}
```

```
"techniques": ["GT85"],
"_id": "L0"
}
```

Excluir nível

Método: **DELETE**

Endpoint: **/v3/level/:id**

Remove um nível específico baseado em seu **_id**.

Listar configuração global de pontos para níveis

Método: **GET**

Endpoint: **/v3/database/level_config**

Retorna a configuração atual da categoria de pontos usada para calcular a progressão dos níveis.

Atualizar configuração global de pontos para níveis

Define a categoria de ponto usada para calcular a evolução de níveis.

Método: **PUT**

Endpoint: **/v3/database/level_config**

Corpo do JSON:

```
{
  "_id": "global",
  "pointCategory": "xp"
}
```

Atualizar posições dos níveis

Recalcula automaticamente as posições dos níveis com base no valor de **minPoints** de cada um.

Método: **PUT**

Endpoint: **/v3/level/position**

Recurso: **Leaderboard**

Este recurso permite configurar rankings baseados em resultados de jogadores. É possível definir como os líderes serão calculados, qual a métrica de avaliação (como pontos, moedas ou ações) e o período de apuração (semanal, mensal, tempo real, etc).

Listar Leaderboards

Método: **GET**

Endpoint: **/v3/leaderboard**

Descrição: Retorna a lista de leaderboards configurados no projeto.

Exemplo de Resposta:

```
[
  {
    "title": "Top Players",
    "description": "Players with the most points xp in the week",
    "principalType": 0,
    "operation": {
      "type": 3,
      "achievement_type": 0,
      "item": "xp",
      "filters": [],
      "sort": -1,
      "sub": false
    },
    "period": {
      "type": 0,
      "timeAmount": 1,
      "timeScale": 6
    },
    "techniques": ["GT03"],
    "_id": "DTjTvZ5"
  }
]
```

Criar Leaderboard

Método: **POST**

Endpoint: **/v3/leaderboard**

Descrição: Cria um novo leaderboard com critérios de cálculo e período definidos.

Exemplo de Body da Requisição:

```
{
  "title": "Top Players",
  "description": "Players with the most points xp in the week",
  "principalType": 0,
  "operation": {
    "type": 3,
    "achievement_type": 0,
    "item": "xp",
    "filters": [],
    "sort": -1,
    "sub": false
  },
  "period": {
    "type": 0,
    "timeAmount": 1,

```

```
    "timeScale": 6
  },
  "techniques": ["GT03"],
  "_id": "DTjTvZ5"
}
```

Deletar Leaderboard

Método: DELETE

Endpoint: /v3/leaderboard/:id

Descrição: Remove um leaderboard existente com base no seu ID.

Obter Lista de Líderes (Aggregate)

Método: POST

Endpoint: /v3/leaderboard/:id/leader/aggregate?period=&live=true

Descrição: Retorna a lista de líderes de um leaderboard, podendo usar agregações personalizadas. Parâmetros opcionais na URL incluem:

- **period:** período específico de apuração.
- **live:** define se o ranking deve ser recalculado em tempo real (true/false).

Exemplo de Resposta:

```
[
  {
    "_id": "tom_D0zCMvq",
    "total": 30,
    "position": 1,
    "move": "up",
    "player": "tom",
    "name": "Tom",
    "extra": { "cache": "D0zCMvq" },
    "boardId": "DTjTvZ5"
  },
  {
    "_id": "jerry_D0zCMvq",
    "total": 30,
    "position": 2,
    "move": "up",
    "player": "jerry",
    "name": "Jerry",
    "image": "https://my.funifier.com/images/funny.png",
    "extra": { "cache": "D0zCMvq" },
    "boardId": "DTjTvZ5"
  }
]
```

Resetar Cache de Leaderboards

Método: GET

Endpoint: `/v3/leaderboard/reset`

Descrição: Limpa o cache de resultado de líderes de todos os leaderboards.

Exemplo de Resposta:

```
{
  "status": "OK"
}
```

Recurso: Virtual Good

Módulo responsável por configurar catálogos de recompensas (lojas) em que jogadores podem trocar pontos ou moedas por itens físicos ou virtuais. Cada item pode ter quantidade limitada, regras para compra e um limite de aquisições por jogador.

Listar Catálogos

Método: `GET`

Endpoint: `/v3/virtualgoods/catalog`

Descrição: Retorna a lista de catálogos de itens virtuais cadastrados.

Criar Catálogo

Método: `POST`

Endpoint: `/v3/virtualgoods/catalog`

Descrição: Cria um novo catálogo de itens virtuais.

Exemplo de Body da Requisição:

```
{
  "catalog": "Gifts",
  "extra": {},
  "i18n": {},
  "_id": "gifts"
}
```

Excluir Catálogo

Método: `DELETE`

Endpoint: `/v3/virtualgoods/catalog/:id`

Descrição: Remove um catálogo de itens virtuais.

Listar Itens

Método: `GET`

Endpoint: `/v3/virtualgoods/item`

Descrição: Retorna todos os itens cadastrados nos catálogos virtuais.

Criar Item

Método: POST

Endpoint: /v3/virtualgoods/item

Descrição: Cria um novo item no catálogo de recompensas.

Exemplo de Body da Requisição:

```
{
  "catalogId": "gifts",
  "name": "Bike",
  "description": "Eletric bike",
  "amount": -1,
  "active": true,
  "extra": {},
  "requires": [],
  "rewards": [],
  "notifications": [],
  "i18n": {},
  "techniques": ["GT08"],
  "_id": "DTj7IVn"
}
```

Excluir Item

Método: DELETE

Endpoint: /v3/virtualgoods/item/:id

Descrição: Remove um item do catálogo de recompensas.

Realizar Compra

Método: POST

Endpoint: /v3/virtualgoods/purchase

Descrição: Permite que um jogador compre um item do catálogo, debitando os pontos e creditando o item.

Exemplo de Body da Requisição:

```
{
  "player": "tom",
  "item": "DTj7IVn",
  "total": 1
}
```

Excluir Compra

Método: DELETE

Endpoint: /v3/virtualgoods/purchase

Descrição: Remove uma compra e desfaz os débitos e créditos gerados.

Exemplo de Body da Requisição:

```
{
  "_id": "650b0f2a8325771ffaa5bcfb"
}
```

Recurso: Lottery

Módulo para configurar sorteios. Quando o sorteio é realizado, as recompensas são dadas a um número de vencedores selecionado de forma aleatória. Por exemplo, um sorteio de uma viagem com apenas um ganhador, que será sorteado em uma data definida pelo administrador. Para participar, o jogador acumula cupons de sorteio.

Listar Sorteios

Método: GET

Endpoint: /v3/lottery

Descrição: Retorna a lista de sorteios configurados no sistema.

Exemplo de Resposta:

```
[
  {
    "title": "Travel to Cancun",
    "description": "Travel with a companion to Cancun, with airfare and accommodation, for 7 days.",
    "drawDate": 1690824650503,
    "autoExecute": true,
    "choiceMethod": "random_ticket",
    "maxWinners": 1,
    "maxPerPlayer": 1,
    "techniques": ["GT74"],
    "rewards": [
      {
        "total": 1,
        "type": 2,
        "item": "flight_ticket"
      }
    ],
    "_id": "DTj0x5z"
  }
]
```

Criar Sorteio

Método: POST

Endpoint: /v3/lottery

Descrição: Cria um novo sorteio com as configurações desejadas.

Exemplo de Body da Requisição:

```
{
```

```
"title": "Travel to Cancun",
"description": "Travel with a companion to Cancun, with airfare and accommodation, for 7
days.",
"drawDate": 1690824650503,
"autoExecute": true,
"choiceMethod": "random_ticket",
"maxWinners": 1,
"maxPerPlayer": 1,
"techniques": ["GT74"],
"rewards": [
  {
    "total": 1,
    "type": 2,
    "item": "flight_ticket"
  }
],
"_id": "DTj0x5z"
}
```

Deletar Sorteio

Método: DELETE

Endpoint: /v3/lottery/:id

Descrição: Remove um sorteio específico da plataforma.

Listar Cupons

Método: GET

Endpoint: /v3/lottery/ticket?lottery=DTj0x5z

Descrição: Lista todos os cupons emitidos para um sorteio específico.

Criar Cupom

Método: POST

Endpoint: /v3/lottery/ticket

Descrição: Emite um novo cupom para um jogador participar do sorteio.

Exemplo de Body da Requisição:

```
{
  "lottery": "DTj0x5z",
  "player": "tom"
}
```

Deletar Cupom

Método: DELETE

Endpoint: /v3/lottery/ticket/:id

Descrição: Remove um cupom de sorteio específico.

Executar Sorteio

Método: GET

Endpoint: `/v3/lottery/DTj0x5z/execute`

Descrição: Executa o sorteio e seleciona os vencedores aleatoriamente.

Reverter Execução do Sorteio

Método: DELETE

Endpoint: `/v3/lottery/DTj0x5z/execute`

Descrição: Reverte a execução de um sorteio, permitindo realizar novamente.

Listar Vencedores

Método: GET

Endpoint: `/v3/lottery/winner?lottery=DTj0x5z`

Descrição: Retorna a lista de vencedores de um sorteio executado.

Listar Participantes

Método: GET

Endpoint: `/v3/lottery/participants?lottery=DTj0x5z`

Descrição: Mostra os participantes, quantidade de cupons, ganhadores e dados do sorteio.

Recurso: Competition

Módulo para promover disputas entre dois ou mais jogadores em um determinado período, avaliando quem atinge os melhores resultados. As competições podem ter número máximo de participantes e de vencedores, além de regras de pontuação e critérios de ranqueamento definidos.

Listar Competições

Método: GET

Endpoint: `/v3/competition`

Descrição: Retorna a lista de competições configuradas.

Criar Competição

Método: POST

Endpoint: `/v3/competition`

Descrição: Cria uma nova competição entre jogadores.

Exemplo de Body da Requisição:

```
{  
  "title": "Sales Race",
```

```
"description": "Leads the competition who has the highest amount of closed sales in the month.",
"period": {
  "expression": "-0M-;+1M+"
},
"maxWinners": 1,
"maxPlayers": 100,
"minScore": 0,
"operation": {
  "type": 1,
  "achievement_type": 0,
  "item": "sell",
  "filters": [],
  "sort": -1,
  "sub": false
},
"requires": [
  {
    "total": 10,
    "type": 0,
    "item": "coin",
    "operation": 1,
    "extra": {},
    "restrict": false,
    "perPlayer": false
  }
],
"rewards": [
  {
    "total": 100,
    "type": 0,
    "item": "xp",
    "operation": 0,
    "extra": {
      "position_start": 1,
      "position_ends": 3
    },
    "restrict": false,
    "perPlayer": false
  }
],
"notifications": [],
"active": true,
"teamCompetition": false,
"autoExecute": true,
"extra": {},
"techniques": ["GT26"],
"_id": "race"
}
```

Deletar Competição

Método: DELETE

Endpoint: /v3/competition/:id

Descrição: Remove uma competição específica da plataforma.

Listar Participações (Joins)

Método: GET

Endpoint: /v3/competition/join?competition=race

Descrição: Retorna a lista de jogadores inscritos em uma competição.

Criar Participação (Join)

Método: POST

Endpoint: /v3/competition/join

Descrição: Inscreve um jogador em uma competição.

Exemplo de Body da Requisição:

```
{
  "competition": "race",
  "player": "jerry"
}
```

Deletar Participação (Join)

Método: DELETE

Endpoint: /v3/competition/join

Descrição: Remove a inscrição de um jogador em uma competição.

Exemplo de Body da Requisição:

```
{
  "competition": "race",
  "player": "tom"
}
```

Listar Líderes da Competição

Método: POST

Endpoint: /v3/competition/leader/aggregate?id=race

Descrição: Retorna a posição atual dos jogadores inscritos, com base na métrica definida.

Executar Competição

Método: GET

Endpoint: /v3/competition/:id/execute

Descrição: Finaliza a competição, calcula os resultados e premia os vencedores.

Reverter Execução da Competição

Método: DELETE

Endpoint: /v3/competition/:id/execute

Descrição: Reverte a execução da competição e remove os resultados registrados.

Recurso: Swap

Módulo que permite que jogadores troquem itens conquistados na gamificação com outros jogadores. Por exemplo, um jogador pode oferecer uma camiseta em troca de uma bola e 10 moedas. Pode ser usado para criar mercados de troca como de figurinhas. As trocas são criadas pelos próprios jogadores, e não pelo administrador.

Listar Trocas

Método: GET

Endpoint: /v3/database/swap

Descrição: Retorna a lista de trocas (swaps) disponíveis criadas pelos jogadores.

Criar Troca

Método: POST

Endpoint: /v3/swap

Descrição: Cria uma nova proposta de troca. O jogador define os itens que está oferecendo e os itens que deseja em troca.

Exemplo de Body da Requisição:

```
{
  "seller": "tom",
  "rewards": [
    {
      "total": 1,
      "type": 2,
      "item": "DTj7IVn"
    }
  ],
  "requires": [
    {
      "total": 2,
      "type": 0,
      "item": "coin"
    }
  ]
}
```

Deletar Troca

Método: DELETE

Endpoint: `/v3/swap/:id`

Descrição: Remove uma proposta de troca existente, identificada pelo seu ID.

Realizar Troca (Adquirir Swap)

Método: POST

Endpoint: `/v3/swap/acquire`

Descrição: Permite que um jogador aceite uma proposta de troca criada por outro jogador.

Exemplo de Body da Requisição:

```
{
  "buyer": "jerry",
  "swap": "650c67f88325771ffaa78a24"
}
```

Receber Recompensas da Troca

Método: POST

Endpoint: `/v3/swap/receive`

Descrição: O jogador que criou a troca (vendedor) recebe os itens oferecidos pelo comprador.

Exemplo de Body da Requisição:

```
{
  "swap": "650c67f88325771ffaa78a24"
}
```

Listar Contra Propostas

Método: POST

Endpoint: `/v3/database/swap_counter_offer/aggregate`

Descrição: Lista as contra propostas feitas por compradores para uma troca específica.

Exemplo de Body da Requisição:

```
[
  {
    "$match": {
      "swap": "650c683c8325771ffaa78a3f"
    }
  }
]
```

Criar Contra Proposta

Método: POST

Endpoint: `/v3/swap/counter/offer`

Descrição: Permite que um jogador proponha uma troca alternativa ao vendedor.

Exemplo de Body da Requisição:

```
{
```

```
"swap": "650c683c8325771ffaa78a3f",
"buyer": "jerry",
"offer": [
  {
    "total": 1,
    "type": 0,
    "item": "coin"
  }
]
```

Deletar Contra Proposta

Método: DELETE

Endpoint: </v3/swap/counter/offer/:id>

Descrição: Remove uma contra proposta previamente feita por um comprador.

Exemplo de Body da Requisição:

```
{
  "swap": "650c683c8325771ffaa78a3f",
  "buyer": "jerry",
  "offer": [
    {
      "total": 1,
      "type": 0,
      "item": "coin"
    }
  ]
}
```

Aceitar Contra Proposta

Método: POST

Endpoint: </v3/swap/counter/offer/accept>

Descrição: Permite que o vendedor aceite uma contra proposta feita por um comprador.

Exemplo de Body da Requisição:

```
{
  "offer": "650c69aa8325771ffaa78b1b"
}
```

Receber Recompensas de Contra Proposta

Método: POST

Endpoint: </v3/swap/counter/offer/receive>

Descrição: Permite que o vendedor receba os itens propostos por um comprador em uma contra proposta aceita.

Exemplo de Body da Requisição:

```
{
```

```
"swap": "650c683c8325771ffaa78a3f"
}
```

Recurso: Question

Módulo para criar perguntas para os jogadores. A apresentação da pergunta pode se dar no formato tradicional com a pergunta e as opções de resposta abaixo, mas também em formatos de mini games como por exemplo, um caça-palavras, uma cruzadinha, etc. Cada pergunta pode apresentar texto, imagens, conteúdos audiovisuais, e as respostas podem ser de múltipla escolha, verdadeiro ou falso, dissertação, etc.

Listar Perguntas

Método: GET

Endpoint: [/v3/question](#)

Descrição: Retorna a lista de perguntas configuradas.

Criar Pergunta

Método: POST

Endpoint: [/v3/question](#)

Descrição: Cria uma nova pergunta com opções de resposta.

Exemplo de Body da Requisição:

```
{
  "_id": "64a5b2c2d8dcca49bcf7eb6e",
  "type": "MULTIPLE_CHOICE",
  "title": "Visual Components",
  "question": "In Funifier, what is the small visual components that can be added to a web page to display feedbacks?",
  "grade": 1,
  "choices": [
    { "answer": "1", "label": "Points", "grade": 0, "extra": {} },
    { "answer": "2", "label": "Badges", "grade": 0, "extra": {} },
    { "answer": "3", "label": "Challenges", "grade": 0, "extra": {} },
    { "answer": "4", "label": "Leaderboards", "grade": 0, "extra": {} },
    { "answer": "5", "label": "Widgets", "grade": 1, "extra": {} }
  ],
  "techniques": ["GT05"],
  "select": "one_answer",
  "answerNumbering": "uppercase_letters",
  "shuffle": false,
  "feedbacks": [],
  "extra": {}
}
```

Deletar Pergunta

Método: DELETE

Endpoint: `/v3/question/:id`

Descrição: Remove uma pergunta existente pelo seu identificador.

Listar Respostas de Jogadores

Método: GET

Endpoint: `/v3/question/log?question=:id`

Descrição: Retorna o histórico de respostas dos jogadores para uma determinada pergunta.

Criar Resposta de Jogador

Método: POST

Endpoint: `/v3/question/log`

Descrição: Registra a resposta de um jogador a uma pergunta.

Exemplo de Body da Requisição:

```
{
  "question": "64a5b2c2d8dcca49bcf7eb6e",
  "answer": ["5"],
  "player": "tom"
}
```

Deletar Resposta de Jogador

Método: DELETE

Endpoint: `/v3/question/log/:id`

Descrição: Remove uma entrada de resposta registrada de um jogador.

Recurso: Quiz

Descrição

Módulo para agrupar várias perguntas em um bloco único a ser respondido pelos jogadores. Por exemplo, uma prova com várias questões. O administrador define o valor total em pontos e o sistema calcula a pontuação com base na performance. Os quizzes podem ser apresentados como mini games, como uma corrida espacial.

Listar Quizzes

Método: GET

URL: `/v3/database/quiz`

Descrição: Retorna todos os quizzes cadastrados.

Criar Quiz

Método: POST

URL: `/v3/quiz`

Descrição: Cria um novo quiz.

Exemplo de Requisição:

```
{
  "_id": "650c82fe832",
  "title": "Funifier Exam",
  "description": "Test your knowledge about the best gamification platform in the world.",
  "grade": 10,
  "i18n": {},
  "extra": {},
  "feedbacks": [],
  "questionNumbering": "uppercase_letters",
  "showGradeBeforeFinish": false,
  "shuffle": false
}
```

Deletar Quiz

Método: DELETE

URL: `/v3/quiz/{quiz_id}`

Descrição: Remove um quiz existente.

Listar Perguntas de um Quiz

Método: GET

URL: `/v3/quiz/{quiz_id}/question`

Descrição: Retorna todas as perguntas de um quiz específico.

Criar Pergunta para um Quiz

Método: POST

URL: `/v3/question`

Descrição: Adiciona uma nova pergunta ao quiz.

Exemplo de Requisição:

```
{
  "quiz": "650c82fe832",
  "type": "MULTIPLE_CHOICE",
  "title": "Best Platform",
  "question": "What is the best gamification platform?",
  "grade": 1,
  "choices": [
    { "answer": "1", "label": "Funifier", "grade": 1, "extra": {}, "gradePercent": 100, "gradeCheck": true },
    { "answer": "2", "label": "Points", "grade": 0, "extra": {}, "gradePercent": 0 },
    { "answer": "3", "label": "Badges", "grade": 0, "extra": {}, "gradePercent": 0 },
    { "answer": "4", "label": "Leaderboards", "grade": 0, "extra": {}, "gradePercent": 0 }
  ],
  "i18n": {},
  "select": "one_answer",
  "shuffle": false
}
```

Iniciar Quiz

Método: POST

URL: </v3/quiz/start>

Descrição: Inicia um quiz para um jogador.

Exemplo de Requisição:

```
{
  "quiz": "650c82fe832",
  "player": "tom"
}
```

Registrar Respostas em Lote

Método: POST

URL: </v3/question/log/bulk>

Descrição: Registra múltiplas respostas de uma só vez.

Exemplo de Requisição:

```
[
  {
    "quiz": "650c82fe832",
    "quiz_log": "650dc6168325771ffaa94098",
    "question": "650dc4d98325771ffaa93e5e",
    "answer": ["1"],
    "player": "tom"
  }
]
```

Finalizar Quiz

Método: POST

URL: </v3/quiz/finish>

Descrição: Finaliza a tentativa do jogador em um quiz.

Exemplo de Requisição:

```
{
  "quiz_log": "650dc6168325771ffaa94098"
}
```

Recurso: **Mystery Box**

Módulo para configurar prêmios aleatórios. O jogador não sabe o prêmio que irá ganhar. Por exemplo: uma raspadinha onde o jogador só descobre se ganhou algum prêmio ou não depois de raspar a cartela; um jogo de cara ou coroa, onde existe 50% de chance para cada opção; uma roda da fortuna, onde existem várias opções, algumas com prêmios legais e outras nem tanto. Você configura este módulo informando um título para a caixa surpresa, as opções possíveis e a probabilidade de cada uma ser escolhida, além das combinações vencedoras com seus respectivos prêmios.

Listar Mystery Boxes

Método: GET

Endpoint: </v3/mystery>

Descrição: Retorna a lista de caixas de prêmios aleatórios configuradas.

Criar Mystery Box

Método: POST

Endpoint: </v3/mystery>

Descrição: Cria uma nova caixa de prêmios aleatórios com suas opções e probabilidades.

Exemplo de Body da Requisição:

```
{
  "title": "Heads or Tails",
  "options": [
    { "title": "Heads", "value": "heads", "probability": 0.5 },
    { "title": "Tails", "value": "tails", "probability": 0.5 }
  ],
  "columns": 1,
  "requirements": [],
  "win_chart": [
    {
      "combination": ["heads"],
      "orderSensitive": false,
      "reward": {
        "total": 1,
        "type": 0,
        "item": "coin"
      }
    }
  ],
  "techniques": ["GT72"],
  "_id": "64a5b464d8dcca49bcf7edd0"
}
```

Deletar Mystery Box

Método: DELETE

Endpoint: /v3/mystery/{mystery_id}

Descrição: Remove uma caixa de prêmios aleatórios previamente configurada.

Avaliar Condições de Mystery Box

Método: GET

Endpoint: /v3/mystery/evaluate/{mystery_id}?player={player_id}

Descrição: Avalia se o jogador está apto a executar a Mystery Box, verificando limites e requisitos.

Executar Mystery Box

Método: GET

Endpoint: `/v3/mystery/execute/{mystery_id}?player={player_id}`

Descrição: Executa a lógica de sorteio de uma Mystery Box para um jogador específico e retorna o resultado.

Recurso: Folder

Módulo para criar pastas virtuais onde você pode incluir qualquer tipo de conteúdo. As pastas podem ser utilizadas, por exemplo, para construir um curso online, composto de módulos, onde cada módulo possui um conjunto de aulas, e cada aula possui um conjunto de conteúdos. Esse módulo permite monitorar o progresso do jogador dentro dessa árvore de diretórios e conteúdos, registrar logs de ações e aplicar técnicas de jogos para criar experiências educativas e engajadoras.

Listar Pastas

Método: GET

Endpoint: `/v3/database/folder`

Descrição: Retorna a lista de pastas existentes.

Criar Pasta

Método: POST

Endpoint: `/v3/folder`

Descrição: Cria uma nova pasta com título e pasta-pai opcional.

Exemplo de Body da Requisição:

```
{
  "title": "CoreDrives",
  "parent": "DOMmmNf"
}
```

Deletar Pasta

Método: DELETE

Endpoint: `/v3/folder/{folder_id}`

Descrição: Remove uma pasta existente a partir do seu identificador.

Obter Breadcrumb da Pasta

Método: POST

Endpoint: `/v3/folder/breadcrumb`

Descrição: Retorna o caminho hierárquico da pasta informada, desde a raiz até ela.

Exemplo de Body da Requisição:

```
{
  "folder": "CD1"
}
```

Listar Tipos de Conteúdo de Pasta

Método: GET

Endpoint: /v3/database/folder_content_type

Descrição: Retorna os tipos de conteúdo que podem ser adicionados às pastas.

Criar ou Atualizar Tipo de Conteúdo

Método: PUT

Endpoint: /v3/database/folder_content_type

Descrição: Define um novo tipo de conteúdo ou atualiza um tipo existente.

Exemplo de Body da Requisição:

```
{
  "_id": "text",
  "input": "formulary",
  "form": [
    {
      "name": "title",
      "type": "string",
      "title": "Title",
      "dropdown": {
        "type": "",
        "static": [],
        "dynamic": {}
      }
    },
    {
      "name": "content",
      "type": "text",
      "title": "Content",
      "dropdown": {
        "type": "",
        "static": [],
        "dynamic": {}
      }
    }
  ],
  "title": "Text",
  "entity": "text__c"
}
```

Recurso: Database

Permite acessar as coleções do banco de dados da gamificação. Cada gamificação tem seu próprio banco de dados. Através desse módulo é possível listar coleções existentes, consultar, inserir, atualizar e excluir dados de coleções, além de criar e gerenciar índices.

Listar Coleções

Método: GET

Endpoint: </v3/database/collections>

Descrição: Lista o nome de todas as coleções existentes no banco de dados da gamificação.

Listar Dados de uma Coleção

Método: GET

Endpoint: </v3/database/:collection>

Descrição: Retorna os dados da coleção especificada. O nome da coleção deve ser informado na URL.

Criar Registro em uma Coleção

Método: POST

Endpoint: </v3/database/:collection>

Descrição: Cria um novo registro em uma coleção.

Exemplo de Body da Requisição:

```
{
  "year": 2010,
  "fuel": "gasoline",
  "price": 50000,
  "name": "Civic",
  "description": "Honda Civic",
  "brand": "honda"
}
```

Atualizar Registro em uma Coleção

Método: PUT

Endpoint: </v3/database/:collection>

Descrição: Atualiza um registro existente em uma coleção.

Exemplo de Body da Requisição:

```
{
  "_id": "6553e49815865a7a732f5fc7",
  "year": 2010,
  "fuel": "gasoline",
  "price": 50000,
  "name": "Civic",
  "description": "Honda Civic",
  "brand": "honda"
}
```

Excluir Registro(s) de uma Coleção

Método: DELETE

Endpoint: /v3/database/:collection?q=_id:'contend_id'

Descrição: Exclui um ou mais registros de uma coleção, com base no parâmetro **q** informado na URL.

Executar Agregações

Método: POST

Endpoint: **/v3/database/:collection/aggregate?strict=true**

Descrição: Executa uma lista de comandos **aggregate** no banco de dados, seguindo a sintaxe do MongoDB.

Exemplo de Body da Requisição:

```
[
  { "$match": { "actionId": "sell", "userId": "jerry", "attributes.price": { "$gte": 10 } } },
  { "$limit": 1 }
]
```

Inserção em Massa (Bulk)

Método: POST

Endpoint: **/v3/database/:collection/bulk**

Descrição: Permite inserir múltiplos registros em uma coleção de uma só vez.

Exemplo de Body da Requisição:

```
[
  {
    "_id": "6553e49815865a7a732f5fc7",
    "year": 2010,
    "fuel": "gasoline",
    "price": 60000,
    "name": "Honda Civic",
    "description": "Honda Civic",
    "brand": "honda"
  },
  {
    "_id": "6553e49815865a7a732f5fc8",
    "year": 2012,
    "fuel": "gasoline",
    "price": 70000,
    "name": "Toyota Corola",
    "description": "Toyota Corola",
    "brand": "toyota"
  }
]
```

Listar Índices

Método: GET

Endpoint: **/v3/database/:collection/index**

Descrição: Lista os índices existentes em uma determinada coleção.

Criar Índice

Método: POST

Endpoint: `/v3/database/:collection/index`

Descrição: Cria um novo índice para uma coleção.

Exemplo de Body da Requisição:

```
{
  "fuel": 1
}
```

Excluir Índice

Método: DELETE

Endpoint: `/v3/database/:collection/index/:index_name`

Descrição: Exclui um índice específico de uma coleção.

Recurso: Upload

Permite enviar arquivos (como imagens) para o servidor da gamificação, com suporte a envio via `FormData`, definição de nome e metadados adicionais no campo `extra`.

Upload de Imagem

Método: POST

Endpoint: `/v3/upload/image`

Descrição: Realiza o upload de uma imagem para o servidor da Funifier, utilizando um formulário multipart (`FormData`). É possível enviar dados adicionais através do campo `extra`.

Exemplo de Body da Requisição (via `FormData`):

- `file`: conteúdo da imagem (ex: `profile-pic.jpg`)
- `extra`: `{ "session": "images" }`

Headers:

- `Authorization`: `{token}`
- `Content-Type`: `multipart/form-data` (gerenciado automaticamente pelo `FormData`, portanto deve ser `undefined` no código)

Resposta esperada:

```
{
  "uploads": [
    {
      "url": "https://cdn.funifier.com/uploads/images/abc123/profile-pic.jpg"
    }
  ]
}
```

Anexo III - Aggregates Funifier

A plataforma Funifier utiliza o MongoDB como banco de dados, permitindo que desenvolvedores realizem consultas avançadas por meio do recurso de **aggregates**. Essa funcionalidade é fundamental para extrair relatórios, gerar indicadores personalizados e construir dashboards inteligentes a partir dos dados gerados pelas ações dos jogadores, seus desafios, recompensas e comportamento na plataforma.

Neste anexo, apresentamos a estrutura das coleções mais utilizadas na plataforma, exemplos práticos de comandos aggregate e explicações detalhadas sobre a sintaxe e os filtros específicos oferecidos pela Funifier — como as expressões de datas relativas (ex: `-0M-` para o início do mês atual). Este conteúdo é essencial para os candidatos à certificação Funifier Gamification Developer I, pois oferece o conhecimento necessário para construir queries eficientes que viabilizam análises profundas e decisões orientadas por dados dentro de projetos gamificados.

Os comandos aggregates são executados em uma coleção do MongoDB. O nome da coleção é informado na url que tem o seguinte formato `/v3/database/{collection}/aggregate`. Os comandos aggregate são enviados no corpo da requisição no formato JSON, usando o método POST. Por exemplo, se você quiser executar um aggregate na coleção "achievement" para somar quantos pontos xp o jogador john teve na gamificação faça a requisição abaixo:

```
# Soma o total de pontos xp do john na coleção achievement
# POST /v3/database/achievement/aggregate
[
  { "$match": { "player": "john", "type": 0, "item": "xp" } },
  { "$group": { "_id": null, "total": { "$sum": "$total" } } }
]
```

ESTRUTURA DOS OBJETOS DAS COLEÇÕES

Abaixo temos uma lista das principais coleções e estrutura de seus objetos:

Coleção **"player"** onde estão todos os jogadores da gamificação. Neste exemplo temos o jogador John Travolta, com login John, e-mail john@funifier.com, que pertence a equipe de vendas, é dos Estados Unidos, está no departamento de IT e foi criado em 5 de julho de 2023. Esta é a estrutura dos objetos dentro desta coleção:

```
{ "_id": "john", "name": "John Travolta", "email": "john@funifier.com", "teams": ["sales"], "extra":
{"country": "USA", "department": "IT"}, "created": { "$date": "2023-07-05T20:57:25.776Z"},
"updated": { "$date": "2023-07-05T20:57:25.777Z"}}
```

Coleção **"action"** onde estão as configurações dos tipos de ações; No exemplo temos uma ação vender, com dois atributos, produto do tipo string e preço do tipo número, e essa ação está ativa. Esta é a estrutura dos objetos dentro desta coleção:

```
{"_id": "sell", "action": "Sell", "attributes": [{"name": "product", "type": "String"}, {"name": "price", "type": "Number"}], "active": true}
```

Coleção **"action_log"** onde ficam todas as ações executadas pelos jogadores. No exemplo temos um registro de que o jogador John fez uma venda de um livro por 120 no dia 5 de julho de 2023. O campo userId se relaciona com o campo _id da coleção player. O campo actionId se relaciona com o campo _id da coleção action. Esta é a estrutura dos objetos dentro desta coleção:

```
{"_id": "64a5d92", "actionId": "sell", "userId": "john", "time": {"$date": "2023-07-05T20:57:33.303Z"}, "attributes": {"product": "book", "price": 120}}
```

Coleção **"achievement"** onde estão as conquistas dos jogadores, como um desafio concluído, um nível alcançado, pontos que ganhou, item que comprou; No exemplo abaixo John conquistou 25 pontos XP no dia 5 de julho de 2023. O campo type indica o tipo da conquista, que pode ser 0 para point, 1 para challenge, 2 para virtual good, 3 para level. O campo item indica o id do item que foi conquistado. Que pode ser o id de um point, level, challenge, etc. O campo player se relaciona com o campo _id da coleção player. Esta é a estrutura dos objetos dentro desta coleção:

```
{"_id": "64a5d2", "player": "john", "total": 25.0, "type": 0, "item": "xp", "time": {"$date": "2023-07-05T20:57:33.303Z"}}
```

Coleção **"challenge"** onde estão os desafios da gamificação. Neste exemplo temos o desafio Assistir Vídeo que dá 10 pontos XP como recompensa. Esta é a estrutura dos objetos dentro desta coleção:

```
{"challenge": "Watch Video", "active": true, "_id": "DTo8dS3", "description": "Complete this challenge by watching a video", "rules": [{"actionId": "watch_video", "operator": 5, "total": 0}], "points": [{"total": 10.0, "category": "xp", "operation": 0}]}
```

Coleção **"point_category"** onde estão os tipos de pontos da gamificação. Neste exemplo temos o ponto do tipo XP. Esta é a estrutura dos objetos dentro desta coleção:

```
{"_id": "xp", "category": "Experience Points", "shortName": "XP", "extra": {}}
```

Coleção **"catalog_item"** onde estão os itens da loja virtual. Neste exemplo temos uma camisa, que pode ser adquirida por 15 moedas, e no estoque existem apenas 100 disponíveis, este item foi criado no dia 5 de julho de 2023. Esta é a estrutura dos objetos dentro desta coleção:

```
{"_id": "prd1", "catalogId": "gifts", "name": "T-shirt", "description": "White t-shirt with the Funifier logo", "amount": 100, "active": true, "created": {"$date": "2023-07-05T17:10:58.416Z"}, "extra": {}, "requires": [{"total": 15, "type": 0, "item": "coin", "operation": 1}]}
```

EXEMPLOS

Abaixo temos alguns dos principais exemplos de comandos de agregação utilizados em projetos reais de gamificação usando a plataforma FUNIFIER.

EXEMPLO 1: AGGREGATE PARA CALCULAR OS 10 JOGADORES COM MAIS PONTOS XP NA GAMIFICAÇÃO NO MÊS ATUAL

Neste exemplo começamos o aggregate na coleção achievement que é onde estão as conquistas dos pontos, filtrando pelo item "xp" e nas datas estamos filtrando usando a expressão "-0M-" que indica o início do mês atual, e a expressão "-0M+" que indica o fim do mês atual. Esta expressão é específica da plataforma FUNIFIER. Mais a frente é feito um lookup na coleção player para recuperar o nome do jogador.

```
# POST /v3/database/achievement/aggregate
[
  {"$match":{"type":0, "item": "xp", "time":{"$gte":{"$date":"-0M-"}, "$lte":{"$date":"-0M+"}}}},
  {"$group":{"_id":"$player", "total":{"$sum":"$total"} }},
  {"$sort":{"total":-1 }},
  {"$lookup": {"from":"player", "localField":"_id", "foreignField":"_id", "as":"p" }},
  {"$unwind" : "$p" },
  {"$project":{" _id":1, "player":"$p.name", "total":1 }},
  {"$limit":10}
]
```

Expressões de Data (Funifier Syntax)

A expressão de data é composta de 3 partes obrigatórias e uma opcional. A primeira parte é o sinal de - ou + que indica se é para voltar no tempo ou avançar no tempo. A segunda parte é uma letra que indica a unidade de tempo. Letra "m" para minuto, "h" para hora, "d" para dia, "w" para semana, "M" para mês, e "y" para ano. A terceira parte é um número com a quantidade de tempo. Por exemplo -2d significa voltar 2 dias em relação ao momento atual. A quarta parte é o sinal de - ou + que indica se a data calculada deve recuar para o início ou avançar para o final da unidade de tempo. Por exemplo -1d- significa voltar 1 dia e recuar até a 00:00:00 que é o primeiro momento da unidade dia.

- -0d-: Início do dia atual (00:00:00)
- -0d+: Fim do dia atual (23:59:59)
- -1d: Agora, um dia atrás
- -0w- / -0w+: Semana atual (início/fim)
- -0M- / -0M+: Mês atual (início/fim)
- -1M-: Início do mês anterior

Formato: [sinal][unidade][quantidade][ajuste opcional]

Exemplo: -2w+ = Fim da semana retrasada

Aqui estão mais alguns exemplos de expressões de data que podem ser usadas nos filtros:

Expressão "-0d", agora.

Expressão "-0d-", 00:00:00 do dia atual

Expressão "-0d+", 23:59:59 do dia atual

Expressão "-1d", exactly one day before now

Expressão "-1d-", 00:00:00 the day before the current day

Expressão "-1d+", 23:59:59 the day before the current day

Expressão "-0w-", 00:00:00 on the most recent first day of the current week

Expressão "-0w+", 23:59:59 on the last day of the current week

Expressão "-1w", exactly one week before now

Expressão "-0M-", 00:00:00 on the first day of the month that the current day is

Expressão "-0M+", 23:59:59 on the last day of the month that the current day is

Expressão "-1M", exactly one month before now

EXEMPLO 2: AGGREGATE PARA SABER QUANTOS JOGADORES DISTINTOS COMPLETARAM UM CERTO DESAFIO

POST /v3/database/achievement/aggregate

```
[
  {"$match":{ "type":1, "item": "DTo8dS3" }},
  {"$group":{ "_id":"$player", "challenge":{"$first":"$item" } }},
  {"$group":{ "_id":"$challenge", "total_players":{"$sum":1} }},
  {"$lookup": { "from":"challenge", "localField":"_id", "foreignField":"_id", "as":"c" }},
  {"$project":{ "_id":1, "total_players":1, "challenge_name":{"$first":"$c.challenge" } } }
]
```

EXEMPLO 3: AGGREGATE PARA SABER A MÉDIA DE PONTOS XP GANHOS PELOS JOGADORES EM UM DETERMINADO MÊS

POST /v3/database/achievement/aggregate

```
[
  {"$match": { "type": 0, "item": "xp", "time": { "$gte": { "$date": "2023-08-01T00:00:00.000Z"},
"$lte": { "$date": "2023-08-31T23:59:59.999Z" } } } },
  {"$group": { "_id": "$player", "averageXP": { "$avg": "$total" } } }
]
```

EXEMPLO 4: AGGREGATE PARA SABER QUAIS FORAM OS 3 ITENS MAIS VENDIDOS NA LOJA VIRTUAL ESTE ANO

POST /v3/database/achievement/aggregate

```
[
  {"$match":{ "type":2, "time":{"$gte":{"$date":"-0y-"}, "$lte":{"$date":"-0y+" } } }},
  {"$group":{ "_id":"$item", "totalSold":{"$sum":"$total" } }},
]
```

```

{"$lookup": {"from": "catalog_item", "localField": "_id", "foreignField": "_id", "as": "ci" }},
{"$project": { "_id": 1, "catalogItemName": {"$first": "$ci.name"}, "totalSold": 1 }},
{"$sort": { "totalSold": -1 }},
{"$limit": 3}
]

```

EXEMPLO 5: AGGREGATE PARA SABER QUEM FORAM OS JOGADORES QUE EXECUTARAM AÇÕES ONTEM

POST /v3/database/action_log/aggregate

```

[
  {"$match": { "time": {"$gte": {"$date": "-1d-"}, "$lte": {"$date": "-1d+"}} }},
  {"$group": { "_id": "$userId" }},
  {"$lookup": {"from": "player", "localField": "_id", "foreignField": "_id", "as": "p" }},
  {"$project": { "_id": 1, "playerName": {"$first": "$p.name" } }}
]

```

Anexo IV - Triggers Funifier

Triggers são códigos JAVA executados dentro do FUNIFIER ENGINE quando um evento específico acontecer dentro da gamificação. As triggers oferecem aos desenvolvedores uma grande flexibilidade para manipular informações em tempo real, permitindo mudar o comportamento padrão das técnicas de jogos e demais funcionalidades da plataforma. Por exemplo, a funcionalidade padrão de cadastro de um jogador na gamificação, não faz nada além de cadastrar os dados do jogador no banco de dados. Porém, com a Trigger, o desenvolvedor pode escrever um código JAVA para enviar um email de boas vindas para o jogador, logo após ele ser cadastrado no banco de dados. Triggers podem ser configuradas através do FUNIFIER STUDIO ou pela FUNIFIER API.

O caminho para configuração de triggers no STUDIO é: **/studio/trigger**

O endpoint para configuração de triggers na API REST é: **/v3/trigger**

CONFIGURAÇÃO DE UMA TRIGGER

A configuração de uma trigger é composta de três informações essenciais: o evento que indica quando a trigger deve ser executada, a entidade que informa que tipo de objeto está vinculado àquele evento, e o script Java a ser executado. Veja abaixo um exemplo de uma trigger que irá alterar o nome do jogador para letra maiúscula, antes de criar o jogador no banco de dados. Neste exemplo o evento é **"before_create"** e a entidade é **"player"**.

```
# Trigger que alterar o nome do jogador para letra maiúscula antes do jogador ser criado
# POST /v3/trigger
{
  "name": "Make the player's name uppercase", "_id": "DTv7uHc",
  "description": "Before creating the player, change the letters of the name to uppercase",
  "entity": "player",
  "event": "before_create",
  "script": "void trigger(event, entity, player, database){ entity.name =
entity.name.toUpperCase(); }"
}
```

Os campos utilizados nesta configuração básica de trigger são:

- **"_id"** : Identificador único da trigger; Caso não seja informado o Funifier define um valor;
- **"name"**: Nome da trigger;
- **"description"**: Descreve o que a trigger faz quando é executada pela engine;
- **"entity"**: Qual entidade está sendo observada; Neste exemplo estamos observando a entidade player;
- **"event"**: Evento que irá acionar a trigger quando ele ocorrer na entidade especificada; Neste exemplo estamos usando o evento **"before_create"**, que acontece antes de criar o jogador no banco de dados.
- **"script"**: Código JAVA que será executado quando a trigger for acionada. Todo script começa com a declaração do método void trigger(event, entity, player, database){};

EVENTOS E ENTIDADES

Eventos determinam quando uma trigger será executada. Por exemplo **"before_create"**, é um evento que acontece antes de criar algo. Os eventos são expressões compostas de um prefixo e um sufixo. O prefixo indica o momento do evento (before ou after). O sufixo indica a operação (create, update, delete, win). Já as entidades indicam o que está sendo afetado pelo evento. Por exemplo, player, team, challenge. Veja abaixo alguns exemplos de combinação de eventos e entidades, e quando cada uma destas é acionada:

- **"event": "before_create", "entity": "player"** = Executado logo antes do novo jogador ser cadastrado no banco de dados; O objeto manipulado neste caso é um **Player**;
- **"event": "after_create", "entity": "player"** = Executado logo após o novo jogador ser cadastrado no banco de dados; O objeto manipulado neste caso é um **Player**;
- **"event": "before_update", "entity": "player"** = Executado quando um jogador existente estiver sendo modificado. Logo antes da alteração ser registrada no banco de dados;
- **"event": "after_delete", "entity": "player"** = Executado logo após um jogador ser removido da gamificação; O objeto manipulado neste caso é um **Player**;
- **"event": "after_create", "entity": "challenge"** = Executado logo após o administrador da gamificação criar um novo desafio; O objeto manipulado neste caso é um **Challenge**;
- **"event": "after_win", "entity": "challenge"** = Executado logo após um jogador conquistar um desafio; O objeto manipulado neste caso é um **Achievement**;
- **"event": "before_create", "entity": "action"** = Executado logo antes do administrador da gamificação criar uma nova action; O objeto manipulado neste caso é um **Action**;

- "event": "before_win", "entity": "action" = Executado logo antes de um jogador realizar uma ação; O objeto manipulado neste caso é um **ActionLog**;
- "event": "after_win", "entity": "level" = Executado logo após um jogador subir de nível; O objeto manipulado neste caso é um **Achievement**;
- "event": "after_win", "entity": "catalog_item" = Executado logo após um jogador comprar um item da loja virtual; O objeto manipulado neste caso é um **Achievement**;
- "event": "after_win", "entity": "lottery" = Executado logo após um jogador ser sorteado na loteria; O objeto manipulado neste caso é um **Achievement**;
- "event": "after_win", "entity": "mystery_box" = Executado logo após um jogador ser ganhar um prêmio surpresa; O objeto manipulado neste caso é um **Achievement**;
- "event": "after_win", "entity": "competition" = Executado logo após um jogador vencer uma competição; O objeto manipulado neste caso é um **Achievement**;
- "event": "after_create", "entity": "question_log" = Executado logo após um jogador responder uma pergunta; O objeto manipulado neste caso é um **QuestionLog**;
- "event": "before_create", "entity": "achievement" = Executado logo antes de registrar uma recompensa gerada por um desafio; Por exemplo quando está sendo registrado os pontos gerados pelo desafio; O objeto manipulado neste caso é um **Achievement**;
- "event": "before_create", "entity": "car__c" = Executado logo antes do administrador da gamificação criar um objeto customizado car; O objeto neste caso é um **HashMap** com qualquer estrutura que o administrador esteja usando na collection car__c;

SCRIPT DA TRIGGER

Uma vez que a trigger foi acionada, a plataforma irá executar o método trigger que foi declarado dentro do script JAVA. Veja abaixo um exemplo do código JAVA de uma Trigger para colocar o campo name do jogador em letras maiúsculas antes de criar o jogador no banco:

```
/* "event": "before_create", "entity": "player" */
void trigger(event, entity, player, database) {
    entity.name = entity.name.toUpperCase();
}
```

Os parâmetros informados na assinatura do método trigger são:

- "event" : String com evento que foi acionado, Neste exemplo é "before_create";
- "entity": Objeto que está sendo manipulado; Neste exemplo é do tipo Player; Alguns tipos de objetos JAVA recebidos são Player, Challenge, Action, ActionLog, Achievement;
- "player" : String com o id do jogador que deu origem ao evento;
- "database" : Objeto utilitário para acessar o banco de dados da gamificação;

TIPOS DE OBJETOS

Um dos parâmetros do método trigger, é o entity. Este parâmetro é um objeto que pode ser de vários tipos diferentes. Alguns dos tipos de objetos que recebemos são Player, Challenge, Action, ActionLog, e Achievement. Mas existem vários outros tipos que você poderá manipular

na sua trigger. Veja abaixo a estrutura dos principais objetos que serão recebidos no parâmetro entity dentro das triggers:

- **Player** : {"_id": "john", "name": "John Travolta", "email": "john@funifier.com", "image": {"small": {"url": "https://a.com/a.jpg"}, "medium": {"url": "https://a.com/a.jpg"}, "original": {"url": "https://a.com/a.jpg"}}, "teams": ["sales"], "extra": {"country": "USA", "department": "IT"}, "created": {"\$date": "2023-07-05T20:57:25.776Z"}, "updated": {"\$date": "2023-07-05T20:57:25.777Z"}}
- **Challenge** : {"challenge": "Watch Video", "active": true, "id": "DTo8dS3", "description": "Complete this challenge by watching a video", "rules": [{"actionId": "watch_video", "operator": 5, "total": 0}], "points": [{"total": 10.0, "category": "xp", "operation": 0}]}
- **Action** : {"id": "sell", "action": "Sell", "attributes": [{"name": "product", "type": "String"}, {"name": "price", "type": "Number"}], "active": true}
- **ActionLog** : {"id": "64a5d92", "actionId": "sell", "userId": "john", "time": {"\$date": "2023-07-05T20:57:33.303Z"}, "attributes": {"product": "book", "price": 120}}
- **Achievement** : {"_id": "64a5d2", "player": "john", "total": 25.0, "type": 0, "item": "xp", "time": {"\$date": "2023-07-05T20:57:33.303Z"}}

EXEMPLOS DE CÓDIGO JAVA DE TRIGGER

Podemos utilizar as triggers para por exemplo: Enviar email de boas vindas para o jogador quando ele entrar na gamificação; Publicar uma mensagem de parabenização em uma rede social quando o jogador ganhar uma medalha, etc. Perceba que este recurso deve ser usado com moderação, apenas nos casos onde as configurações padrão de técnicas de jogos do Funifier não forem suficientes para alcançar o resultado esperado pelo negócio. Por exemplo, na configuração de um desafio é possível definir que, ao completar um desafio o jogador ganha 10 pontos. Neste caso não é necessário a utilização de uma trigger para dar os 10 pontos. Porém, caso a pontuação não seja tão direta assim, envolvendo uma regra especial que não pode ser traduzida na configuração padrão do desafio usamos a trigger. Por exemplo, suponha que a pontuação para aquele desafio envolve uma fórmula onde consideramos quantos dias de empresa o jogador tem multiplicado pelo peso do departamento em que o jogador está lotado dentro da empresa. Então esta pontuação será diferente de jogador para jogador, mesmo que eles estejam completando o mesmo desafio. Este sim é um exemplo de situação onde a trigger é necessária.

EXEMPLO 1: TRIGGER PARA DUPLICAR PONTOS DE JOGADORES DA ÁREA DE TI

Neste exemplo, vamos criar uma trigger para dobrar a quantidade de pontos ganhos pelos jogadores do departamento de TI quando eles completarem um desafio. Neste código o parâmetro entity é um objeto do tipo Achievement. Este objeto tem um campo type que indica o tipo de conquista. Neste código vamos: Verificar se o achievement é do tipo ponto; Usar o manager para consultar quem é o jogador que está recebendo este ponto; Ver se no campo extra do jogador existe o departamento de TI; E multiplicar o total de pontos por 2. Veja abaixo o código JAVA da trigger:

```
/* "event": "before_create", "entity": "achievement" */
```



```

void trigger(event, entity, player, database) {
    if(entity.type == Achievement.TYPE_POINT) {
        Player currentPlayer = manager.getPlayerManager().findByld(player);
        if("IT".equals(currentPlayer.extra.department)) { entity.total = entity.total * 2; }
    }
}

```

EXEMPLO 2: TRIGGER PARA DAR PONTO PARA TODOS OS AMIGOS DO JOGADOR

Neste exemplo, vamos criar uma trigger para dar 1 ponto para todos os amigos de um jogador, quando este jogador ganhar 100 pontos ou mais em um desafio. Neste código o parâmetro entity é um objeto do tipo Achievement. Este objeto tem um campo type que indica o tipo de conquista. Neste código vamos: Verificar se o achievement é do tipo ponto e se o total de pontos é maior ou igual a 100; Usar o manager para consultar quem é o jogador; Registrar um Achievement de 1 ponto para cada amigo do jogador; Veja abaixo o código JAVA da trigger:

```

/* "event": "before_create", "entity": "achievement" */
void trigger(event, entity, player, database){
    if(entity.type == Achievement.TYPE_POINT && entity.total >= 100) {
        Player p = manager.getPlayerManager().findByld(player);
        for(String friend : p.friends) {
            Achievement a = new Achievement();
            a.player = friend;
            a.total = 1;
            a.type = 0;
            a.item = entity.item;
            a.time = new Date();
            a.id = Guid.newShortGuid();
            manager.getAchievementManager().addAchievement(a);
        }
    }
}

```

EXEMPLO 3: TRIGGER PARA ENVIAR EMAIL DE BOAS VINDAS PARA JOGADOR

Neste exemplo, vamos criar uma trigger para enviar um email de boas vindas ao jogador assim que ele for criado no banco de dados. Neste código o parâmetro entity é um objeto do tipo Player. Este objeto tem os campos name e email que vamos usar no envio do email. Neste código vamos: Consultar quantos jogadores já estão cadastrados na gamificação; Montar e enviar o email usando a biblioteca (org.simplejavamail) que está disponível no contexto da trigger; Veja abaixo o código JAVA da trigger:

```

/* "event": "after_create", "entity": "player" */
void trigger(event, entity, player, database){
    long total = manager.getPlayerManager().findTotal();
    Email email = EmailBuilder
        .startingBlank()
        .from("Company", "your@company.com")
        .to(entity.getName(), entity.email)
        .withSubject("Welcome!")

```

```

        .withPlainText("Welcome " + entity.name + ", you are the member number " + total)
        .buildEmail();
    MailerBuilder.withSMTPServer("host", 587, "login", "password")
        .buildMailer()
        .sendMail(email);
}

```

EXEMPLO 4: TRIGGER PARA FAZER REQUISIÇÕES HTTP USANDO UNIREST

Neste exemplo, vamos criar uma trigger para enviar os dados da compra feita pelo jogador para o Zapier. Para isso vamos precisar fazer uma requisição HTTP na api do Zapier. Neste código o parâmetro entity é um objeto do tipo Achievement. Este objeto tem o login do jogador e ID do item que foi comprado; Neste código vamos: Consultar os dados completos do jogador que fez a compra; Consultar os dados completos do item que foi comprado; Colocar estas duas informações em um novo objeto; Transformar este objeto em JSON; e enviar o JSON no corpo da requisição. A requisição HTTP será feita usando a biblioteca ([com.mashape.unirest](https://github.com/mashape/unirest)) que está disponível no contexto da trigger; Veja abaixo o código JAVA da trigger:

```

/* "event": "after_win", "entity": "catalog_item" */
void trigger(event, entity, player, database){
    Player buyer = manager.getPlayerManager().findById(entity.player);
    CatalogItem item = manager.getCatalogManager().findItemById(entity.item);
    HashMap zap = new HashMap();
    zap.put("purchase", entity);
    zap.put("buyer", buyer);
    zap.put("item", item);
    HttpResponse<String> response =
Unirest.post("https://hooks.zapier.com/hooks/catch/80/b6/")
        .header("Content-Type", "application/json")
        .body(JsonUtil.toJson(zap))
        .asString();
}

```

EXEMPLO 5: TRIGGER PARA CRIAR CÓDIGO PARA CONVIDAR AMIGOS

Neste exemplo, vamos criar uma trigger para criar um código e inserir no perfil do jogador antes do jogador ser criado; O jogador poderá compartilhar este código posteriormente com seus amigos; Neste código o parâmetro entity é um objeto do tipo Player. Este objeto tem um campo extra onde podemos incluir informações adicionais como este código para convidar amigos. Veja abaixo o código JAVA da trigger:

```

/* "event": "before_create", "entity": "player" */
void trigger(event, entity, player, database){
    String code = Guid.shortTimeMillis();
    entity.extra.put("code", code);
}

```

EXEMPLO 6: TRIGGER PARA CONSULTAR OBJETOS NO BANCO DE DADOS

Neste exemplo, vamos criar uma trigger para pesquisar no banco de dados o preço de um produto que o jogador está vendendo e incluir no log de ação que está sendo criado; Usaremos a biblioteca do MongoDB para consultar o objeto customizado. Neste código o parâmetro entity é um objeto do tipo ActionLog. Vamos incluir o preço nos atributos da venda. Veja abaixo o código JAVA da trigger:

```
/* "before_win", "entity": "action" */
void trigger(event, entity, player, database){
    String id = entity.attributes.product;
    Object product = database.getCollection("product__c").findOne('{_id:#}', id).as(Object.class);
    entity.attributes.put("price", product.price);
}
```

EXEMPLO 7: TRIGGER PARA EXECUTAR COMANDOS AGGREGATE NO DB

Neste exemplo, vamos criar uma trigger para descobrir o total de pontos do jogador com mais pontos na gamificação. E vamos incluir esta informação nos atributos do log de ação que está sendo criado. Para isso vamos executar um comando aggregate usando a biblioteca (**org.jongo**) que está disponível no contexto da trigger. Neste código o parâmetro entity é um objeto do tipo ActionLog. Veja abaixo o código JAVA da trigger:

```
/* "before_win", "entity": "action" */
void trigger(event, entity, player, database){
    org.jongo.MongoCollection collection = database.getCollection("achievement");
    List<Object> highest = Arrays.asList(
        collection.aggregate({'$match': {'type': 0, 'item': 'xp'}})
        .and({'$group': {'_id': '$player', 'total': {'$sum': '$total'}}})
        .and({'$sort': {'total': -1}})
        .and({'$limit': 1})
        .as(Object.class));
    if(highest != null && highest.size() > 0) {
        Object highestPointsPlayer = highest.get(0);
        entity.attributes.put("highest", highestPointsPlayer.total);
    }
}
```

MANAGERS

Dentro de uma trigger é possível acessar o objeto "manager". Este objeto dá acesso aos diversos managers disponíveis dentro da FUNIFIER ENGINE. Estes managers permitem acessar poderosas funcionalidades dentro da plataforma. Já utilizamos alguns managers até agora, como por exemplo: **PlayerManager** para localizar um jogador pelo seu id; **CatalogManager** para localizar um item em um catálogo; Veja abaixo alguns exemplos de managers disponíveis dentro da FUNIFIER ENGINE e suas principais operações:

PlayerManager: Forma de acesso **manager.getPlayerManager()**; Permite gerenciar jogadores; Principais métodos:

- Player findById(String id);

- void insert(Player player);
- void delete(String id);

ActionManager : Forma de acesso `manager.getActionManager()`; Permite gerenciar ações e registrar action logs; Principais métodos:

- Action findActionById(String id);
- void addAction(Action action);
- void deleteAction(String id);
- void track(ActionLog log);
- List<Achievement> trackSynchronous(ActionLog log);

CatalogManager : Forma de acesso `manager.getCatalogManager()`; Permite gerenciar e comprar itens de um catálogo; Principais métodos:

- Catalog findById(String id);
- void add(Catalog catalog);
- void delete(String id);
- CatalogItem findItemById(String id);
- void addItem(CatalogItem item);
- void deleteItem(String id);
- Map<String, Object> purchase(Achievement purchase, boolean async);
- void undoPurchase(String id);

LotteryManager : Forma de acesso `manager.getLotteryManager()`; Permite gerenciar, rodar, e criar tickets para participar de sorteios; Principais métodos:

- Lottery find(String id);
- void insert(Lottery lottery);
- void delete(String id);
- void insertTicket(LotteryTicket ticket);
- Iterable<LotteryTicket> execute(String id);
- void undoExecute(String id);

