UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE DO CAMPUS ARARANGUÁ
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Raul Mendes Rosá

Desenvolvimento de uma plataforma de gamificação genérica utilizando arquitetura limpa e princípios S.O.L.I.D

Araranguá
2021

Raul Mendes Rosá

Desenvolvimento de uma plataforma de gamificação genérica utilizando arquitetura limpa e princípios S.O.L.I.D

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia de Computação do Centro de Ciências, Tecnologias e Saúde do Campus Araranguá da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia de Computação.
Orientador: Prof. Fábio Rodrigues de la Rocha, Dr.
Coorientadora: Prof. Eliane Pozzebon, Dra.

Araranguá
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Rosá, Raul Mendes
Desenvolvimento de uma plataforma de gamificação genérica utilizandoarquitetura limpa e princípios S.O.L.I.D / Raul Mendes Rosá ; orientador, Fábio Rodrigues De la Rocha, coorientadora, Eliane Pozzebon, 2021.
81 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá, Graduação em Engenharia de Computação, Araranguá, 2021.

Inclui referências.

1. Engenharia de Computação. 2. Gamificação. 3. Desenvolvimento de software. 4. Arquitetura limpa. 5. Princípios de design. I. De la Rocha, Fábio Rodrigues. II. Pozzebon, Eliane. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Computação. IV. Título.

Raul Mendes Rosá

Desenvolvimento de uma plataforma de gamificação genérica utilizando arquitetura limpa e princípios S.O.L.I.D

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia de Computação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Computação.

Araranguá, 28 de setembro de 2021.

Documento assinado digitalmente
Fabricio de Oliveira Ourique
Data: 04/10/2021 16:12:38-0300
CPF: 916.167.860-00
Verifique as assinaturas em https://v.ufsc.br

Prof. Fabrício de Oliveira Ourique, Dr.
Coordenador do Curso

Banca Examinadora:

Documento assinado digitalmente
Fabio Rodrigues de La Rocha
Data: 04/10/2021 16:04:07-0300
CPF: 963.007.520-20
Verifique as assinaturas em https://v.ufsc.br

Prof. Fábio Rodrigues de la Rocha, Dr.
Orientador

Documento assinado digitalmente
Eliane Pozzebon
Data: 05/10/2021 08:46:55-0300
CPF: 779.213.139-49
Verifique as assinaturas em https://v.ufsc.br

Profa. Eliane Pozzebon, Dra.
Coorientadora
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Jim Lau
Data: 04/10/2021 16:17:26-0300
CPF: 613.464.702-00
Verifique as assinaturas em https://v.ufsc.br

Prof. Jim Lau, Dr.
Avaliador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Luciana Bolan Frigo
Data: 04/10/2021 17:25:30-0300
CPF: 910.389.799-00
Verifique as assinaturas em https://v.ufsc.br

Profa. Luciana Bolan Frigo, Dra.
Avaliadora
Universidade Federal de Santa Catarina

Prof. Martin Augusto Gagliotti Vigil, Dr.
Avaliador Suplente
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Agradeço aos meus pais, pela educação e a oportunidade de estudar nesta grande universidade. Agradeço ao Movimento Empresa Júnior pelas oportunidades de desenvolvimento interpessoal e profissional que me foram dadas, em especial a EJEC por ser o veículo que me levou por todo este trajeto e que despertou o interesse pelo tema. Também agradeço a meus amigos e colegas que compartilharam comigo suas experiências e saberes.

RESUMO

A gamificação tem se tornado popular, com diversos trabalhos implementando processos gamificados, tanto analógicos quanto digitais. Para os processos digitais, um desafio é construir *software* que supra as necessidades que um sistema gamificado possui. Neste trabalho, foi desenvolvida uma plataforma de gamificação capaz de suprir requisitos de diversos ambientes interessados em criar processos gamificados, a fim de demonstrar as etapas de desenvolvimento de uma plataforma deste tipo, para que interessados em gamificar seus processos possam se guiar pelos conceitos abordados ou até mesmo utilizar a plataforma aqui construída. Tal plataforma foi composta de um sistema *web*, um servidor e um aplicativo móvel, ambos desenvolvidos utilizando Arquitetura Limpa, *Design* Orientado ao Domínio e princípios S.O.L.I.D, práticas conhecidas no mercado por possibilitar a criação de aplicações escaláveis e de fácil manutenibilidade. O trabalho também detalha práticas de desenvolvimento, explicando escolhas de linguagens de programação, padrões de código, boas práticas de *design* e bibliotecas utilizadas. Com o fim do desenvolvimento, a plataforma foi distribuída em ambiente de testes para a validação com usuários. Estes usuários exploraram as suas funcionalidades mais importantes, e então responderam um questionário avaliando-a de acordo com a usabilidade e interface de utilização. A aplicação de boas práticas e metodologias foi fundamental na resolução de problemas durante a etapa de testes, resultando em uma boa avaliação por parte dos usuários.

Palavras-chave: Gamificação. Desenvolvimento de software. Arquitetura limpa. Princípios de design.

ABSTRACT

Gamification has become popular, with many works implementing gamified processes, both analogical and digital. For the digital processes, one challenge is to build software that fulfills the needs a gamified system has. In this work, a gamification platform has been developed to supply the requirements of many environments interested in creating gamified processes, in order to demonstrate the development stages of a plataform of this kind, so that people interested in gamifying their processes may guide themselves by the concepts addressed here or even use the platform itself. The platform contained a web system, a web server and a mobile application, both developed using Clean Architecture, Domain Driven Design and S.O.L.I.D. principles, practices well known in the market for enabling the creation of scalable and easy maintainable applications. The work also details the development methodologies, explaining choices such as programming languages, code standards, good design practices and libraries used. By the end of development, the platform was deployed on a test environment for user validation. These users explored its most important features, and then answered a questionnaire rating it according to the usability and user interface. Applying best practices and methodologies was fundamental when solving problems on the tests stage, resulting on a good rating by the users.

Keywords: Gamification. Software development. Clean architecture. Design principles.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	*Asynchronous Javascript and XML*
API	*Application Program Interface*
DDD	*Domain-Driven Design*
HTTP	*Hypertext Transfer Protocol*
JWT	*JSON Web Token*
REST	*Representational State Transfer*
SDK	*Software Development Kit*
SPA	*Single Page Application*
UML	*Unified Modeling Language*
URI	*Universal Resource Identifier*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos a gamificação tem se tornado muito popular no mercado em diversos setores. O processo tem sido aplicado em negócios, companias e plataformas diversas, e mais recentemente acadêmicos e educadores começaram a se interessar pelo tema e suas aplicações (SEABORN; FELS, 2015). O tema tem se tornado imensamente popular, com propostas de revolucionar o comportamento e engajamento em atividades, ganhando mais e mais adeptos assim como críticos e adversários (WOODCOCK; JOHNSON, 2017).

Na academia o interesse também cresceu, principalmente durante os últimos dez anos. Com o interesse aparecem também confusões, principalmente entre os conceitos de gamificação e aprendizado baseado em jogos. Alguns educadores acreditam implementar um enquanto implementam o outro, o que dificulta encontrar assertivamente referências sobre gamificação em si (ALSAWAIER, 2017). Mesmo assim, com o notável crescimento do tema e a quantidade de novos trabalhos publicados, não é difícil encontrar exemplos de pesquisa e estudos envolvendo a aplicação de gamificação em áreas diversas (KUO *et al.*, 2018).

No mercado e nos negócios, a gamificação está presente a muito tempo. Desde os anos 70 já sabe-se de recompensas dadas a colaboradores por diversas funções, e com o passar do tempo mais e mais estímulos têm sido fornecidos para aumentar a produtividade no trabalho. Com o avanço do estilo de gerência por objetivos e a divisão das responsabilidades do trabalho em tarefas, a implementação de mecânicas de jogos que de acordo com Kapp (2013) constituem um sistema gamificado se tornou fácil e natural. Entretanto, não foi até os meados dos anos 2000, quando o termo se tornou famoso e seu conceito conhecido, que os pequenos elementos de gamificação começaram a ser reconhecidos e implementados intencionalmente para este propósito (KORN; SCHMIDT, 2015).

As vantagens da gamificação incluem maior satisfação dos usuários, maior engajamento e participação, e aumento na produtividade em atividades gamificadas. Entretanto, estas vantagens não são simples de extrair e não acontecem sem efeitos colaterais. A aplicação de elementos ditos pervasivos de gamificação, como placar de líderes e conquistas, sem o estímulo devido dos participantes através de outras fontes pode diminuir o engajamento a longo prazo, ou reduzir a eficiência dos participantes em aspectos não gamificados das atividades, um contraponto que deve ser levado em consideração ao construir seu processo gamificado (SEABORN; FELS, 2015).

1.1 JUSTIFICATIVA

Atualmente existem vários trabalhos que abordam gamificação. Pesquisas pelo termo *gamification* ou *gamif** em plataformas acadêmicas como o *Science Direct* retornam inúmeros resultados, contendo desde trabalhos de revisão bibliográfica, pesquisas sobre a

efetividade do conceito e implementação do mesmo em diferentes casos de uso, especialmente em salas de aula. Muitas implementações se dão de forma analógica, mas alguns trabalhos implementam os sistemas em *software*.

Grande parte destas pesquisas mostram a relevância da gamificação em diversos contextos. Jones, Madden e Wengreen (2014) conseguiram aumentar o consumo de frutas e vegetais de estudantes em uma escola através de uma intervenção gamificada. Hitchens e Tulloch (2018) e Korn e Schmidt (2015) verificaram um aumento no engajamento e motivação de estudantes e profissionais em ambientes gamificados. No mercado muitas plataformas surgem para auxiliar na gamificação de processos, como a Gametize, e outras utilizam de gamificação para engajar os usuários em seu conteúdo. Alguns dos exemplos mais famosos destas últimas são Memrise e Duolingo, ambas plataformas de ensino de idiomas que gamificam seus processos de aprendizado. Estes casos fornecem indícios de que a gamificação pode trazer bons resultados em diversos ecossistemas, sejam educacionais ou profissionais.

Uma questão levantada, entretanto, é que grande parte das pesquisas que tratam de gamificação não detalham as implementações em *software*, quando existentes, apenas mencionando-as superficialmente. Isto cria um vácuo onde interessados em desenvolver suas próprias plataformas para experimentação ou replicação dos estudos precisam percorrer um grande caminho para encontrar referências para tal, possivelmente atrasando o prazo da pesquisa ou alocando mais custos. Para solucionar esta questão, este trabalho se propõe a desenvolver uma ferramenta de *sofware* para gamificação, dando enfoque nas suas etapas de desenvolvimento, fornecendo exemplos de metodologias, tecnologias e práticas que podem ser usadas para construir um *software* moderno e que atenda aos requisitos básicos de uma plataforma de gamificação.

Para tal, planeja-se usar práticas de arquitetura e organização de código utilizadas no mercado de tecnologia para a construção de sistemas robustos, escaláveis e de fácil manutenção. Dentre estas práticas, estão a Arquitetura Limpa, os princípios S.O.L.I.D (MARTIN, 2017) e o *Design* Orientado a Domínio (EVANS, 2003). Espera-se que, ao aplicar estes conceitos de arquitetura e *design* de sistemas, e ao utilizar padrões de *design* de código eficientes apreciados pelos desenvolvedores (SHVETS, 2020), este trabalho possa ser utilizado como referência para que estudantes e pesquisadores saibam como desenvolver seus próprios pedaços de *software* de forma robusta e escalável.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Este trabalho tem como objetivo desenvolver o protótipo de uma plataforma de gamificação utilizando boas práticas e metodologias do mercado, para que pessoas e instituições interessadas em gamificar seus processos ou experimentar com o conceito

de gamificação possam agilizar a etapa de desenvolvimento do *software* ou utilizar a plataforma aqui construída.

1.2.2 Objetivos Específicos

Para atingir o objetivo geral, o trabalho conta com os seguintes objetivos específicos:

- Mapear os elementos de gamificação básicos que constituirão a plataforma;
- Desenvolver uma *Application Program Interface* (API) para gerenciar as permissões e os dados dos usuários da plataforma;
- Desenvolver um sistema *web* para os gestores dos jogos configurarem seus sistemas gamificado de forma personalizada;
- Desenvolver um aplicativo móvel para que os jogadores possam participar e engajar no sistema através do seu próprio *smartphone* ou *tablet*;
- Publicar o sistema e o aplicativo em um ambiente de testes público para validação com usuários;
- Escolher alguns usuários para testar a plataforma, aplicando seus sistemas gamificados com auxilio do autor, e coletar seus *feedbacks* e opiniões.

1.3 METODOLOGIA

Este trabalho tem caráter experimental, desenvolvido na Universidade Federal de Santa Catarina - UFSC. A construção da plataforma foi realizada seguindo os seguintes passos:

- Estudo das características de um sistema gamificado e como reproduzí-lo;
- Análise de outras plataformas de gamificação;
- Levantamento de necessidades de possíveis usuários do sistema para modelagem dos requisitos;
- Levantamento dos requisitos e modelagem do domínio da aplicação;
- Escolha das tecnologias para a implementação, visando agilidade de desenvolvimento e facilidade de publicação;
- Desenvolvimento da plataforma até um mínimo produto viável;
- Validação da plataforma com usuários teste;
- Ajustes de funcionalidades e requisitos conforme solicitado e necessitado pelos usuários teste.

1.4 ORGANIZAÇÃO DO TRABALHO

Esta monografia está organizada em 5 capítulos e representa o trabalho de conclusão de curso desenvolvido para a Graduação de Engenharia da Computação na Universidade Federal de Santa Catarina.

- **Capítulo 2:** Apresenta a fundamentação teórica, explicando conceitos que serão utilizados durante o desenvolvimento do trabalho. Seu papel é ajudar o leitor a compreender a origem de algumas práticas e termos utilizados no decorrer do desenvolvimento.
- **Capítulo 3:** Apresenta as escolhas de tecnologias para o desenvolvimento da plataforma, a proposta da plataforma em si e os passos dados para construir um mínimo produto viável.
- **Capítulo 4:** Apresenta os métodos utilizados para validar a plataforma e o público alvo escolhido para os testes.
- **Capítulo 5:** Apresenta as conclusões do trabalho e ideias de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRIA

Neste capítulo são abordados alguns conceitos importantes para a compreensão do trabalho e seu desenvolvimento.

2.1 GAMIFICAÇÃO

A gamificação, de forma sucinta, pode ser definida como o uso de mecânicas, estética e pensamento baseados em jogos para engajar pessoas, promover o pensamento e resolver problemas (KAPP, 2013). Ainda não há um consenso quanto a definição geral, com alguns pesquisadores definindo a gamificação como o uso de elementos de jogos, mecânicas, recursos, *design* ou estrutura em ambientes ou contextos não-lúdicos (ALSAWAIER, 2017), enquanto Kapp (2013) define este emprego como um subtipo da gamificação chamado de **gamificação estrutural**.

De acordo com Deterding *et al.* (2011), a palavra *gamification* apareceu pela primeira vez em 2008, e a partir de então foi popularizada por diversos autores e palestrantes, até se espalhar pelo mundo (SANCHEZ; YOUNG; JOUNEAU-SION, 2016). O conceito se tornou muito popular e tem sido objeto de muitos estudos nos últimos anos, com buscas pelo termo em plataformas como o *Google Scholar* retornando mais de 25.000 resultados (WOODCOCK; JOHNSON, 2017).

A metodologia tem sido amplamente utilizada em empresas assim como em alguns ambientes escolares, e pode tomar várias formas. Ela surge através do uso de narrativas para mudar o contexto de uma atividade corriqueira, através de incentivos usando medalhas e sistemas de recompensa, e através da criação de uma forma de competição social (HANUS; FOX, 2015). O emprego de forma correta da gamificação pode trazer vários benefícios para um grupo, como incentivar o aprendizado, motivar ação, influenciar o comportamento, estimular inovação, construir habilidades e fomentar a aquisição de conhecimento (KAPP, 2013).

Kapp (2013) compila em seu livro uma série de características para jogos e simulações que podem ser usados para criar um bom jogo. Estas características resumem os elementos base que um jogo pode ter:

- **Construtos:** Construtos são objetos construídos no jogo que não existem no mundo real. Eles contemplam os elementos que o jogo possui, e são usados para constituir as barreiras entre o que é e o que não é possível no jogo, dar melhores informações aos jogadores e interagir com eles. Exemplos de construtos são habilidades especiais, níveis, patentes, mecânicas e regras.

- **Desafios:** Desafios são um componente chave em jogos, simulações e gamificações. Os desafios são o que mantêm os jogadores engajados, fornecendo a eles novas experiências e estímulos para continuar jogando. Os desafios devem ser construídos

com cuidado, para se adequarem a curva de experiência do jogador, sendo mais instigantes conforme o mesmo evolui dentro do jogo.

- ***Feedback*:** Um dos elementos mais importantes do jogo, o *feedback* instrui o jogador quanto a quais comportamentos são bons ou ruins dentro do jogo. Através do *feedback*, o jogador pode saber se deve continuar em um curso de ação ou mudar de atitude. Bons *feedbacks* devem levar em consideração o momento correto para serem dados, como eles devem ser entregues e qual tom devem apresentar ao jogador. Exemplos de *feedback* são animações, mudança de elementos em tela, ou adquirência de construtos do jogo.

- **História:** Uma prática comum é usar de uma história para complementar um jogo, simulação ou gamificação. A história serve para dar sentido às mecânicas e outros elementos do jogo, e para criar um modelo mental para os jogadores, onde aqueles elementos pertencem a um mesmo contexto.

2.2 ARQUITETURA DE SOFTWARE

Nesta seção são abordados os conceitos dos modelos de arquitetura de *software* utilizados no desenvolvimento da aplicação. Estes modelos definem como o código é organizado, como diferentes trechos de código compõem-se em sistemas ou camadas, e como estas comunicam-se entre si para entregar um produto completo.

2.2.1 Arquitetura Cliente/Servidor

Este é um modelo de arquitetura de sistemas onde os atores desempenham um ou ambos os papeis abaixo:

- **Cliente:** É o ator que providencia serviços de apresentação para o usuário, tal como os serviços de base de dados, conectividade, acesso e interfaces que o negócio necessita;

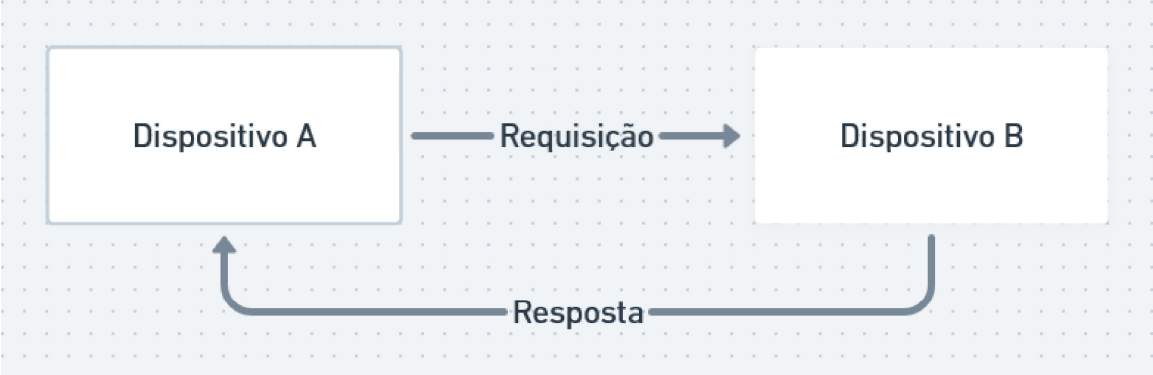
- **Servidor:** É composto por um ou mais processadores multiusuários com uma memória compartilhada que providencia serviços de base de dados, conectividade, processamento e interfaces que o negócio necessita.

Este modelo é relativamente genérico, onde sua maior característica é a interação entre os atores: O cliente faz uma requisição ao servidor; este, por sua vez, processa a requisição e retorna ao cliente uma resposta. Nota-se que, em alguns casos, um servidor pode vir a se tornar cliente numa mesma cadeia de comunicação (SMITH, 1994).

Como exemplo, na Figura 1, o **Dispositivo A** age como cliente, já que faz uma requisição ao **Dispositivo B**, que por sua vez age como servidor ao processar a requisição

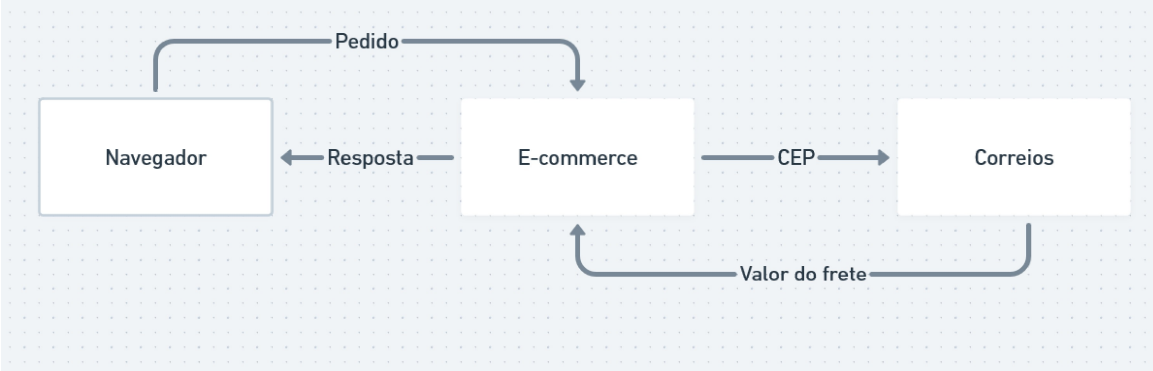
e retornar a resposta para o Dispositivo A. Já no exemplo da Figura 2, um **navegador *web*** faz uma requisição para um serviço de ***e-commerce***, enviando dados para efetuar um pedido. O *e-commerce*, por sua vez, precisa calcular o frete da entrega. Para fazer isso, ele deve realizar uma requisição para o serviço dos **Correios**, enviando o CEP do usuário e recebendo de volta o valor do frete. Por fim, o *e-commerce* retorna a resposta final ao navegador. Na comunicação entre o navegador e o *e-commerce*, o navegador consome os dados do serviço, e portanto age como cliente enquanto o *e-commerce* age como servidor. No segundo passo, o *e-commerce* requisita dados aos Correios, agindo assim como cliente enquanto o serviço dos Correios age como servidor.

Figura 1 – Cliente e Servidor.



Fonte: Elaborado pelo autor (2021).

Figura 2 – Interação entre navegador, *e-commerce* e correios.



Fonte: Elaborado pelo autor (2021).

2.2.2 Arquitetura Limpa

Arquitetura limpa é uma forma de chamar um estilo de arquitetura de *software* que foca na separação de responsabilidades, um objetivo atingido ao separar o código em

camadas. É importante que o sistema seja divido pelo menos em uma camada para as regras de negócio, uma camada do usuário e uma camada com as interfaces do sistema (MARTIN, 2017).

De acordo com Martin (2017), um sistema construído com arquitetura limpa tem como objetivo ser:

- **Independente de *Frameworks***: A arquitetura não depende da existência das bibliotecas de um *software* cheio de funcionalidades. Ela permite que os *frameworks* sejam usados como ferramentas, em vez de fazer o sistema depender de suas limitações pré-estabelecidas.
- **Testável**: As regras de negócio podem ser testadas independente da interface do usuário, banco de dados, servidor *web* ou quaisquer dependências externas.
- **Independente da UI**: A interface do usuário pode ser trocada facilmente, sem alterar o funcionamento do sistema. Por exemplo, uma interface *web* pode ser trocada por uma interface de linha de comando, sem trocar as regras de negócio.
- **Independente de Banco de Dados**: É possível trocar o banco de dados de *Oracle* para *SQL*, *MongoDB*, ou até mesmo um sistema de arquivos. As regras de negócio não estão atreladas a base de dados.
- **Independente de qualquer agência externa**: Na realidade, as regras de negócio não sabem nada sobre as interfaces do mundo exterior.

A Figura 3 é uma representação visual de como uma arquitetura limpa deve ser construída.

2.3 DESIGN DE SISTEMA

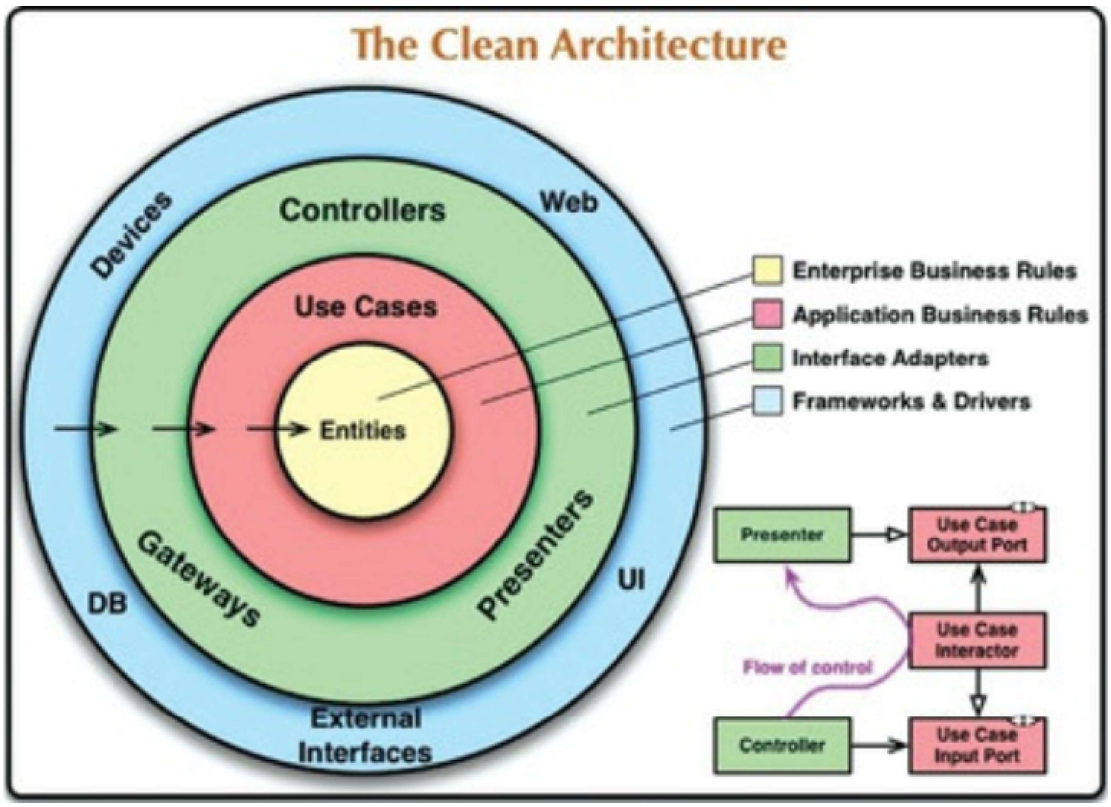
Nesta seção são abordados os conceitos de *design* de *software* utilizados no desenvolvimento da aplicação. Estes conceitos colaboram para a modelagem dos módulos e componentes do sistema, assim como a sua estrutura de arquivos e escrita de código.

2.3.1 Design Orientado ao Domínio

O *Design* Orientado ao Domínio, mais conhecido como *Domain-Driven Design* (DDD), é um conceito de desenvolvimento de *software* que diz que a linguagem e estrutura do sistema deve corresponder ao modelo de negócio que ele representa.

Um *software* desenhado desta forma tem como grande objetivo a comunicação entre as partes interessadas. Desenvolvedores, analistas e outros especialistas são capazes de construir os comportamentos e as regras de negócio juntos, através de um idioma ubíquo e universal definido pelos mesmos. Modelando o sistema utilizando DDD, os desenvolvedores

Figura 3 – Arquitetura Limpa.



Fonte: Martin (2017)

e outros *stakeholders* conseguem maior clareza sobre as regras de negócio, entidades e comportamentos esperados (EVANS, 2003).

2.3.2 Princípios S.O.L.I.D.

Os princípios SOLID são um conjunto de diretrizes que dizem como arranjar as funções e estruturas de dados em classes, e como estas classes devem estar interconectadas. Nota-se que neste caso o conceito de classe refere-se a um conjunto acoplado de dados e funções, não sendo unicamente aplicado à programação orientada a objetos (MARTIN, 2017).

De acordo com Martin (2017), a sigla SOLID se refere aos seus cinco princípios:

- **Princípio de Responsabilidade Única**: Do inglês *Single Responsibility Principle*, este princípio sugere que os módulos e componentes do sistema devem possuir uma única responsabilidade. Esta responsabilidade pode ser definida como as razões pelas quais o componente deve ser modificado, de forma a quebrar componentes grandes em componentes menores com responsabilidades pequenas;

- **Princípio Aberto-Fechado**: Do inglês *Open-Closed Principle*, este princípio descreve que para um *software* ser fácil de mudar, ele deve ser desenhado de forma a possibilitar que o comportamento destes sistemas possa ser incrementado apenas com a adição de novo código, mas sem a necessidade de alteração de código já existente;
- **Princípio de Substituição de Liskov**: Cunhado por Bárbara Liskov em 1988, este princípio define que um sistema deve ser composto de partes substituíveis, e estas partes devem respeitar contratos pré-estabelecidos para permitir esta substituição;
- **Princípio de Segregação de Interface**: Em inglês, *Interface Segregation Principle*, é o princípio que diz que um sistema deve evitar depender de coisas que ele não usa. Na prática, é recomendado que utilize-se interfaces ou outras abordagens para que um sistema dependa apenas de métodos e atributos que ele utilize, independente de quantos destes métodos ou atributos os recursos externos utilizados possuam;
- **Princípio de Inversão de Dependências**: Do inglês *Dependency Inversion Principle*, este princípio dita que o código que implementa as políticas de alto nível de um sistema não devem depender dos detalhes de baixo nível. Pelo contrário, os detalhes devem depender das políticas.

2.4 MODELOS DE CONSTRUÇÃO DE SISTEMA

Esta seção aborda diferentes abordagens de construção de sistema utilizadas no desenvolvimento, que definem o comportamento de um sistema como um todo e a sua forma de se comunicar com os demais sistemas que compõem a plataforma.

2.4.1 REST

Representational State Transfer (REST) é um protocolo para serviços *web*, uma das principais formas de interação entre interfaces deste tipo de sistema. Ela surgiu a partir do protocolo original usado quando a *World Wide Web* foi introduzida. O protocolo possui pelo menos seis premissas de arquitetura:

- **Interface uniforme**, onde todas as interações devem ser realizadas da mesma forma. Tipicamente, isto envolve a utilização de *Hypertext Transfer Protocol* (HTTP), e o acesso é providenciado por meio de uma *Universal Resource Identifier* (URI);
- Utiliza o padrão **cliente-servidor**;
- As interações são ***stateless***, ou seja, o cliente nunca deve presumir que as informações da última requisição foram armazenadas. Por isso, geralmente utilizam-se *tokens* para autenticação, que são enviados com cada requisição;

- Quando possível, recursos devem poder ser armazenados em ***cache***;
- O sistema pode ser construído a partir de múltiplos elementos independentes, que podem ser compilados e executados independentemente. Por exemplo, a camada de negócio pode estar em um servidor *web* enquanto a camada de acesso de dados está executando por meio de um serviço de armazenamento na nuvem;
- De maneira opcional, o servidor pode fornecer ao cliente o código para ser executado, em vez de executar o código ele mesmo. Isto tende a acontecer bastante com Javascript, principalmente em abordagens *Single Page Application* (SPA).

Mesmo não sendo feito unicamente para ser utilizado com HTTP, este é o protocolo mais utilizado, com ampla literatura e exemplos disponíveis (BASS; CLEMENTS; KAZMAN, 1997).

2.4.2 Single Page Applications

SPA são aplicações *web* que carregam uma única página HTML, e então dinamicamente atualizam o conteúdo da página conforme o usuário interage com os elementos presentes, tais como menus e botões. Devido a essa natureza, páginas SPA oferecem uma experiência mais fluida e de aparência nativa, sem a necessidade do recarregamento constante das páginas que ocorre em outros tipos de abordagem.

Estas aplicações podem já conter todo o código a ser executado e elementos a serem mostrados, ou buscar mais dados e códigos através de requisições do tipo *Asynchronous Javascript and XML* (AJAX). Hoje em dia, os *frameworks* mais modernos de desenvolvimento *web* são construídos utilizando SPA, dentre eles *jQuery*, *React*, *AngularJS* e *Vue.js* (SHAHZAD, 2017).

2.5 TRABALHOS RELACIONADOS

Esta seção relaciona o presente trabalho com outros trabalhos que abordam temas similares. Os trabalhos aqui relacionados foram escolhidos por envolverem a aplicação de sistemas gamificados em ambientes práticos, com a avaliação de pessoas reais e implementação em ambientes com múltiplos usuários. Buscou-se trabalhos cujas implementações eram realizadas com o auxílio de *software* específico, mas na ausência de mais exemplos foi relacionado um trabalho com aplicação apenas analógica mas com grande potencial de digitalização.

2.5.1 FIT Game

Preocupados com o consumo de frutas e vegetais entre crianças e adolescentes, os pesquisadores Jones, Madden e Wengreen (2014) resolveram realizar uma intervenção

gamificada em alunos do ensino fundamental para avaliar se o consumo destes alimentos aumentaria com uma nova estratégia de engajamento.

Para tal, eles criaram o *FIT Game*, um jogo analógico desenvolvido para incentivar o consumo de frutas e vegetais. O jogo possuia uma narrativa onde os heróis, chamados de FITs, combatiam e capturavam os VATs (Equipe de Aniquilação da Vegetação, do inglês *Vegetation Annihilation Team*). Os estudantes assumiam o papel de FITs, e precisavam cumprir objetivos para derrotar seus inimigos.

Mecanicamente, o jogo funcionava da seguinte forma: conforme os estudantes consumiam frutas e vegetais, o peso destes era atribuído a uma fórmula que calculava a média de consumo da escola. Uma escola competia contra outras escolas, em um torneio que duraria três rodadas, cada uma com uma duração em dias equivalente ao seu número (por exemplo, a rodada 1 durava um dia, a rodada 2 durava dois dias, assim por diante). Ao fim de cada rodada, uma escola seria vitoriosa se tivesse um consumo médio de frutas e vegetais maior que suas competidoras. As outras escolas entretanto eram fictícias, portanto na prática existia uma média alvo que a escola deveria atingir para ser considerada vitoriosa; os participantes do estudo não sabiam desta informação. Ao término das três rodadas, a escola se qualificaria para ajudar os FITs a capturar os vilões. Além do fator competição, cada grama que a escola excedia ao objetivo era convertida em uma espécie de moeda virtual própria do jogo. No 4º dia, os estudantes votaram em como gastar esta moeda (em itens virtuais ou na direção da narrativa).

Ao término do experimento, os pesquisadores identificaram que o consumo de frutas e vegetais aumentou em 39% e 33% respectivamente durante a aplicação do jogo, mostrando que este jogo é uma alternativa barata de intervenção que as escolas podem implementar para incentivar o consumo destes alimentos.

2.5.2 Gamification em um curso de UML

O trabalho realizado por Jurgelaitis *et al.* (2018) na Universidade de Tecnologia de Kaunas, na Lituânia, teve como objetivo gamificar um curso de *Unified Modeling Language* (UML) para estudantes de graduação. Os pesquisadores utilizaram a plataforma *Moodle* para o processo, com ajuda de alguns *plugins* para automatizar o jogo.

Os componentes de jogos utilizados foram pontos de experiência, moedas, medalhas, placar de líderes e fases. Cada fase do jogo representava uma série de conceitos de UML, como diagramas de classe, atividades e casos de uso. O jogo possuía cinco fases, e os recursos e tarefas das fases superiores eram bloqueados até que o jogador desbloqueasse seu acesso. Cada fase possuía uma série de tarefas, que consistiam em provas e atividades avaliativas sobre o tema daquela fase. Ao completar as tarefas, o jogador recebia recompensas em forma de pontos de experiência, moedas e/ou medalhas, e possivelmente poderia desbloquear uma nova fase. Os jogadores recebiam *feedback* imediato das tarefas, e o seu progresso era medido baseado em qual fase estão e quantos pontos de experiência possuem.

Estas duas informações também compunham o placar de líderes.

O jogo foi aplicado aos estudantes que estavam matriculados na matéria de Análise de Sistemas de *Software* e Ferramentas de *Design*. A participação era voluntária, de forma que o conteúdo da matéria era dado normalmente fora do jogo. O jogo fornecia apenas ferramentas e conhecimentos adicionais, para que os estudantes pudessem conhecer melhor a UML. Como resultado, os pesquisadores conseguirem metrificar um aumento da nota média dos alunos em 0,7 pontos comparado com os alunos do semestre anterior, assim como um aumento na motivação dos mesmos.

2.5.3 O desenho de um sistema gamificado na sala de aula

Os pesquisadores Hitchens e Tulloch (2018) desenvolveram um sistema gamificado para medir se a gamificação poderia de fato aumentar o engajamento e a motivação de estudantes. Eles criaram um sistema baseado em atividades, utilizando uma mecânica de obtenção e gasto de pontos. Os estudantes obtinham os pontos ao realizar uma série de atividades ou através de bom comportamento, e podiam gastar estes pontos em uma série de recompensas, que podiam variar desde cosméticos virtuais até vantagens na sala de aula.

Para auxiliar a pesquisa, foi desenvolvido um *software* que os estudantes poderiam utilizar para monitorar seu progresso. Após o período de pesquisa, os estudantes responderam perguntas para expressar suas opiniões sobre o processo de gamificação. Como resultado, a maior parte dos estudantes achou a iniciativa válida e interessante, e se mostraram mais motivados nos estudos. Entretanto, os pesquisadores ressaltam que houve uma minoria que desaprovou pesadamente, e que esta minoria não deve ser desconsiderada.

3 DESENVOLVIMENTO

Neste capítulo serão abordados os passos do desenvolvimento da plataforma. Primeiramente, será discutido o conceito da plataforma e os elementos de gamificação escolhidos para contemplá-la. Depois, serão discutidas as tecnologias escolhidas para a implementação e a razão de sua escolha. Após isto, será descrita a modelagem do domínio da aplicação, usando os conceitos elaborados por Evans (2003). Então, será abordado o processo de desenvolvimento dos três *softwares* que constituem a plataforma, discorrendo sobre sua estrutura de pastas e arquivos e padrões utilizados. Por fim, há uma demonstração dos fluxos de utilização da plataforma. Os códigos dos sistemas estão disponíveis em repositórios públicos para referência[1].

3.1 GAMETASK: A PLATAFORMA DE GAMIFICAÇÃO

A plataforma de gamificação é chamada de Gametask, com tradução literal do inglês que significa "tarefa de jogo". Seu principal objetivo é ser uma plataforma genérica, permitindo que os usuários criem seus próprios processos gamificados independente do objetivo final, seja para empresas, salas de aula, educação dos filhos e etc. Uma plataforma dessas possibilita abordar diversos elementos de jogos, e deixa em aberto para os usuários a opção de utilizá-los ou não.

Uma parte importante do conceito da plataforma é que o Gametask é não-supervisionado, ou seja, a plataforma não se responsabiliza sobre o funcionamento do jogo, seus processos ou a qualidade do jogo criado. Ela é apenas uma ferramenta que pode ser utilizada para criar processos gamificados, fornecendo um pequeno nível de automação, e portanto os jogos resultantes dependem apenas do planejamento e criatividade de seus utilizadores.

Os usuários da plataforma assumem dois principais papeis: o gestor do jogo e o jogador. O gestor é aquele que cria um jogo, convida pessoas para serem os jogadores e configura todo o processo gamificado. Ele diz como os elementos de gamificação se comportarão no jogo, e arbitra sobre a obteção de pontuações e conquistas. O jogador é o participante de jogo. Ele compete com outros jogadores, completando atividades, obtendo conquistas e pontos de experiência.

Os elementos de gamificação disponíveis na plataforma são inspirados em jogos e outras plataformas de gamificação. Algumas das referências são Duolingo, Tree of Savior, League of Legends e MMORPGs em geral (um gênero de jogos virtuais). Os elementos estão listados de acordo com as definições de elementos básicos de Kapp (2013).

[1] Os repositórios podem ser acessados em `https://github.com/raulrozza/Gametask_API`, `https://github.com/raulrozza/Gametask_Web` e `https://github.com/raulrozza/Gametask_Mobile`.

3.1.1 Constructos

- Atividades: são os objetos que oferecem um objetivo a ser concluído, e garantem aos jogadores pontos de experiência ao serem conquistadas;
- Conquistas: são equivalentes a medalhas, objetos virtuais dados aos jogadores que conseguirem feitos considerados grandiosos no jogo. Cada conquista só pode ser obtida uma única vez;
- Níveis: são representações do progresso do jogador. São objetos obtidos através do acúmulo de pontos de experiência, com objetivo de comparar diferentes jogadores e oferecer competitividade;
- Patentes: são objetos atrelados a certos níveis que agrupam jogadores de determinadas faixas de pontos de experiência. São opcionais no jogo, e têm como objetivo criar um sentimento de grupo e comunidade entre os jogadores da mesma patente, além de estimular competição para atingir patentes maiores;
- Placar de Líderes: são quadros que listam os jogadores de forma decrescente de acordo com a quantidade de pontos de experiência acumulados por eles. Um jogador entra para o placar de líderes assim que obtém algum ponto de experiência, e permanece no placar até que este seja restaurado, momento em que as colocações são zeradas e o placar começa a contabilizar apenas as novas pontuações;
- Pontos de Experiência: são valores numéricos fornecidos aos jogadores pelo cumprimento de atividades. Os pontos de experiência são cumulativos, e denotam o progresso do jogador dentro do jogo. O acúmulo destes pontos garate ao jogador novos níveis;
- Títulos: são nomes virtuais dados aos jogadores que obtiverem determinadas conquistas. Os títulos possuem valor cosmético, onde um jogador pode escolher um de seus títulos para ser mostrado ao lado de seu nome no jogo. Sua função é dar aos jogadores que cumpriram certo objetivos a satisfação de serem reconhecidos imediatamente pelo título;

3.1.2 Desafios

- Colocações: o desafio de se manter no topo do placar de líderes pode se mostrar uma fonte de motivação para jogadores mais competitivos;
- Complecionismo: jogadores complecionistas encontrarão no Gametask o desafio de obter todas as conquistas e títulos do jogo, assim como atingir os maiores níveis possíveis;

- Objetivos: a principal fonte de desafios no Gametask são os objetivos fornecidos pelo gestor do jogo. Estes objetivos mostram aos jogadores quais tarefas eles precisam executar, e devem ser desenhados de forma a fornecer desafios pequenos, médios e grandiosos, para melhor estimular a participação. Para criar objetivos, o gestor do jogo pode criar tanto atividades quanto conquistas;

3.1.3 Feedback

- Barra de progresso: uma barra de progresso no perfil do jogador mostra quantos pontos de experiência faltam para atingir o próximo nível;
- Placar de líderes: O placar de líderes mostra em tempo real como o jogador está se saindo em comparação aos seus colegas no jogo;

3.1.4 História

O Gametask não oferece uma ferramenta unificada de história, mas sim elementos distintos que podem ser usados pelo gestor do jogo para compor uma narrativa. Estes elementos são:

- Alteração das cores da plataforma quando em jogo, ofercendo maior imersão e identidade ao jogo;
- Atribuição de imagens ao jogo e às conquistas, dando a possibilidade de montar uma identidade visual temática do jogo;
- Fornecimento de patentes com nomes e cores personalizadas, que podem constituir o tema do jogo;
- Nomeação de atividades e conquistas de acordo com um tema, fornecendo descrições apropriadas;
- Nomeação opcional de níveis, garantindo mais uma camada narrativa para o jogo;

3.2 TECNOLOGIAS UTILIZADAS

3.2.1 JavaScript

JavaScript é uma linguagem de *scripts* leve e dinâmica, implementada originalmente por Brendon Eich. Ela foi desenvolvida inicialmente como uma linguagem para páginas web, interpretada pelos motores de navegadores, mas posteriormente com sua popularização ela ganhou outros usos, podendo ser utilizada hoje para desenvolvimento de servidores, aplicações *desktop* e aplicativos móveis (MDN, 2021).

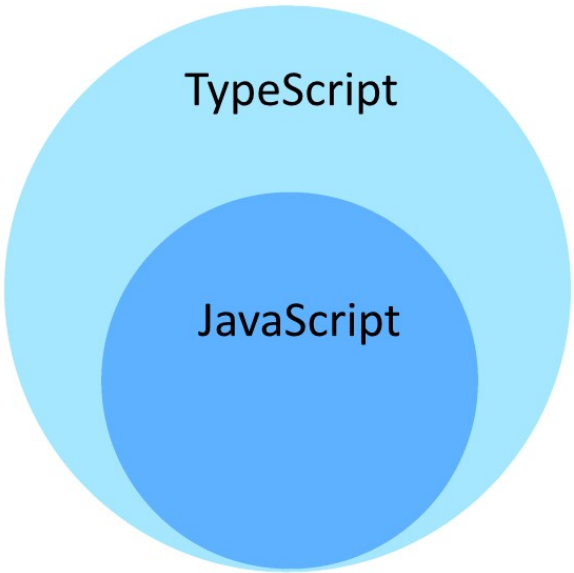
A linguagem foi escolhida para a implementação de toda a plataforma, devido a sua versatilidade, facilidade de aprendizado e quantidade de referências na internet. Com JavaScript, é possível desenvolver o sistema *web*, o aplicativo e o servidor utilizando uma só linguagem. As vantagens são a facilidade de compreensão do código, poder desenvolver as habilidades com uma só linguagem e aplicá-las em todos os ecossistemas, além da fácil reutilização de lógica e código.

3.2.2 TypeScript

JavaScript foi criada para usos rápidos, para resolver pequenos casos de uso em um ambiente *web*. Porém, com o passar dos anos, ela foi sendo empregada em usos mais complexos e cenários mais críticos, de forma que algumas de suas características vantajosas começaram a se tornar um problema. Uma delas é a dinamicidade de alocação de variáveis.

Para resolver muitos destes problemas e tornar a linguagem mais confiável, o TypeScript foi criado. TypeScript é um *superset* de JavaScript, ou seja, é uma linguagem que tem JavaScript como base, mas adiciona regras e recursos por cima da linguagem original, como mostra a Figura 4. TypeScript tem como principal recurso a checagem estática de tipos, ou seja, com TypeScript, é possível realizar anotações de tipos para assegurar a estrutura dos dados e variáveis, assim como em linguagens como C ou Java (TYPESCRIPT, 2021).

Figura 4 – Relação entre TypeScript e JavaScript



Fonte: Greer (2016)

O TypeScript é usado no projeto para prevenir possíveis erros de execução e facilitar o processo de escrita de código. Devido a sua tipagem estática, alguns editores

de código são capazes de executar TypeScript durante a escrita para capturar problemas e oferecer informações sobre as variáveis, funções e classes, agilizando o desenvolvimento. Apesar da sintaxe do sistema ser escrita completamente em TypeScript, no processo de compilação todo o código é transformado em JavaScript para ser executado pelos motores e transpiladores, de forma que o código final é totalmente escrito em JavaScript.

3.2.3 Node.js

Node.js é uma plataforma desenhada para desenvolver aplicações de rede escaláveis. É uma *runtime* de Javascript assíncrona e baseada em eventos, inspirada na *Event Machine* do *Ruby* e no *Twisted* do *Python*. O modelo de concorrência do Node é baseado em eventos, assim como um navegador, garantindo chamadas não bloqueantes e portanto a ausência de *deadlocks* (NODE.JS, s.d.).

O Node.js foi construído por cima do motor V8 da *Google*, o motor do navegador *Google Chrome*. Utilizando este motor, ele é capaz de executar código JavaScript fora de um navegador, expandindo o uso da linguagem para outros ambientes (ZHAOA; XIAB; LE, 2013).

A plataforma foi escolhida para o desenvolvimento da API, pois possibilita a escrita de um servidor escalável utilizando JavaScript, uma linguagem já conhecida.

3.2.4 MongoDB

O MongoDB é um banco de dados não relacional, orientado a documentos (SUBRAMANIAN, 2019). Ele salva os dados em documentos com estruturas similares a JSON, através de modelos que mapeiam os dados diretamente para objetos no código da aplicação, o que torna o acesso mais simples do que em bancos de dados relacionais (MONGODB, s.d.).

Uma das grandes vantagens do MongoDB é que, por ser orientado a documentos e esquemas, os documentos da base de dados não precisam ter os mesmos campos. Isto é especialmente valioso durante os estágios iniciais do desenvolvimento, onde você não precisa adicionar ou remover colunas no esquema (SUBRAMANIAN, 2019). Os dados podem simplesmente ser criados com novos campos, ou com campos ausentes, de forma que mudanças no domínio da aplicação ou nos requisitos do sistema, que são comuns nas primeiras fases de desenvolvimento, se dão de forma mais ágil e facilitada. Esta é a principal razão do MongoDB ter sido escolhido como o banco de dados do *Gametask*.

3.2.5 ReactJS

React é uma biblioteca de criação de interfaces de usuário, desenvolvida pela equipe do *Facebook* para lidar, inicialmente, com problemas no seu sistema de anúncios. A biblioteca oferece uma maneira declarativa de construir interfaces, de forma que o

programador apenas declara como a interface se comporta de acordo com os dados, e então o React cuida da renderização das telas.

Para renderizar as interfaces, o React constrói uma representação virtual da estrutura de elementos HTML, chamada de *VirtualDOM*. A *VirtualDOM* é uma estrutura de dados que representa toda a hierarquia de componentes React e é mantida na memória. Quando o estado de um componente React muda, a biblioteca compara a árvore do *VirtualDOM* resultante com a árvore anterior, e renderiza apenas as diferenças entre elas no documento HTML. Isto acaba sendo menos custoso do que manipular os elementos imperativamente, o que acaba garantindo um ganho de performance (SUBRAMANIAN, 2019).

React foi escolhido para o desenvolvimento do Gametask por ter uma comunidade ativa, com várias referências na internet para consulta; pela facilidade de aprendizado, já que a interface é construída puramente com JavaScript e JSX (a linguagem de marcação do React, muito similar a HTML); e por quê a biblioteca é desacoplada de qualquer API de interfaces, ou seja, pode ser usada em mais de um ambiente. No caso do *Gametask*, o React é integrado com a *web* através da biblioteca *ReactDOM*, e utilizado para a construção do aplicativo móvel com o *React Native*.

3.2.6 React Native

React Native é a biblioteca que serve de ponte entre o React e as APIs nativas dos dispositivos móveis. Assim como o *ReactDOM* usa as abstrações do React para criar as interfaces nos documentos HTML, o React Native utiliza uma instância embutida do *framework JavaScriptCore* para renderizar componentes nativos específicos de cada plataforma através de chamadas JavaScript (OCCHINO, 2015).

A biblioteca foi escolhida para desenvolver o aplicativo móvel pois, com exceção dos componentes específicos de plataformas móveis, todo o código JavaScript e React pode ser reutilizado nos sistemas *web* e móvel. Isto torna o desenvolvimento muito mais ágil, e a curva de aprendizado muito menor.

3.2.7 Expo

O Expo é uma ferramenta utilizada para facilitar o desenvolvimento móvel com React Native. Normalmente, ao iniciar no desenvolvimento móvel, é necessário instalar diversas ferramentas na máquina para conseguir compilar o código e gerar um executável para testar nos sistemas operacionais, geralmente *Android* e iOS. Além disso, é preciso manipular muito código nativo para habilitar o uso de algumas APIs nativas, como acesso a câmera e geolocalização. Este processo pode levar um tempo considerável muitas vezes, especialmente quando o desenvolvedor pretende criar um MVP ou uma versão rápida.

O Expo então fornece um ambiente com as ferramentas nativas pré-configuradas para agilizar esta etapa. A ferramenta fornece um aplicativo móvel instalável nas lojas dos

sistemas operacionais, que se comunica com a ferramenta de desenvolvimento. No ambiente de desenvolvimento do Expo, o desenvolvedor tem a opção de criar seu código React em JavaScript, e enviar o pacote com o código minificado e compilado para o aplicativo do Expo, que então o executa em cima de um ambiente nativo preparado. Desta forma, é possível testar o código em React de maneira ágil, além de publicar versões de testes que são executadas da mesma forma (FERNANDES, 2018).

A velocidade e facilidade de desenvolvimento que o Expo trás fez com que ele fosse escolhido para desenvolver o aplicativo móvel do GameTask.

3.3 MODELAGEM DO DOMÍNIO

Nesta seção será descrita a modelagem do domínio da aplicação, inspirado nas definições de (EVANS, 2003). O domínio é o conjunto de regras de negócio que definem a aplicação, que são obtidas a partir requisitos funcionais e não-funcionais. O domínio aqui abordado é constituído das entidades presentes no sistema, seus casos de uso e mecânicas específicas. A vantagem de realizar esta modelagem é a fácil visualização das entidades presentes no sistema, assim como quais usuários realizarão cada tarefa, quais regras de negócio devem ser implementadas e como estes elementos devem ser desacoplados do restante da implementação em código.

Primeiramente, serão definidos os requisitos. Os requisitos da aplicação são descritos tendo em mente os elementos de gamificação que serão abordados na aplicação, assim como as funcionalidades indispensáveis para um *software* para a internet. No apêndice A estão os diagramas de classe, contendo as entidades do sistema, e casos de uso, que demonstram as interações que os atores devem possuir com o mesmo. Adiante, há uma breve explicação sobre a elaboração destes diagramas, o que finaliza a modelagem do domínio.

3.3.1 Requisitos Funcionais

1. Um usuário deve poder criar sua conta;
2. Um usuário deve poder realizar *login* com sua conta;
3. Um usuário deve poder criar um jogo, se tornando o Administrador daquele jogo;
4. Um administrador deve poder convidar outras pessoas para seu jogo, que se tornarão jogadores;
5. Um jogador deve poder entrar em um jogo que foi convidado;
6. Um jogador deve poder alterar seus dados básicos, como nome e avatar;
7. Um administrador deve poder criar conquistas;

8. Um administrador deve poder criar títulos;
9. Um administrador deve poder associar títulos a conquistas;
10. Um administrador deve poder criar atividades;
11. Um administrador deve poder alterar o tema e as informações básicas do seu jogo;
12. Um administrador deve poder configurar os níveis do jogo;
13. Um administrador deve poder configurar as patentes do jogo;
14. Um jogador deve poder obter conquistas;
15. Um jogador deve poder saber quais conquistas recebeu, e quais falta receber;
16. Um jogador deve poder obter títulos;
17. Um jogador deve poder concluir atividades;
18. Um jogador deve poder selecionar um de seus títulos como título principal;
19. Um jogador deve poder ganhar níveis e patentes;
20. Um jogador só poderá obter conquistas, títulos e completar atividades mediante aprovação de um administrador;
21. Um jogador deve ser adicionado ao placar de líderes ao ganhar pontos de experiência;
22. Um jogador deve poder ver seu progresso de experiência para o próximo nível;
23. Um jogador deve ser capaz de visualizar o placar de líderes;
24. O placar de líderes deve ser ordenado pela quantidade de pontos de experiência, de forma decrescente;
25. Um administrador deve poder recomeçar o placar de líderes, zerando as pontuações;
26. Um jogador deve poder visualizar uma linha do tempo contendo as atividades executadas por todos os jogadores do seu jogo.

3.3.2 Requisitos Não-Funcionais

1. Os dados da plataforma devem ser persistidos em um banco de dados não relacional;
2. As regras de negócio da aplicação devem ser processadas por um sistema desacoplado, um servidor;
3. Os clientes do servidor devem receber e enviar dados para processamento a partir de requisições HTTP;

4. Um administrador deve ter acesso às funções de administração a partir de uma plataforma *web*, totalmente responsiva;
5. Um jogador deve ter acesso aos seus recursos a partir de um aplicativo móvel;
6. O aplicativo móvel deve ser compatível com os sistemas operacionais Android e iOS;
7. Um usuário que não realizou login não deve ter acesso aos recursos de criação/participação de jogos, assim como os recursos de adminstrador e jogador;
8. O sistema deve mostrar que uma requisição HTTP está em progresso através de *feedback* visual sempre que possível

3.3.3 Entidades

As figuras Figura 29, Figura 30 e Figura 31 do apêndice A demonstram as entidades do domínio da aplicação. As figuras fazem partes do mesmo diagrama, inspirado no modelo de diagrama de classes de UML. Estas entidades podem ser implementadas através de estruturas de dados, coleções no bando de dados e/ou classes na aplicação.

3.3.4 Casos de uso

As figuras Figura 32, Figura 34 e Figura 33 do apêndice A definem os casos de uso da plataforma. Cada figura mostra quais ações os atores devem poder realizar. O principal ator é o Usuário, que pode ser dividido entre Administrador (aquele que gerencia um jogo) e Jogador. Um usuário pode assumir o papel de administrador, jogador ou ambos no mesmo jogo.

3.4 DESENVOLVIMENTO DO SISTEMA SERVIDOR

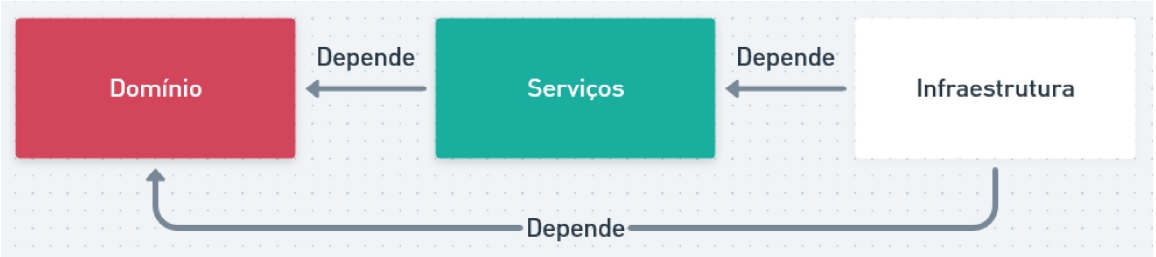
O servidor do Gametask é desenvolvido no formato API REST. Seu objetivo é processar as requisições que os outros sistemas farão, aplicar as regras de negócio da aplicação e persistir as informações no banco de dados. A aplicação é desenvolvida em TypeScript, usando o MongoDB para a persistência.

3.4.1 Arquitetura

A arquitetura do sistema é dividida em camadas, seguindo os princípios da Arquitetura Limpa (MARTIN, 2017), cada uma com suas responsabilidades bem definida. As camadas são **domínio**, **serviços** e **infraestrutura**, como mostra a Figura 5. A hierarquia das camadas define a ordem de dependência entre elas: a camada de domínio fica no topo da hierarquia e é autocontida, não dependendo de nenhuma outra camada. A camada logo abaixo é a de serviços, dependendo apenas das entidades e protocolos definidos na

camada de domínio. Por fim, a camada de infraestrutura coordena a camada de serviços, dependendo também da camada de domínio para acessar serviços tanto internos quanto externos.

Figura 5 – Arquitetura em Camadas do Servidor



Fonte: Elaborado pelo autor (2021).

A vantagem da arquitetura em camadas é que o código das camadas de mais alto nível fica auto contido, não enxergando as camadas abaixo. De forma similar, camadas mais abaixo apenas consomem os recursos das camadas acima, realizando processamento se necessário. Desta maneira, cada camada pode ter sua única responsabilidade sem que o código de uma camada interfira na outra, o que permite que novas funcionalidades sejam implementadas em uma camada por vez, facilita a detecção de erros e também na manutenção do código no longo prazo.

3.4.1.1 Domínio

A camada de domínio é responsável pelas regras de negócio da aplicação. Nela, são definidos os contratos e protocolos que especificam as entidades do sistema e as interfaces de APIs externas. Os arquivos definidos nesta camada podem ser dos seguintes tipos:

- **Adapters**: Adaptadores são classes responsáveis por receber os dados em um formato, processá-los e retornar um objeto em outro formato. São usadas no sistema para traduzir os dados que a camada de serviços manipula para o formato utilizado pelos repositórios, de forma abstraída;
- **DTOs**: Objetos de Transferência de Dados, do inglês *Data Transfer Objects*, são interfaces que definem os tipos de dados que transitam entre camadas na aplicação. Eles são principalmente usados para definir o formato dos objetos passados pelos controladores para os serviços ou para os provedores. Eles diferem dos **adapters** de forma que apenas definem contratos, não realizando quaisquer transformações nos dados passados;
- **Entities**: São interfaces que modelam o formato das entidades do sistema. Elas definem quais atributos as entidades possuem, e quais seus tipos de dado;

- **Errors**: São interfaces que modelam o formato dos objetos de erro gerados pelo sistema. Ajudam a padronizar os erros, de forma que quando implementados, diferentes erros processam os dados de sua própria maneira, mas devem respeitar um molde para demonstrá-los;
- **Providers**: São interfaces que definem contratos que devem ser respeitados quando a aplicação usar bibliotecas e plugins externos, os provedores. Eventualmente a aplicação precisará acessar serviços externos, cujos códigos são abstraídos e inacessíveis. Para respeitar o Princípio de Substituição de Liskov, em vez de depender das implementações destes serviços, a aplicação depende apenas do contrato estabelecido pelo *provider*. As implementações destes serviços também respeitam o contrato, permitindo que diferentes implementações de um mesmo serviço externo possam ser usadas ou trocadas na aplicação sem afetar o comportamento do sistema;
- **Repositories**: O padrão *repositories* permite abstrair a camada de dados da aplicação, fornecendo uma API baseada em coleções, com métodos personalizados com base nos dados que queremos acessar. Desta forma, em vez de usar diretamente os motores do banco de dados, ORM ou APIs externas, o sistema utiliza o repositório de dados, se tornando completamente independente do serviço de persistência de dados utilizado. Nesta parte da camada de domínio, são definidas as interfaces que os repositórios da aplicação devem respeitar, de forma que os serviços do servidor possam depender apenas das abstrações, respeitando o Princípio de Substituição de Liskov.
- **Rules**: São classes que gerenciam um conjunto de regras específicas da aplicação. Como o GameTask é uma aplicação com elementos de jogos, alguns destes elementos dependem de regras complexas para funcionar, como o sistema de incremento de níveis. Nesta seção da camada de domínio é definido o funcionamento destas mecânicas, de forma que as outras camadas dependam das regras definidas aqui. Desta forma, quaisquer alterações mecânicas são abstraídas das outras camadas da aplicação, e podem ser feitas em uma única classe ou objeto. Cada classe de regra é responsável por apenas uma única regra, respeitando o Princípio de Responsabilidade Única.

3.4.1.2 Serviços

A camada de serviços é responsável pelo gerenciamento das regras de negócio da aplicação. Cada serviço é a virtualização de um caso de uso, fazendo uso das regras, repositórios e provedores definidos no domínio para processar uma requisição feita pelo usuário.

De forma a tornar os serviços independentes das implementações da camada de infraestrutura, os serviços dependem apenas das interfaces declaradas na camada de domínio. As implementações são injetadas pelo construtor, de acordo com o Princípio de

Inversão de Dependências, fazendo com que os serviços não enxerguem estas implementações e se tornem completamente desacoplados das camadas debaixo. Cada classe de serviço é responsável por um e somente um caso de uso, respeitando também o Princípio de Responsabilidade Única.

Esta, junto com a camada de domínio, são as camadas mais importantes da aplicação, e juntas a definem completamente. Estas duas camadas são independentes de *frameworks*, bibliotecas e quaisquer serviços externos. Elas são desacopladas o suficiente para que sejam dissociadas de um sistema servidor, podendo igualmente representar um sistema *desktop*, um aplicativo ou um programa de linha de comando.

3.4.1.3 Infraestrutura

A camada de infraestrutura é a camada que implementa os serviços utilizados pelas camadas acima, e realiza a comunicação do sistema com o mundo exterior. Esta é a camada que define o sistema como uma API REST, expondo seus serviços através de rotas acessíveis via requisições HTTP. Também é aqui que o sistema cria suas opiniões, escolhendo *frameworks*, bibliotecas, banco de dados e outros serviços externos para cumprir as responsabilidades definidas na camada de domínio. Esta camada é responsável também pelo tratamento de erros, capturando os erros que ocorrem nesta e nas camadas acima, e devolvendo-os de forma legível ao usuário. A estrutura da camada contém os seguintes tipos de arquivos:

- **Container**: Os containers são onde as implementações dos repositórios e dos provedores são associadas às suas interfaces, por meio de uma biblioteca de injeção de dependências. Aqui também são implementados os provedores do sistema, seguindo as interfaces da camada de domínio;
- **Errors**: Aqui são definidas as classes de erros personalizadas do sistema, seguindo as interfaces da camada de domínio. Estas classes ajudam outras seções da aplicação a diferenciar erros internos de erros externos ou desconhecidos, além de padronizar as respostas de erro para os consumidores da aplicação;
- **Http**: Nesta seção é implementada a comunicação HTTP do sistema. É configurado o servidor HTTP, as rotas da aplicação e pontos de acesso do sistema, as classes controladoras que instanciam as classes de serviço com suas dependências, e as funções *middleware*. Estes arquivos são os responsáveis pela comunicação externa, definindo o formato de entrada e saída de dados da aplicação;
- **Mongoose**: Os arquivos nesta seção são implementações da camada de dados do sistema, que gerenciam a conexão com o MongoDB. Mongoose é o nome da biblioteca usada para acessar o *driver* do banco de dados. Aqui, são definidos os esquemas do banco, as estruturas de persistência de dados usadas pelo MongoDB, respeitando

as entidades definidas na camada de domínio. Também são implementados os repositórios, que acessam os dados armazenados de acordo com os protocolos definidos também na camada de domínio.

3.4.2 Módulos

O sistema é modularizado, de forma que algumas regras de negócio e pontos de entrada são agrupados de acordo com um conceito de responsabilidade única. Os módulos representam um conjunto maior de casos de uso, cujas responsabilidades dizem respeito a um fluxo específico da API. Os módulos da aplicação são os seguintes:

- **Shared**: O módulo compartilhado é o principal módulo do sistema. Este é o módulo que inicia os serviços HTTP, e importa os demais módulos, agrupando-os em um único sistema. É nele que fica o arquivo inicial da aplicação, que o Node.js utiliza para executá-la. Este módulo fornece aos demais módulos recursos compartilhados, como entidades e repositórios que são utilizados em mais de um módulo;
- **Users**: O módulo de usuários contém os pontos de acesso e regras de negócio que dizem respeito a gestão do usuário. Ele expõe os serviços de alteração, criação e listagem de um usuário, assim como o serviço de autenticação;
- **Games**: O módulo de jogos agrupa os pontos de acesso e regras de negócio responsáveis por criar e estruturar um jogo. Estas regras incluem atualizar, criar, listar e remover atividades, conquistas, jogos, placar de líderes e títulos;
- **Players**: O módulo de jogadores concentra as funcionalidades que dizem respeito ao gerenciamento de um jogador, e as ações que um jogador pode executar. Estas regras constituem a criação de um novo jogador, a requisição de atividades e conquistas, o fornecimento destas para um jogador, o gerenciamento dos títulos do jogador, assim como a linha do tempo da aplicação.

3.4.3 Fluxo do sistema

Ao utilizar a API, o usuário percorre um fluxo padrão, com variações de processamento dependendo da requisição feita.

Inicialmente, o usuário deve fazer uma requisição HTTP para a URL do servidor, com um método de acordo com o tipo de processamento que deseja fazer: **GET** para obtenção de recursos; **POST** para a criação de um recurso; **PATCH** para a alteração parcial de um recurso e **DELETE** para a remoção de um recurso. Nota-se a ausência do método **PUT**. O método PATCH é utilizado em seu lugar pois a API oferece apenas serviços de alteração parcial de um recurso, e não alteração total, o que por convenção é geralmente assossiado ao método PATCH em vez de PUT.

A API fornece algumas rotas públicas, mas para a maioria das rotas é necessário que o usuário se autentique. Este processo se dá através da rota de autenticação, onde o usuário fornece e-mail e senha cadastrados, e como resposta recebe um *token* que garante sua autenticidade. Com o *token*, o usuário pode acessar algumas rotas gerais, como listagem e criação de jogos, criação de jogadores e alteração de seus dados básicos. Para acessar as rotas de gerenciamento de um jogo, é necessário selecioná-lo através rota de seleção de jogos. Como resposta, a API retornará um novo *token* que pode ser usado para informar qual jogo está sendo manipulado nas requisições. Algumas destas rotas possuem ainda mais uma camada de segurança, impedindo que usuários que não sejam administradores do jogo não tenham acesso aos recursos oferecidos.

3.4.4 Bibliotecas

Abaixo estão listadas bibliotecas usadas para a implementação de um ou mais recursos importantes do sistema:

- **Aws SDK**: Biblioteca que permite conectar a aplicação ao *Amazon Web Services*, a plataforma de serviços da nuvem na Amazon. A API utiliza o serviço de armazenamento para guardar as imagens enviadas pelos usuários;
- **BCryptJS**: Esta biblioteca fornece métodos de criptografia usados pelo sistema para criptografar a senha dos usuários durante a criação, e para comparar as senhas enviadas com as senhas salvas durante a autenticação;
- **Express**: É um *framework* leve de criação de servidores, contendo abstrações e facilitadores que tornam o desenvolvimento de um servidor em Node.js mais ágil;
- **Jest**: Ferramenta de ambiente de testes automatizados em JavaScript, usada para realizar os testes unitários e de integração que garantem o funcionamento do sistema;
- **JsonWebToken**: A biblioteca implementa a geração e verificação de códigos *JSON Web Token* (JWT), usados como *token* de autenticação no sistema;
- **Mongoose**: É a biblioteca que abstrai o *driver* do MongoDB, oferecendo formas mais simples de se conectar a uma base e dados e manipular coleções. O MongoDB possui um *driver* nativo em JavaScript, mas as abstrações e funcionalidades extras que o Mongoose fornecem tornam seu uso muito mais vantajoso;
- **Multer**: É uma biblioteca que oferece *middlewares* para o Express capazes de interceptar o envio de arquivos em requisições HTTP. Através do Multer, é mais simples capturar os arquivos enviados e processá-los no servidor;
- **Tsyringe**: É a biblioteca de injeção de dependências utilizada na aplicação. Ela funciona através de *decorators*, possibilitando decorar as classes e métodos sinalizando

onde uma dependência deve ser injetada. Então, quando um controlador precisa resolver uma instância de um provedor, repositório ou serviço, o Tsyringe injeta automaticamente as dependências sinalizadas.

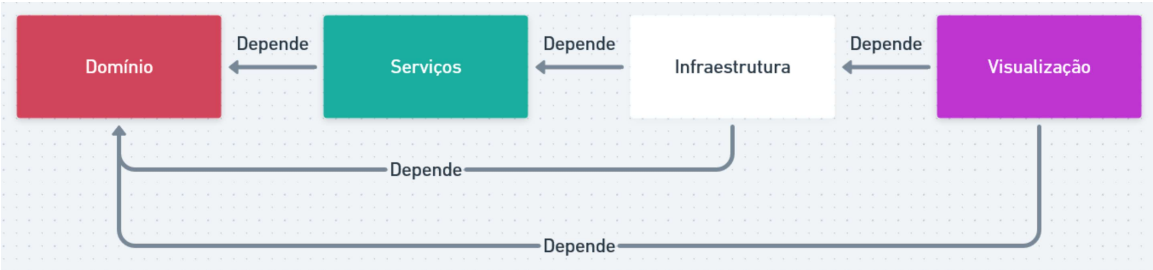
3.5 DESENVOLVIMENTO DA PLATAFORMA WEB

A plataforma *web* é desenvolvida no modelo SPA, utilizando a biblioteca React. O objetivo da plataforma é expor as funcionalidades de gerência para o gestor do jogo, permitindo-o criar jogos, configurá-los e convidar jogadores. A plataforma deve ser completamente responsiva, permitindo ao gestor acessá-la do seu *smartphone* para conveniência, mas o principal meio de acesso é o computador.

3.5.1 Arquitetura

A arquitetura da plataforma segue um modelo similar à arquitetura do servidor. Ela possui as mesmas camadas do servidor, porém com a adição de uma outra camada: a camada de visualização. Esta é a camada mais abaixo, responsável por renderizar a interface do usuário, manipulando elementos visuais, estilos e animações. A Figura 6 mostra a relação entre estas camadas.

Figura 6 – Arquitetura em Camadas da Plataforma *Web*



Fonte: Elaborado pelo autor (2021).

A camada de visualização depende das camadas de domínio e infraestrutura, mas não enxerga a camada de serviços. Os serviços funcionam como uma camada intermediária que conecta as camadas de domínio e infraestrutura, e outras partes da plataforma não têm acesso a eles. Na camada de visualização é onde os componentes React são criados e onde as páginas são montadas.

Conceitualmente, a camada de visualização deveria ser a única a enxergar a biblioteca de criação de interfaces. A camada de infraestrutura apenas disponibiliza os controladores e define as rotas da aplicação, e deve fazê-lo de forma declarativa e genérica. Porém, para fins de agilidade no desenvolvimento, concessões foram feitas e a camada de

infraestrutura utiliza componentes *React* para definir as rotas da aplicação, e *hooks* do React para criar os controladores.

3.5.1.1 Domínio

A camada de domínio é similar a mesma camada do servidor. A grande diferença é que o servidor processa os dados e precisa gerenciar as regras de negócio; a plataforma *web* apenas consome os dados processados, portanto, a camada de domínio possui somente a responsabilidade de declarar as interfaces de entidades, provedores e repositórios, sem a necessidade de processamento lógico.

Os arquivos encontrados na camada de domínio da plataforma são os mesmos encontrados na camada de domínio do servidor: **DTOs**, **Entities**, **Providers** e **Repositories**. Seus códigos são construídos da mesma forma.

Uma diferença notável é que as dependências não são injetadas automaticamente através de uma biblioteca como o Tsyringe. Isto por quê alguns provedores da plataforma são implementados na camada de infraestrutura, mas outros são implementados na camada de visualização, através de Contextos React. Os repositórios e provedores são obtidos então utilizando o padrão *Factory*, ou fábrica, que será abordado mais adiante.

3.5.1.2 Serviços

A camada de serviços possui a mesma responsabilidade de sua camada homônima no servidor, e é construída praticamente da mesma maneira. Porém, o tratamento de erros das regras de negócio é realizado nesta camada. Desta forma, os serviços sempre retornam aos controladores objetos que contém o resultado, em caso de sucesso, mas também o erro caso algum problema aconteça.

3.5.1.3 Infraestrutura

A camada de infraestrutura na plataforma *web* possui responsabilidades similares com a mesma camada no servidor. Ela define diversas opiniões no sistema, escolhendo os provedores de requisição HTTP e outros serviços, e coordenando os seviços da aplicação por meio de controladores. Ela expõe controladores para as camadas abaixo, e instâncias dos provedores e repositórios através de fábricas. Fábricas são classes ou funções que criam uma instância de um objeto sob demanda, de forma centralizada. Através das fábricas, as camadas de infraestrutura e visualização podem obter instâncias dos provedores e repositórios dependendo apenas de suas interfaces, sem conhecer suas implementações. A camada está organizada da seguinte forma:

- **Controllers**: Os controladores são funções usadas na camada de visualização, que coordenam os serviços da aplicação. Eles são desenvolvidos utilizando os *hooks* React, de forma que os componentes da camada abaixo apenas precisam chamá-los para

controlar um fluxo de dados. Cada controlador é responsável por um único serviço, e expõe métodos para executá-los, assim como alguns metadados;

- **Providers**: Os provedores implementados na camada de infrastrutura seguem o modelo daqueles implementados na mesma camada no servidor;
- **Repositories**: Os repositórios são implementados exatamente da mesma maneira que na camada de infraestrutura do servidor;
- **Routes**: As rotas do sistema são definidas através de componentes React de roteamento. A camada de visualização agrega estes componentes formando a árvore de rotas total da aplicação. Esta parte da camada é a fronteira entre as camadas de infraestrutura e visualização, portanto contém importações de arquivos de ambas camadas.

3.5.1.4 Visualização

A camada de visualização é responsável por renderizar os elementos de interface visual para o usuário. Os componentes visuais são criados com React, que injeta os elementos HTML dinamicamente na árvore de elementos do navegador através do JavaScript.

É através desta camada que o usuário interage com o sistema. A camada também abriga o arquivo principal da aplicação, que importa todos os outros objetos e módulos.

Dentro desta camada, os arquivos estão organizados da seguinte forma:

- **Components**: Os componentes React que são reutilizados são declarados aqui. Elementos que representam uma pequena parte do *layout* de uma página ou que possuem um comportamento ou lógica complexo são abstraídos e isolados em componentes;
- **Contexts**: Contextos são componentes React capazes de armazenar estados e compartilhar estes estados com toda a árvore de elementos definida abaixo deles na hierarquia de forma otimizada. Estes contextos implementam provedores que precisam utilizar recursos do navegador ou manipular estados da aplicação, como os provedores de armazenamento e tema;
- **Helpers**: Funções auxiliares que são reutilizadas em múltiplos componentes são isoladas aqui;
- **Hooks**: Os *hooks* são uma funcionalidade da biblioteca React que permite declarar funções isoladas que manipulam estados e lógica de componente. Estas funções então podem ser utilizadas pelos componentes, se anexando a eles como um 'gancho', de forma que influenciam seu comportamento e fluxo de renderização;
- **Pages**: São os componentes React que representam uma única página *web*. Estes componentes têm a responsabilidade de processar a lógica da página, utilizando os

controladores para gestão de dados e montando o *layout* com a união de outros componentes;

- **Validation**: Os esquemas de validação utilizados para validar campos de formulários são salvos aqui. Ao enviar um formulário, o programa submete os dados enviados ao esquema para verificar se sua estrutura, tipos e conteúdo são válidos antes de enviar uma requisição ao servidor.

3.5.2 Módulos

A plataforma *web* é modularizada assim como o servidor. Seus módulos agregam páginas que representam um único fluxo ou a manipulação de uma única entidade.

- **Shared**: O módulo compartilhado do sistema é onde os demais módulos são agregados. Ele é responsável por inicializar a aplicação, definir os contextos, componentes e estilos globais e definir quais rotas são acessíveis para quais níveis de autenticação;
- **Landing**: Este módulo define a página pública da aplicação, onde o usuário pode se cadastrar e se autenticar;
- **User**: O módulo do usuário contém as regras de negócio e componentes que compõem a página de criação e seleção de jogos;
- **Dashboard**: O módulo do *dashboard* contém a página principal, que mostra as atividades, conquistas e placar de líderes; e também é responsável pelas páginas de gerenciamento de atividades e conquistas;
- **Game Management**: Este módulo é responsável pela página de configuração do jogo, onde é possível definir os níveis, patentes, tema e informações básicas do jogo;
- **Manage Players**: Este módulo contém a página de gerenciamento de jogadores, onde é possível manipular os títulos do jogo, e aceitar ou recusar as requisições de conclusão de atividades e conquistas.

3.5.3 Fluxo do sistema

O usuário pode acessar o sistema por uma URL através de um navegador de internet. Ao acessar, ele se encontra na página inicial, onde pode realizar seu cadastro ou fazer o *login*, caso já seja cadastrado.

Ao efetuar *login*, ele tem acesso ao *Lobby*, uma página que lista todos os jogos que o usuário administra. Nesta página é possível criar novos jogos e gerar códigos de convite para os jogos existentes, que podem ser enviados para os jogadores. Jogadores podem entrar em um jogo de um código desta forma, ou lendo um código QR presente na seção de convite do jogo nesta página.

Selecionando um jogo no *Lobby*, o usuário adentra o *dashboard* do Gametask. Na primeira página, é possível visualizar o placar de líderes atual, e restaurá-lo se desejar. Também pode-se visualizar todas as atividades e conquistas já cadastradas no jogo. A partir desta página, é possível gerenciar os jogadores, atividades, conquistas ou o próprio jogo.

Nas páginas de gerenciamento de atividades e conquistas, o usuário pode visualizar todos os elementos daquele tipo cadastrados, assim como cadastrar novos elementos, removê-los ou atualizá-los. Especialmente no cadastro de conquistas, o usuário pode selecionar um título que a conquista garante ao ser concluída. É possível criar um título imediatamente antes de associá-lo à uma conquista nesta página, agilizando o processo.

Na página de gerenciamento do jogo, o usuário pode atualizar a imagem do jogo, assim como seu nome, descrição e tema. O tema escolhido define as cores da plataforma *web* enquanto estiver no *dashboard* deste jogo, assim como as cores do aplicativo quando o respectivo jogo estiver sendo jogado. Nesta página é possível também definir os níveis existentes no jogo, configurando os pontos de experiência necessários para obtê-los, e também configurar as patentes do jogo, associando-as aos níveis já criados.

Finalmente, na página de gerenciamento de jogadores, o usuário tem acesso à todas as requisições de conclusão de atividades e conquistas feitas pelos jogadores, assim como os títulos cadastrados no jogo. O usuário pode então criar novos títulos, atualizá-los e removê-los. Pode-se também aceitar ou remover as requisições, ou visualizar mais detalhes sobre elas antes de tomar uma decisão.

3.5.4 Bibliotecas

A aplicação *web* utiliza algumas bibliotecas adicionais para implementar recursos importantes:

- **Axios**: Biblioteca que implementa um cliente HTTP para ser utilizado nos navegadores ou no Node.js;
- **CryptoJS**: Biblioteca de criptografia utilizada para gerar os códigos de convite para os jogos;
- **Formik**: Biblioteca que fornece componentes e *hooks* React para a construção de formulários mais performáticos;
- **JsonWebToken**: A mesma biblioteca utilizada no servidor para criar e verificar os tokens JWT. É usada aqui para decodificar o token recebido;
- **Polished**: Uma biblioteca que disponibiliza ferramentas úteis para a criação de estilos em JavaScript;
- **React Router Dom**: Biblioteca que fornece componentes de roteamento em React;

- **React Toastify**: Uma biblioteca que oferece uma API para criar balões de notificação na aplicação;
- **Styled Components**: Biblioteca que permite escrever estilos em linguagem CSS e associá-los diretamente aos componentes React, criando componentes com estilos isolados;
- **Yup**: Biblioteca de validação de dados, utilizada para validar os formulários da aplicação.

3.6 DESENVOLVIMENTO DO APLICATIVO MÓVEL

O aplicativo móvel é desenvolvido para as plataformas iOS e Android, utilizando React Native. É através do aplicativo que os jogadores podem interagir com o sistema, participando de jogos, obtendo pontos e competindo com seus colegas.

O desenvolvimento do aplicativo móvel é similar ao desenvolvimento da plataforma *web*, dado que ambos são desenvolvidos utilizando a mesma biblioteca. A principal diferença é que os componentes React disponíveis pela biblioteca React Native representam componentes nativos dos sistemas operacionais, como *Views* e *Touchables*, diferente dos elementos HTML disponíveis através do ReactDOM.

Devido a similaridade, a **arquitetura** utilizada é a mesma, com a mesma organização de pastas, as mesmas camadas e as mesmas responsabildidades por camada. Adicionalmente, algumas bibliotecas utilizadas na plataforma *web* são também utlizadas no aplicativo. Estas bibliotecas são: **Axios**, **CryptoJS**[2], **Formik**, **Polished**, **Styled Components**[3] e **Yup**.

3.6.1 Módulos

Assim como os outros sistemas, o aplicativo é dividido em módulos que agrupam conjuntos de código com responsabilidades similares. O módulo principal, onde o aplicativo é inicializado, se chama **Shared** e contém as mesmas responsabilidades do módulo homônimo da plataforma *web*. Os demais módulos são:

- **Authentication**: Este é o módulo de autenticação, que implementa as telas e as regras de negócio de cadastro e autenticação do usuário;
- **Choose Game**: Este módulo contém as telas e regras de negócio disponíveis para os usuários autenticados, antes da seleção de um jogo. Ele permite a alteração dos

[2] O pacote se chama *react-native-crypto-js*. A biblioteca original utiliza o pacote *crypto* do Node.js, mas que não está disponível no ambiente onde o React Native é executado. Portanto, este pacote é a implementação de CryptoJS usando *react-native-crypto*, um substituto para a dependência original.

[3] A biblioteca original disponibiliza elementos HTML para a criação de componentes estilizados. Mas o Styled Components fornece um pacote alternativo, chamado *styled-components/native*, que oferece os elementos nativos do React Native para a criação dos componentes.

dados do usuário, adicionar um jogo através de um código de convite e selecioná-lo para jogar;

- **Selected Game**: Este módulo disponibiliza os fluxos de jogo para um usuário. Através dele são implementados os casos de uso de conclusão de atividades, obtenção de conquistas, alteração de títulos e visualização dos elementos do jogo, como o placar de líderes e o nível do jogador.

3.6.2 Fluxo do Sistema

Ao abrir o aplicativo, o usuário se encontra na tela inicial. Esta é a tela de autenticação, onde ele pode se autenticar, caso já seja cadastrado, ou então criar um cadastro. Após autenticado, um usuário é direcionado para o *Lobby*, assim como na plataforma *web*. O *lobby* do aplicativo, entretanto, permite que o usuário adicione jogos através de um código de convite, enviado por um administrador, ou através da leitura do código QR do jogo. Adicionalmente, o usuário pode também alterar seus dados básicos, como nome e foto de perfil.

Ao selecionar um jogo no *Lobby*, o usuário ganha acesso às telas específicas do jogo. O tema do aplicativo é alterado de acordo com as cores configuradas no jogo escolhido. A página inicial contém uma linha do tempo com todas as atividades efetuadas pelos jogadores, sejam elas cumprimento de atividades, obtenção de conquistas, obtenção de patentes ou aumento de níveis. Ao lado da linha do tempo, está o placar de líderes, com todos os jogadores que pontuaram desde a última restauração do placar.

A segunda página contém uma lista de todas as atividades cadastradas no jogo. O jogador pode então selecioná-las para enviar uma requisição de conclusão, onde informa qual atividade conclui e pode contribuir com informações adicionais, como a data de conclusão.

A terceira e última página é o perfil do jogador. Ele contém o nome e a foto do usuário, além de informações como seu nível atual, quantos pontos de experiência possui e quantos faltam para o próximo nível, sua patente e título atuais e uma lista de todas as conquistas do jogo. As conquistas não obtidas são transparentes, enquanto as já obtidas são opacas. Clicando em uma conquista não obtida, o jogador pode enviar uma requisição de obtenção. Nesta página também o usuário pode alterar seu título.

3.6.3 Bibliotecas

Além das bibliotecas reutilizadas da plataforma *web*, o aplicativo utiliza algumas bibliotecas exclusivas:

- **Async Storage**: Uma biblioteca que permite acessar o armazenamento nativo do dispositivo móvel, através de métodos assíncronos de acesso ao disco;

- **Expo**: O Expo é o ambiente que executa o aplicativo, mas também fornece uma série de pacotes facilitadores com integrações de APIs nativas. Dos vários pacotes existentes, a biblioteca utiliza: *expo-barcode-scanner*, para a leitura de códigos QR; *expo-clipboard*, para a captura de dados da área de transferência; *expo-font*, para a utilização de fontes personalizadas no sistema; e *expo-image-picker*, para selecionar imagens da galeria do dispositivo;
- **React Navigation**: Uma biblioteca de roteamento e navegação em React Native. Ela permite criar navegação com comportamento nativo de forma simplificada;

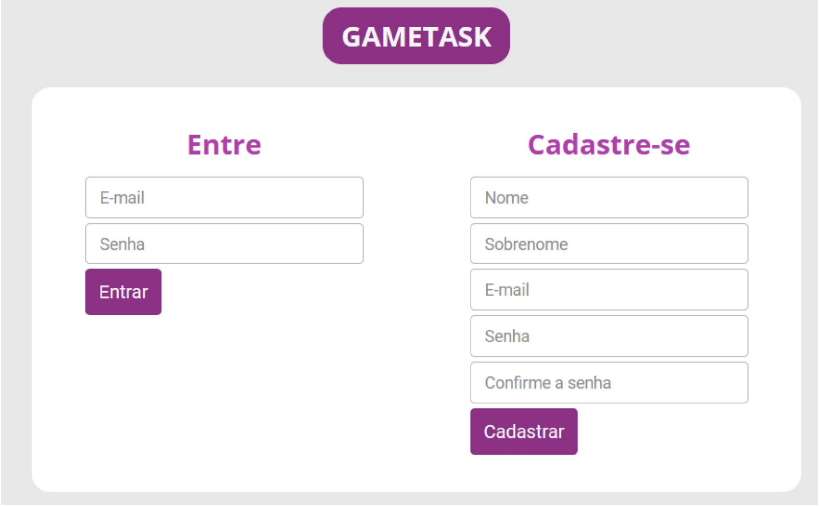
3.7 FLUXOS DE UTILIZAÇÃO DA PLATAFORMA

Esta seção aborda os fluxos que os usuários deve percorrer para utilizar a plataforma.

3.7.1 Autenticação

Para poder acessar a plataforma, um usuário deve se autenticar. O passo inicial é criar uma conta, caso não possua. Para criar sua conta, o usuário deve informar seu nome, sobrenome e e-mail, e fornecer uma senha válida. A Figura 7 mostra a tela de autenticação da plataforma *web*, e a Figura 8 mostra a tela de criação de conta do aplicativo móvel.

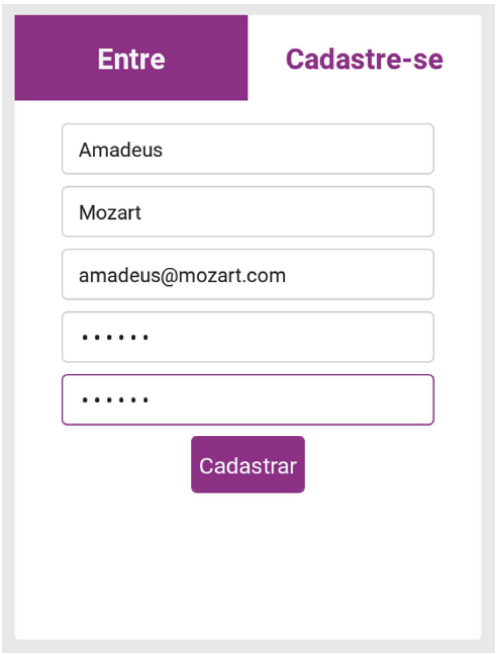
Figura 7 – Tela de autenticação da plataforma *web*



Fonte: Elaborado pelo autor (2021).

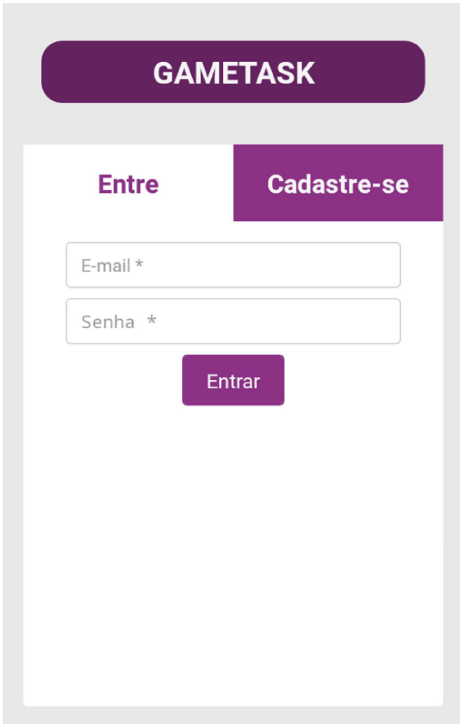
Após criar a conta, o usuário deve se autenticar informando e-mail e senha. A Figura 7 mostra a tela de autenticação da plataforma *web*, enquanto a Figura 9 mostra a tela de autenticação do aplicativo móvel.

Figura 8 – Tela de criação de conta no aplicativo móvel



Fonte: Elaborado pelo autor (2021).

Figura 9 – Tela de autenticação no aplicativo móvel

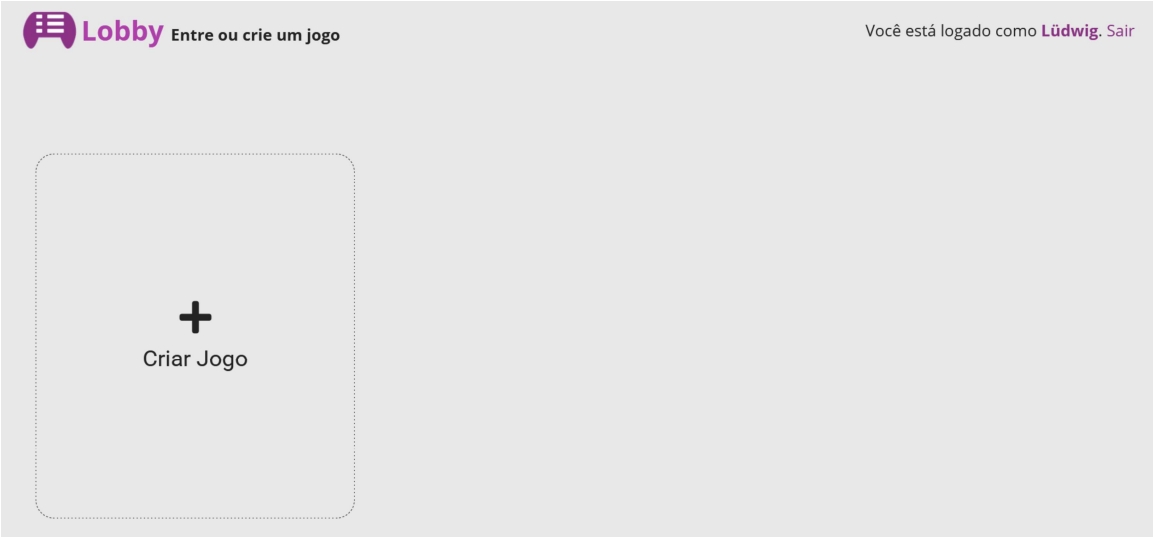


Fonte: Elaborado pelo autor (2021).

3.7.2 Criação de um jogo

Após se autenticar na plataforma *web*, o usuário é direcionado ao *Lobby*, uma tela que lista todos os jogos administrados por este usuário, como mostra a Figura 10. Nesta tela é possível criar um jogo clicando em **Novo Jogo**, preenchendo suas informações como mostra a Figura 11.

Figura 10 – Tela de listagem de jogos



Fonte: Elaborado pelo autor (2021).

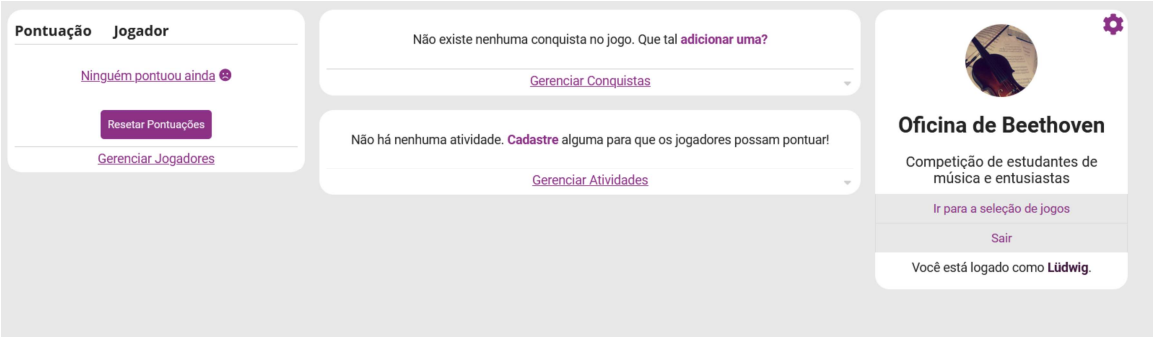
Figura 11 – Formulário de criação de um jogo

Fonte: Elaborado pelo autor (2021).

3.7.3 Configuração do jogo

Ao entrar em um jogo, o usuário tem acesso à tela da Figura 12. Através do painel da direita, o usuário tem acesso à seção de configurações do jogo. Aqui o usuário pode: alterar as informações do jogo, inclusive o seu tema; configurar os níveis do jogo; e configurar suas patentes. As figuras 13, 14 e 15 mostram as seções onde é possível realizar estas configurações.

Figura 12 – Tela inicial do jogo

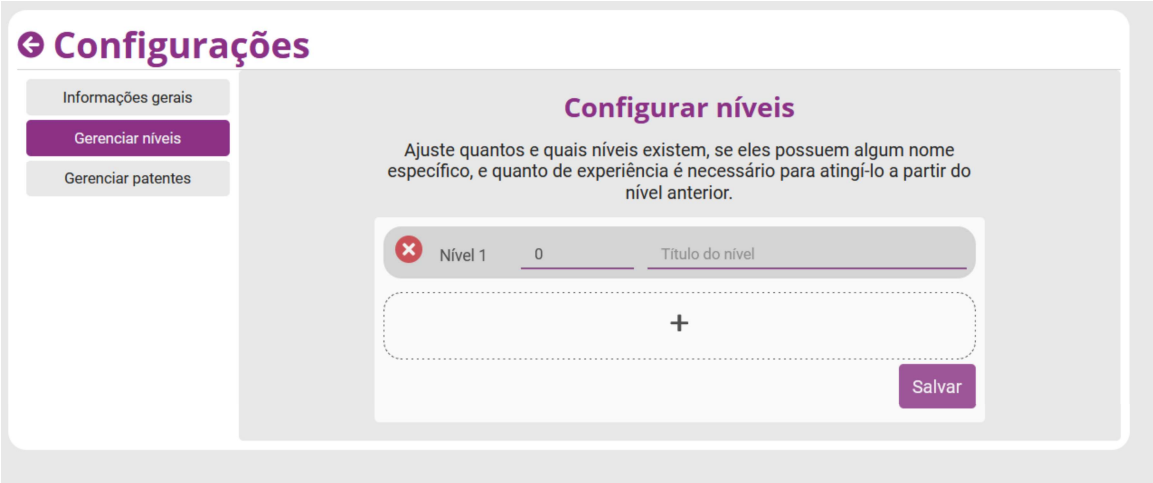


Fonte: Elaborado pelo autor (2021).

Figura 13 – Tela de configuração do jogo

Fonte: Elaborado pelo autor (2021).

Figura 14 – Seção de configuração de níveis



Fonte: Elaborado pelo autor (2021).

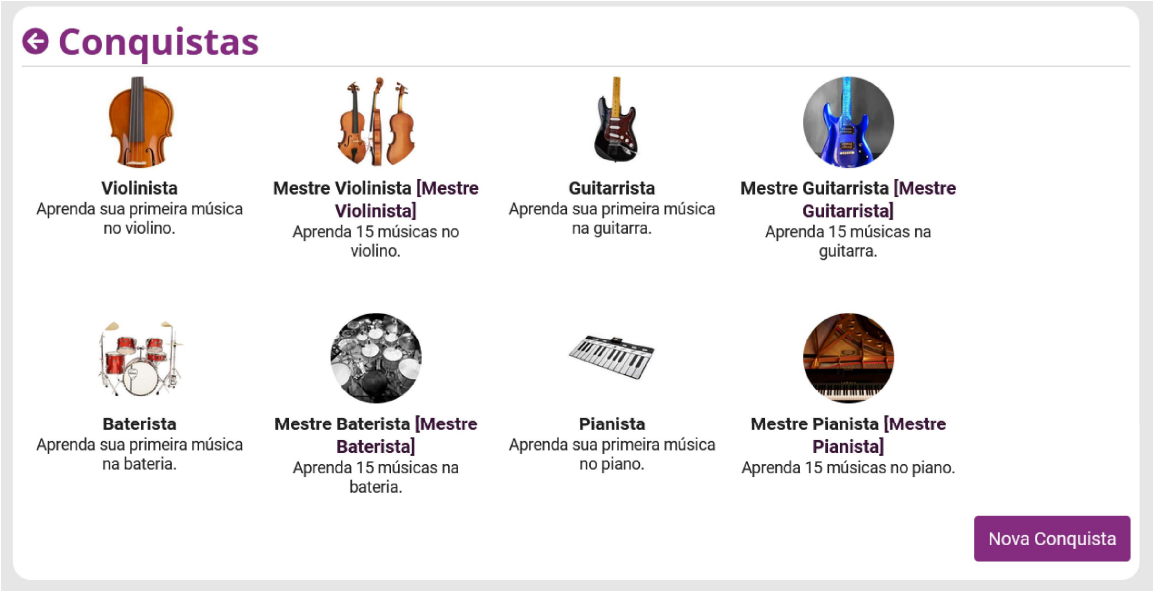
Figura 15 – Seção de configuração de patentes

Fonte: Elaborado pelo autor (2021).

A partir da tela inicial, o usuário também pode acessar a tela de configuração de conquistas. Esta tela lista todas as conquistas do jogo, como mostra a Figura 16. Através do botão **Nova conquista** é possível adicionar uma conquista ao jogo, fornecendo as informações necessárias. Uma conquista também pode ser alterada ou excluída. A Figura 17 mostra o formulário de criação e alteração de uma conquista.

Além de configurar conquistas, o usuário pode acessar o painel de configuração de atividades também a partir da tela inicial. No painel, demonstrado pela Figura 18, o usuário pode visualizar todas as atividades registradas no jogo. O usuário também pode criar, alterar e excluir atividades.

Figura 16 – Tela de configuração de conquistas



Fonte: Elaborado pelo autor (2021).

Figura 17 – Formulário de criação e alteração de conquista

Fonte: Elaborado pelo autor (2021).

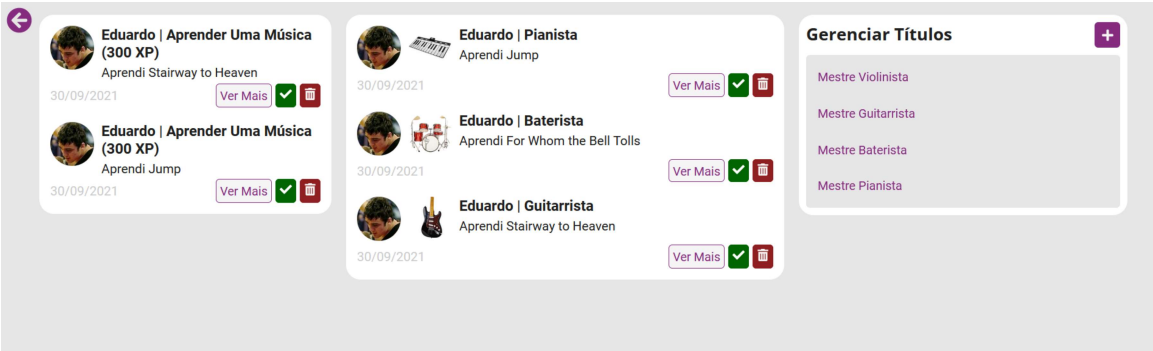
Figura 18 – Tela de configuração de atividades

Fonte: Elaborado pelo autor (2021).

3.7.4 Validação de requisições

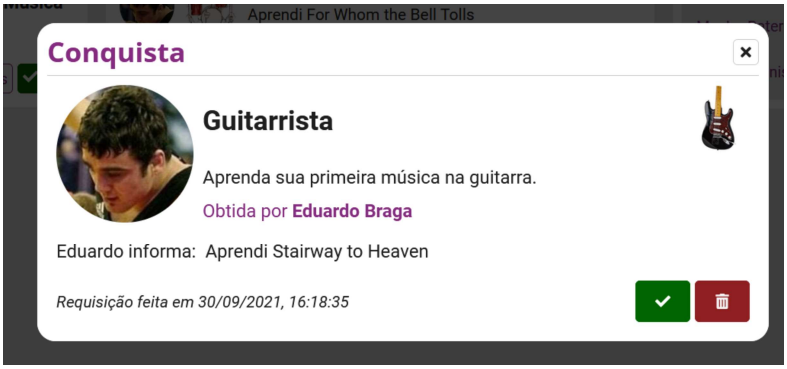
Quando os jogadores requisitam atividades ou conquistas, cabe ao administrador do jogo aprovar ou desaprovar estas requisições. A partir da página inicial, demonstrada na Figura 12, o usuário pode acessar a tela de **Gerência de jogadores**. Nesta tela é possível visualizar todas as requisições para cumprimento de atividades, desbloqueio de conquistas, e as configurações dos títulos existentes no jogo. Um usuário pode aprovar ou excluir uma requisição, e também pode visualizar mais detalhes, como mostra a Figura 20. Além disso, ele pode criar, alterar e remover títulos a partir desta tela.

Figura 19 – Tela de gerência de jogadores



Fonte: Elaborado pelo autor (2021).

Figura 20 – Seção de detalhes de uma requisição



Fonte: Elaborado pelo autor (2021).

3.7.5 Participar de um jogo

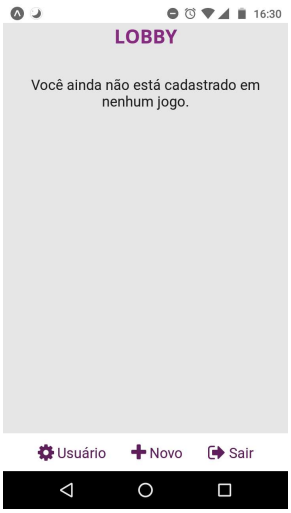
Para participar de um jogo, um usuário deve ter acesso ao código de convite do jogo, ou ao código QR de acesso. Um administrador pode compartilhar estes códigos, que são obtidos através do *lobby* na plataforma *web*. A Figura 21 mostra os códigos que podem ser compartilhados.

Com acesso a pelo menos um código de convite, o jogador deve se autenticar pelo aplicativo, onde então terá acesso ao *lobby*, assim como mostra a Figura 22. Nesta tela, o usuário pode alterar seus dados ao pressionar o botão **Usuário**, ou adicionar um novo jogo ao pressionar o botão **Novo**. Então, basta colar o código de convite ou ler o código QR do jogo para concluir a adição do novo jogo. A Figura 23 mostra a tela onde pode-se informar o código necessário.

Figura 21 – Seção de compartilhamento de jogo da plataforma

Fonte: Elaborado pelo autor (2021).

Figura 22 – Tela de seleção de jogos do aplicativo móvel



Fonte: Elaborado pelo autor (2021).

Figura 23 – Tela de adição de jogo do aplicativo móvel

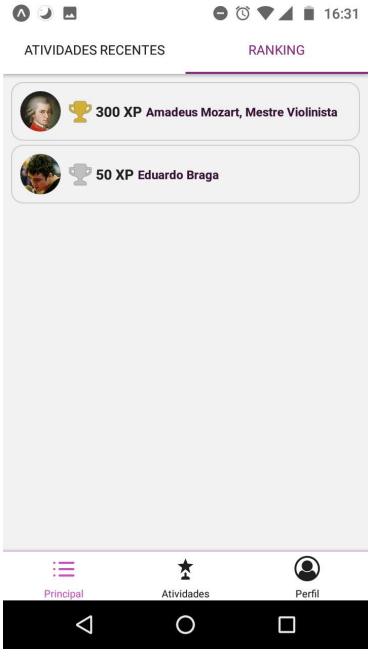
Fonte: Elaborado pelo autor (2021).

Uma vez participando de um jogo, o jogador pode acessar os seus recursos. É possível visualizar a linha do tempo, com todas as atividades realizadas pelos jogadores; o placar de líderes; a seção de atividades, onde pode-se realizar uma requisição de conclusão; e o seu perfil, onde é possível ver o progresso de experiência, selecionar um título e requisitar o desbloqueio de conquistas.

Figura 24 – Linha do tempo do aplicativo móvel

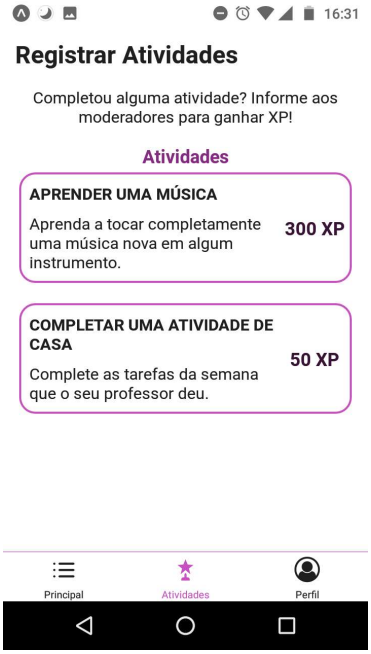
Fonte: Elaborado pelo autor (2021).

Figura 25 – Placar de líderes do aplicativo móvel



Fonte: Elaborado pelo autor (2021).

Figura 26 – Tela de atividades do aplicativo móvel



Fonte: Elaborado pelo autor (2021).

Figura 27 – Perfil do usuário no aplicativo móvel

Fonte: Elaborado pelo autor (2021).

Figura 28 – Conquistas no perfil do usuário

Fonte: Elaborado pelo autor (2021).

4 VALIDAÇÃO

Neste capítulo será abordado o processo de validação da plataforma, desde a distribuição da mesma em ambiente de testes até os *feedbacks* recolhidos de seus usuários de teste.

4.1 DISTRIBUIÇÃO

Após finalizada, é necessário distribuir a aplicação para que usuários possam utilizá-la e testá-la. Várias alternativas de distribuição estão disponíveis no mercado, muitas delas gratuitas. Plataformas conhecidas que fornecem este serviço são Digital Ocean, Heroku, Vercel, entre outras. Para o GameTask, foram escolhidos o Heroku e o Expo.

4.1.1 Plataforma Web e Servidor

O Heroku é uma plataforma de nuvem usada para hospedar aplicações de forma automatizada (HEROKU, 2021). Ela possui suporte para criar um fluxo de distribuição de aplicativos Node.js e React completamente automatizado. Os custos do Heroku são altos, mas em seu plano gratuito a plataforma permite hospedar até 5 aplicativos com os recursos mínimos, o que é o suficiente para a hospedagem tanto do servidor quanto da aplicação *web* do GameTask.

O processo de hospedagem funciona da seguinte forma:

1. É necessário fazer o *upload* do código fonte para alguma plataforma de versionamento. No caso do GameTask, todo o código está disponível em repositórios privados do Github, que é uma das mais usadas plataformas para versionar e compartilhar código atualmente;

2. Criar uma conta no Heroku[1] e efetuar login;

3. No *dashboard* da plataforma, cria-se uma nova aplicação, fornecendo um nome para identificá-la;

4. Na aba de configurações da aplicação, é possível definir as variáveis de ambiente do sistema. Variáveis de ambiente são valores injetados pelo sistema operacional em uma aplicação, permanecendo fora de seu código fonte. Este processo é utilizado para manter valores sensíveis fora do código compartilhado, como *tokens* de autenticação de serviços externos, segredos de criptografia e outras informações;

5. Na aba de configurações, também é possível configurar o processo de distribuição da aplicação, chamado *buildpack*. O Heroku possui um motor nativo para Node.js, que

[1] Pode ser feito através do link https://signup.heroku.com/

foi usado para a distribuição do servidor. Para a distribuição da plataforma *web*, foi utilizado um *buildpack* desenvolvido pela comunidade chamado *Create React App Buildpack*[2];

6. Na aba de Distribuição, é possível conectar sua conta do Heroku com sua conta do Github. Desta forma, é possível selecionar o repositório com o código fonte da aplicação, e o Heroku se encarregará de baixar o código, efetuar a compilação do mesmo e distribuí-lo em uma URL pública.

Após o processo de distribuição, tanto a aplicação *web* quanto o servidor estão disponíveis para acesso público. Existem algumas limitações quanto a distribuição no serviço gratuito do Heroku, entre elas:

- O serviço é desligado automaticamente após um período de inatividade, ou seja, um período sem acessos;
- Ao receber um acesso enquanto inativo, o Heroku precisa montar uma máquina virtual para abrigar o serviço e ativá-lo. Este processo demora alguns segundos, gerando um tempo de espera considerável sempre que a plataforma for acessada pela primeira vez em algumas horas.

Entretanto, estas limitações não afetam o uso pretendido da plataforma.

4.1.2 Aplicativo Móvel

O processo de distribuição padrão de um aplicativo móvel consiste na compilação do seu código em arquivos executáveis do tipo APK para Android ou IPA para iOS. Para efetuar a compilação para Android, é necessário o *Software Development Kit* (SDK) do Android, obtido geralmente através da plataforma Android Studio. Já para a compilação para iOS, é necessário possuir o SDK disponível pela plataforma XCode, a plataforma de desenvolvimento de aplicativos iOS, que pode ser utilizada apenas em sistemas operacionais da Apple, o que é uma grande limitação. Após ter os arquivos executáveis, é necessário enviá-los para as respectivas lojas dos sistemas operacionais, a App Store do iOS ou a Google Play Store do Android, onde os aplicativos podem finalmente ser baixados. Os processos possuem altos custos e podem demorar algum tempo, visto que as empresas fazem uma extensa validação das aplicações. Para apenas testar a aplicação, é necessário um processo alternativo.

A alternativa escolhida foi a publicação da aplicação através do próprio Expo, a plataforma escolhida para desenvolver o aplicativo. O Expo proporciona a opção de publicar o código da aplicação em seu repositório público, onde ela fica acessível para ser

[2] Disponível publicamente em https://github.com/mars/create-react-app-buildpack

baixada através do aplicativo Expo Go, o mesmo utilizado durante a etapa de desenvolvimento. Desta forma, usuários de teste podem baixar o Expo Go e acessar o aplicativo do GameTask através dele, diminuindo o tempo necessário para publicar a aplicação. Uma outra vantagem desta abordagem para a fase de testes é que atualizações publicadas através do Expo são automaticamente baixadas pelos usuários nas próximas vezes que estes executarem o aplicativo, permitindo que os usuários sempre utilizem a última versão do GameTask disponível.

4.2 TESTES

Inicialmente, planejou-se testar a aplicação em um ambiente real, criando processos gamificados em salas de aula ou no dia a dia de empresas, medindo o progresso dos processos, o engajamento dos colaboradores e a usabilidade da plataforma. Entretanto, com a pandemia de COVID-19, estes esforços seriam muito custosos, e então uma forma alternativa de testar a aplicação foi concebida. Esta nova forma, enfim, busca realizar apenas testes de usabilidade da plataforma.

Para efetuar os testes na aplicação, potenciais usuários foram convidados a partir de grupos de estudantes universitários em redes sociais como o Slack e o Whatsapp. Foram escolhidos grupos onde os participantes pudessem ter conhecimento de jogos ou de gamificação, de forma a simular grosseiramente um público alvo real. Uma página web foi criada com o propósito de instruí-los no processo de testagem, providenciando um formulário para ser respondido com perguntas avaliando a usabilidade do GameTask e pedindo sugestões de melhorias. A pesquisa ficou ativa por 6 dias, e obteve 13 respostas. Na página de instruções, os usuários foram orientados a realizar as seguintes ações:

1. Acessar a plataforma *web*, criar uma conta se autenticar;
2. Criar um jogo;
3. Criar uma conquista;
4. Criar uma atividade;
5. Configurar os níveis do jogo;
6. Baixar o aplicativo do GameTask;
7. Autenticar-se no aplicativo;
8. Participar do jogo criado;
9. Concluir uma atividade.

O objetivo destes passos é medir a facilidade que os usuários possuem em navegar pelos fluxos da aplicação. Esta facilidade se dá através do *design* das interfaces do sistema e na experiência de usuário que a plataforma providencia. Além destes passos objetivos, também foi recomendado que os usuários testassem outras funcionalidades da plataforma, afim de descobrirem fluxos de seu interesse ou problemas de usabilidade.

Após concluir os passos, a página direciona o usuário a um formulário anônimo, onde ele deve responder a uma série de perguntas sobre sua experiência utilizando o GameTask.

Quadro 1 – Perguntas de validação do GameTask.

Pergunta	**Tipo da resposta**.
Quantos anos você tem?	Menos de 18 anos; 19-21; 22-25; 26-29; 30 anos ou mais
Você sabe o que é gamificação?	Sim; não
Quão fácil e intuitivo foi o processo de criação da sua conta?	Escala 1-5
Quão fácil e intuitivo foi o processo de criação de um jogo?	Escala 1-5
Quão fácil e intuitivo foi o processo de criação de uma conquista?	Escala 1-5
Quão fácil e intuitivo foi o processo de criação de uma atividade?	Escala 1-5
Quão fácil e intuitivo foi encontrar as opções para configurar os níveis do jogo?	Escala 1-5
Se possível, deixe seu feedback sobre o que achou mais interessante na plataforma, ou o que você acha que faltou e deveria melhorar	Descritiva
Quão fácil e intuitivo foi o processo instalação do aplicativo?	Escala 1-5
Quão fácil e intuitivo foi entrar em um jogo?	Escala 1-5
Quão fácil e intuitivo foi o processo de concluir uma atividade?	Escala 1-5
Se possível, deixe seu feedback sobre o que achou mais interessante no aplicativo, ou o que você acha que faltou e deveria melhorar	Descritiva

Fonte: Elaborado pelo autor.

A pesquisa foi respondida em sua maioria por pessoas de 19 a 21 anos, e todas responderam saber o que é gamificação. A média dos valores das perguntas objetivas (de escala de 1 a 5) se encontra no Quadro 2.

Com relação as respostas descritivas, houveram muitas respostas elogiando a usabilidade e a facilidade de acessar as funcionalidades descritas no roteiro, mas também foram recebidas muitas críticas quanto a alguns elementos específicos. O elemento mais criticado foi a localização da configuração de níveis e patentes da plataforma, sendo mencionado em

Quadro 2 – Respostas objetivas das avaliações.

Pergunta	**Média**
Quão fácil e intuitivo foi o processo de criação da sua conta?	4,77
Quão fácil e intuitivo foi o processo de criação de um jogo?	4,38
Quão fácil e intuitivo foi o processo de criação de uma conquista?	4,68
Quão fácil e intuitivo foi o processo de criação de uma atividade?	4,38
Quão fácil e intuitivo foi encontrar as opções para configurar os níveis do jogo?	2,92
Quão fácil e intuitivo foi o processo instalação do aplicativo?	4,54
Quão fácil e intuitivo foi entrar em um jogo?	3,69
Quão fácil e intuitivo foi o processo de concluir uma atividade?	4,46
Avaliação média	4,23

Fonte: Elaborado pelo autor.

5 das 12 respostas recebidas. Outro ponto levantado com certa frequência para a aplicação *web* foi a dificuldade de utilizá-la através de um dispositivo móvel. Apesar de ter sido desenvolvida para uso em computadores e existir uma recomendação para seu uso na plataforma devida, alguns usuários optaram por usá-la através do seu dispositivo móvel, e relataram problemas como dificuldade na visualização de campos de formulário, sendo mencionado em 2 das 12 respostas. Com relação ao aplicativo móvel, a principal crítica foi quanto ao processo de participar de um jogo. 4 entre os 11 usuários que responderam sobre o aplicativo mencionaram dificuldades em encontrar a tela onde poderiam participar do jogo, dificuldades para submeter o código de convite ou ler o código QR do jogo. Isso pode se dar devido a interação entre os dois sistemas, ou pela falta de instruções em tela sobre como utilizar as plataformas.

Durante a pesquisa também foram relatados alguns erros na plataforma, que foram prontamente resolvidos. Importante ressaltar que a quantidade e a qualidade das amostras da pesquisa não são suficientes para sustentar argumentos de qualidade da plataforma, principalmente devido ao possível enviesamento causado pelo fato dos usuários teste pertencerem a grupos muito similares de indivíduos (em sua maioria, graduandos em tecnologia). Entretanto, a pesquisa serve como um vislumbre inicial dos acertos e erros do desenvolvimento, assim como potenciais caminhos futuros a serem percorridos. As respostas estão disponíveis no Apêndice B.

5 CONCLUSÃO

Após o estudo de elementos de gamificação, implementações do sistema em outros trabalhos e também no mercado, foi desenvolvida a plataforma de gamificação GameTask, que fornece aos seus usuários a possibilidade de criar um processo gamificado e participar do mesmo de forma virtual através de um dispositivo móvel. A plataforma foi desenvolvida utilizando padrões e práticas valorizadas e conhecidas no mercado, dentre elas a Arquitetura Limpa e os Princípios S.O.L.I.D.

Os princípios de *design* utilizados foram demonstrados em exemplos de código e nas explicações conceituais das camadas de arquitetura do sistema, de forma que interessados em desenvolver suas próprias plataformas possam se inspirar na arquitetura implementada e realizar sua própria implementação. O foco na explicação dos conceitos utilizados, em vez de apenas mostrar os códigos feitos, incentiva os leitores a estudarem e descobrirem suas próprias implementações independente de linguagem e *frameworks*, extendendo o alcance do trabalho.

As práticas de arquitetura e princípios de *design* implementadas mostraram seu valor durante as etapas de desenvolvimento e validação. Com a separação de responsabilidade em camadas, a plataforma pôde ser desenvolvida com uma abordagem *top-down*, ou seja, com a finalização da camada mais acima antes da implementação das camadas mais abaixo. Isso possibilitou a identificação de problemas de domínio ou necessidades de outras entidades e casos de uso antes que escolhas de bibliotecas, banco de dados ou *frameworks* fosse necessária. A aplicação dos princípios S.O.L.I.D, especialmente os princípios de Inversão de Dependências e Responsabilidade Única, foram fundamentais para a agilidade no desenvolvimento, de forma que muitos algoritmos puderam ser levados de um sistema ao outro sem necessidade de alteração. Casos de uso como a autenticação de usuário, que envolve a manipulação de tokens JWT, puderam ser implementados da mesma maneira tanto na plataforma *web* quanto no aplicativo móvel por dependerem apenas de abstrações dos provedores de token, sendo que na prática bibliotecas diferentes foram utilizadas em cada ambiente. Durante a validação, os erros e problemas de mal funcionamento relatados pelos usuários puderam ser rapidamente resolvidos pela facilidade de encontrar o fluxo onde o problema ocorria. A separação de responsabilidades por camadas permitiu identificar a natureza dos problemas, sejam problemas de interface do usuário, de comunicação entre sistemas ou de regras de domínio.

A plataforma desenvolvida também atendeu as expectativas, recebendo uma boa avaliação de seus usuários de teste, se provando um protótipo viável e um potencial produto de mercado. As tecnologias utilizadas se mostraram eficientes tanto no processo inicial de desenvolvimento quanto no processo de correção de erros. Nota-se entretanto que a eficiência das tecnologias utilizadas se dá especificamente ao caso de uso do projeto, onde a plataforma deveria ser desenvolvida rapidamente, e portanto uma única linguagem e

banco de dados facilmente modificável foram características imprescindíveis para o sucesso do desenvolvimento. Sabe-se que em cenários de maior escalabilidade estas escolhas podem não ser ótimas.

Por fim, foi desenvolvida uma plataforma escalável, cuja arquitetura permite que novas funcionalidades e recursos sejam adicionados rapidamente e com facilidade. A plataforma admite a implementação de processos gamificados com diversos elementos de jogos, o que permite que pesquisadores e interessados criem seus próprios jogos e possam experienciar a gamificação imediatamente. Além disso, os conceitos abordados podem ser utilizados pelos mesmos para a construção de suas próprias plataformas com seus próprios graus de complexidade, visto que estas práticas e metodologias visam a criação de sistemas complexos e grandes.

5.1 TRABALHOS FUTUROS

Dado o caráter prático do trabalho e as dificuldades encontradas para sua validação devido à pandemia de COVID-19, existe a possibilidade de melhoria de funcionalidades e interface, além de estudos mais aprofundados utilizando a ferramenta.

5.1.1 Testar a ferramenta em ambientes reais

Devido a pandemia, aplicar o GameTask em ambientes reais, seja em empresas ou escolas, se tornou uma opção muito custosa e impraticável. Entretanto, com o cenário de vacinação mais claro e a perspectiva de retorno a normalidade a partir de 2022, se torna viável a utilização da plataforma para estudos sobre gamificação.

Pretende-se que o GameTask continue disponível para utilização gratuita, e portanto pode-se utilizar a ferramenta para aplicar processos gamificados em salas de aula ou times de empresas para avaliar o impacto da gamificação nestes cenários. Especialmente interessante pode ser o acompanhamento de progresso de grupos cujos processos se tornaram gamificados, em comparação com grupos de controle onde os processos continuam de forma convencional, afim de comparar os dados e obter conclusões mais assertivas sobre a eficiência da gamificação.

5.1.2 Melhorias de interface

Como a ferramenta foi desenvolvida como prótótipo, existe muito espaço para a melhoria de interface do GameTask. Um maior estudo de práticas de UI/UX pode incrementar a experiência dos usuários, tornando a identidade visual da plataforma mais atrativa e intuitiva. Especialmente, a adoção de sugestões já dadas no processo de validação, como facilitar o acesso às configurações do jogo ou clarificar o processo de entrada em um jogo podem melhorar consideravelmente a experiência que o usuário tem ao utilizar o GameTask.

5.1.3 Publicação em ambiente de mercado

Caso a plataforma se torne bem aceita e seu valor seja reconhecido, é possível ainda que o GameTask venha a ser publicado em mercado. Os passos para tal incluem a adquirência de um domínio, configuração de distribuição e disponibilidade do aplicativo nas lojas oficiais de *smartphones*.

REFERÊNCIAS

ALSAWAIER, Raed S. The Effect of Gamification on Motivation and Engagement. **The International Journal of Information and Learning Technology**, 2017. DOI: https://doi.org/10.1108/IJILT-02-2017-0009.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. 4. ed. [*S.l.*]: Addison-Wesley Professional, dez. 1997.

DETERDING, Sebastian *et al.* Gamification: using game-design elements in non-gaming contexts. **Conference on Human Factors in Computing Systems**, p. 2425–2428, mai. 2011. DOI: http://dx.doi.org/10.1145/1979742.1979575.

EVANS, Eric. **Domain-Driven Design**: Tackling Complexity in the Heart of Software. [*S.l.*]: Addison Wesley, 2003.

FERNANDES, Diego. **Expo**: o que é, para que serve e quando utilizar? 2018. Disponível em: https://blog.rocketseat.com.br/expo-react-native/. Acesso em: 19 ago. 2021.

GREER, Derek. **Exploring TypeScript**. Ago. 2016. Disponível em: https://lostechies.com/derekgreer/2016/08/30/exploring-typescript/. Acesso em: 19 ago. 2021.

HANUS, Michael D.; FOX, Jesse. Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. **Computers & Education**, v. 80, p. 152–161, 2015. DOI: https://doi.org/10.1016/j.compedu.2014.08.019.

HEROKU. **What is Heroku**. 2021. Disponível em: https://www.heroku.com/what. Acesso em: 12 set. 2021.

HITCHENS, Michael; TULLOCH, Rowan. A gamification design for the classroom. **Interactive Technology and Smart Education**, v. 15, p. 28–45, mar. 2018. DOI: https://doi.org/10.1108/ITSE-05-2017-0028.

JONES, Brooke A.; MADDEN, Gregory J.; WENGREEN, Heidi J. The FIT Game: preliminary evaluation of a gamification approach to increasing fruit and vegetable consumption in school. **Preventive Medicine**, 2014. DOI: http://dx.doi.org/10.1016/j.ypmed.2014.04.015.

JURGELAITIS, Mantas *et al.* Implementing gamification in a university-level UML modeling course: A case study. **Computer Applications in Engineering Education**, out. 2018. DOI: 10.1002/cae.22077.

KAPP, Karl M. **The gamification of learning and instruction fieldbook**: Ideas into practice. [*S.l.*]: Pfeiffer, 2013.

KORN, Oliver; SCHMIDT, Albrecht. Gamification of Business Processes: Re-designing Work in Production and Service Industry. **Procedia Manufacturing**, v. 3, p. 3424–3431, out. 2015. DOI: `https://doi.org/10.1016/j.promfg.2015.07.616`.

KUO, Ming-Shiou *et al.* What Are the Better Gamification Tools for Elementary School Teachers? **7th International Congress on Advanced Applied Informatics**, 2018. DOI: `10.1109/IIAI-AAI.2018.00074`.

MARTIN, Robert C. **Clean Architecture**: A craftsman's guide to software structure and design. [*S.l.*]: Prentice Hall, 2017.

MDN, Web Docs. **O que é JavaScript?** 2021. Disponível em: `https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript`. Acesso em: 19 ago. 2021.

MONGODB. **O que é MongoDB?** Disponível em: `https://www.mongodb.com/pt-br/what-is-mongodb`. Acesso em: 19 ago. 2021.

NODE.JS. **About Node.js**. Disponível em: `https://nodejs.org/en/about/`. Acesso em: 19 ago. 2021.

OCCHINO, Tom. **React Native**: Bringing modern web techniques to mobile. 2015. Disponível em: `https://engineering.fb.com/2015/03/26/android/react-native-bringing-modern-web-techniques-to-mobile/`. Acesso em: 19 ago. 2021.

SANCHEZ, Eric; YOUNG, Shawn; JOUNEAU-SION, Caroline. Classcraft: from gamification to ludicization of classroom management. **Springer Science**, 2016. DOI: `https://link.springer.com/article/10.1007/s10639-016-9489-6`.

SEABORN, Katie; FELS, Deborah I. Gamification in Theory and Action: A Survey. **Int. J. Human Computer Studies**, v. 74, p. 14–31, 2015. DOI: `http://dx.doi.org/10.1016/j.ijhcs.2014.09.006`.

SHAHZAD, Farrukh. Modern and Responsive Mobile-enabled Web Applications. **Procedia Computer Science**, v. 110, p. 410–415, 2017. DOI: `https://doi.org/10.1016/j.procs.2017.06.105`.

SHVETS, Alexander. **Dive Into Design Patterns**. [*S.l.*: *s.n.*], 2020.

SMITH, Patrick. **Client/server computing**. 2. ed. [*S.l.*]: Sams Publishing, 1994.

SUBRAMANIAN, Vasan. **Pro MERN Stack**: Full Stack Web App Development with Mongo, Express, React, and Node. [*S.l.*]: Apress, 2019.

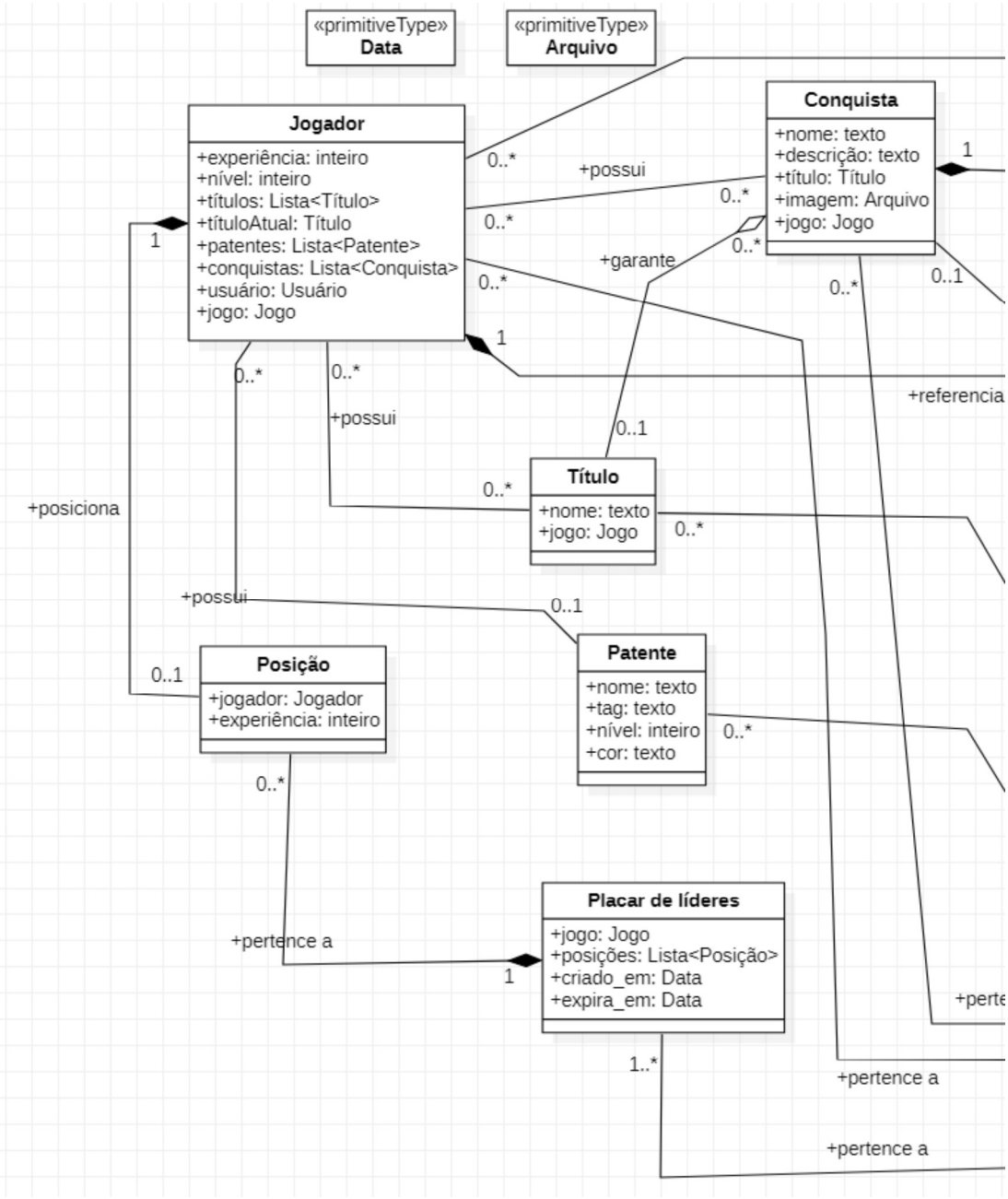
TYPESCRIPT. **TypeScript for the New Programmer**. 2021. Disponível em: `https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html`. Acesso em: 19 ago. 2021.

WOODCOCK, Jamie; JOHNSON, Mark R. Gamification: What it is, and how to fight it. **The Sociological Review**, 2017. DOI: `https://journals.sagepub.com/doi/10.1177/0038026117728620`.

ZHAOA, Shuman; XIAB, Xiaoling; LE, Jiajin. A Real-Time Web Application Solution Based on Node.js and WebSocket. **Advanced Materials Research**, v. 816–817, p. 1111–1115, 2013. DOI: `http://dx.doi.org/10.4028/www.scientific.net/AMR.816-817.1111`.

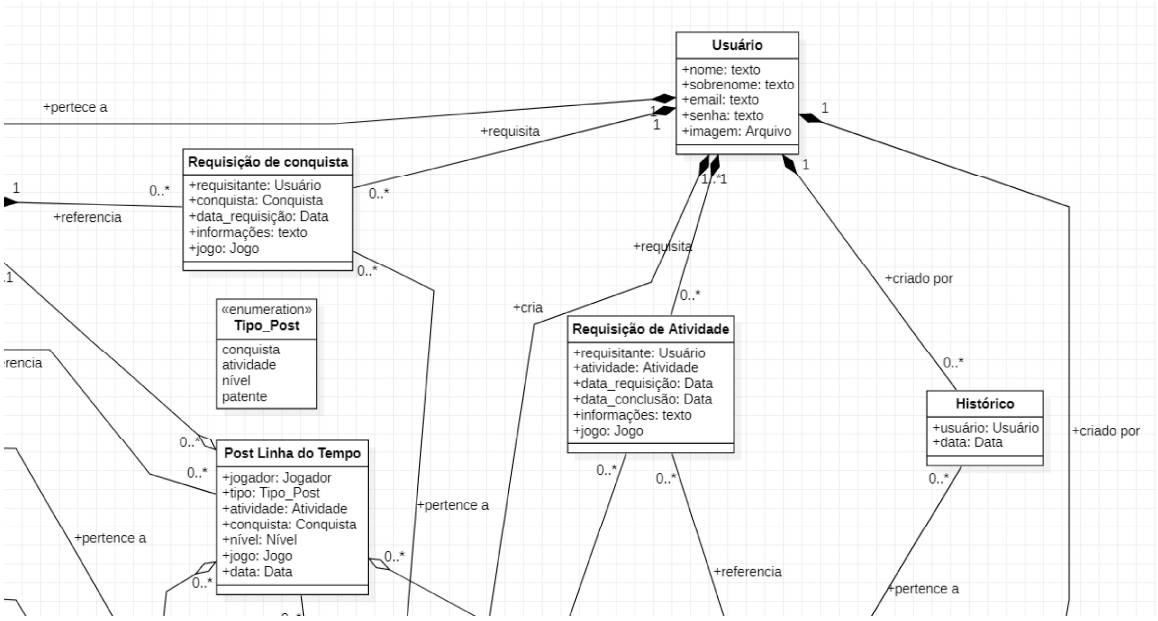
APÊNDICE A – DIAGRAMAS DO DOMÍNIO

Figura 29 – Entidades do sistema: Parte 1



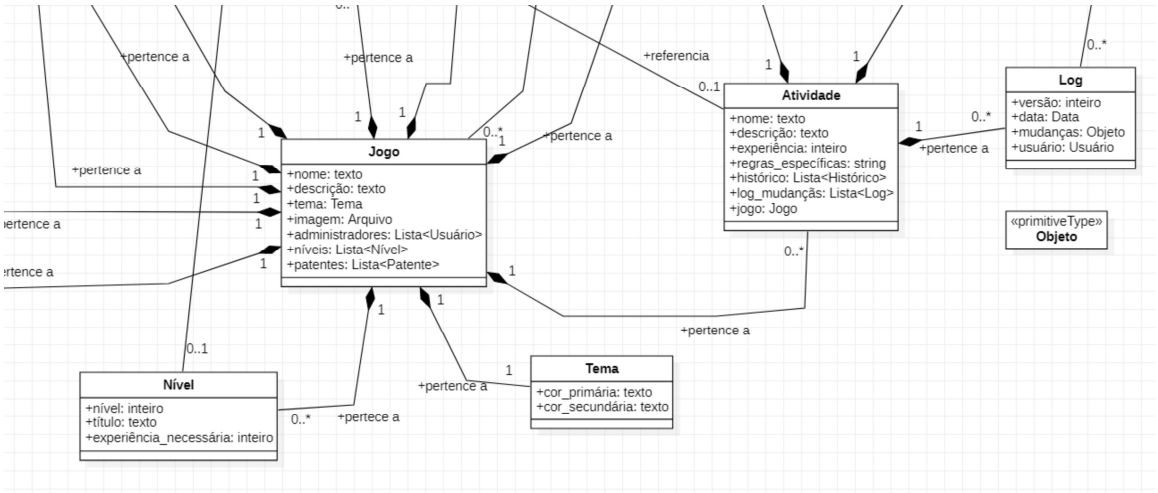
Fonte: Elaborado pelo autor (2021).

Figura 30 – Entidades do sistema: Parte 2



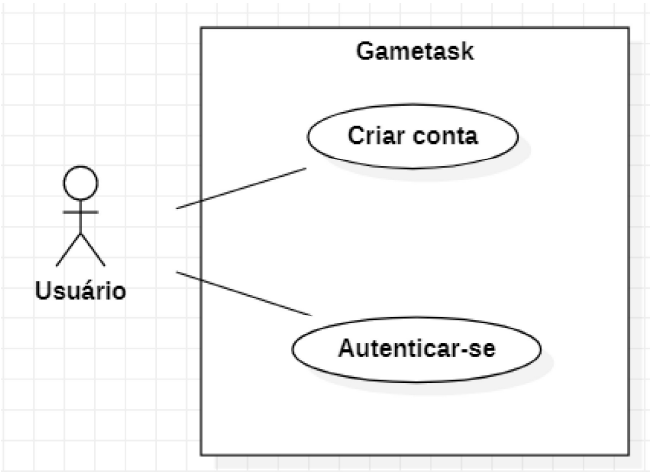
Fonte: Elaborado pelo autor (2021).

Figura 31 – Entidades do sistema: Parte 3



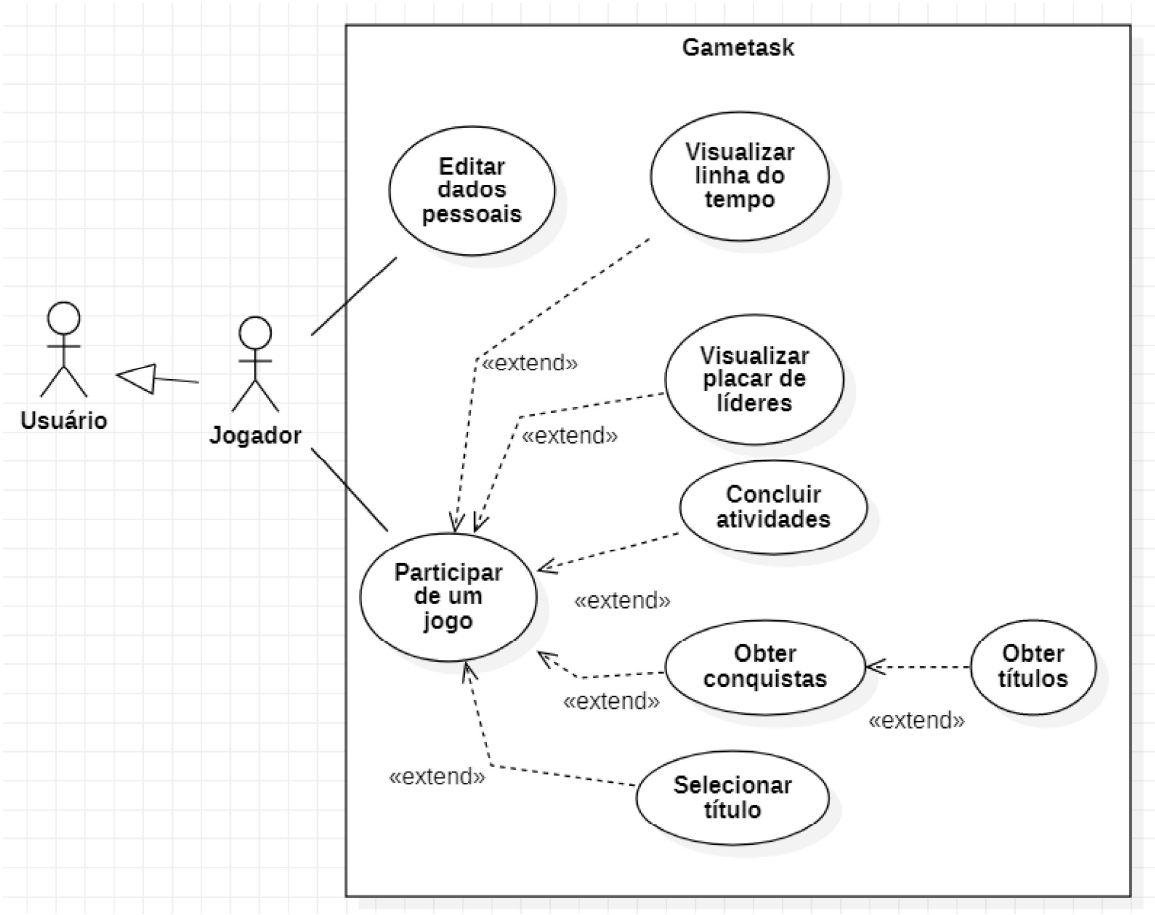
Fonte: Elaborado pelo autor (2021).

Figura 32 – Casos de uso do usuário



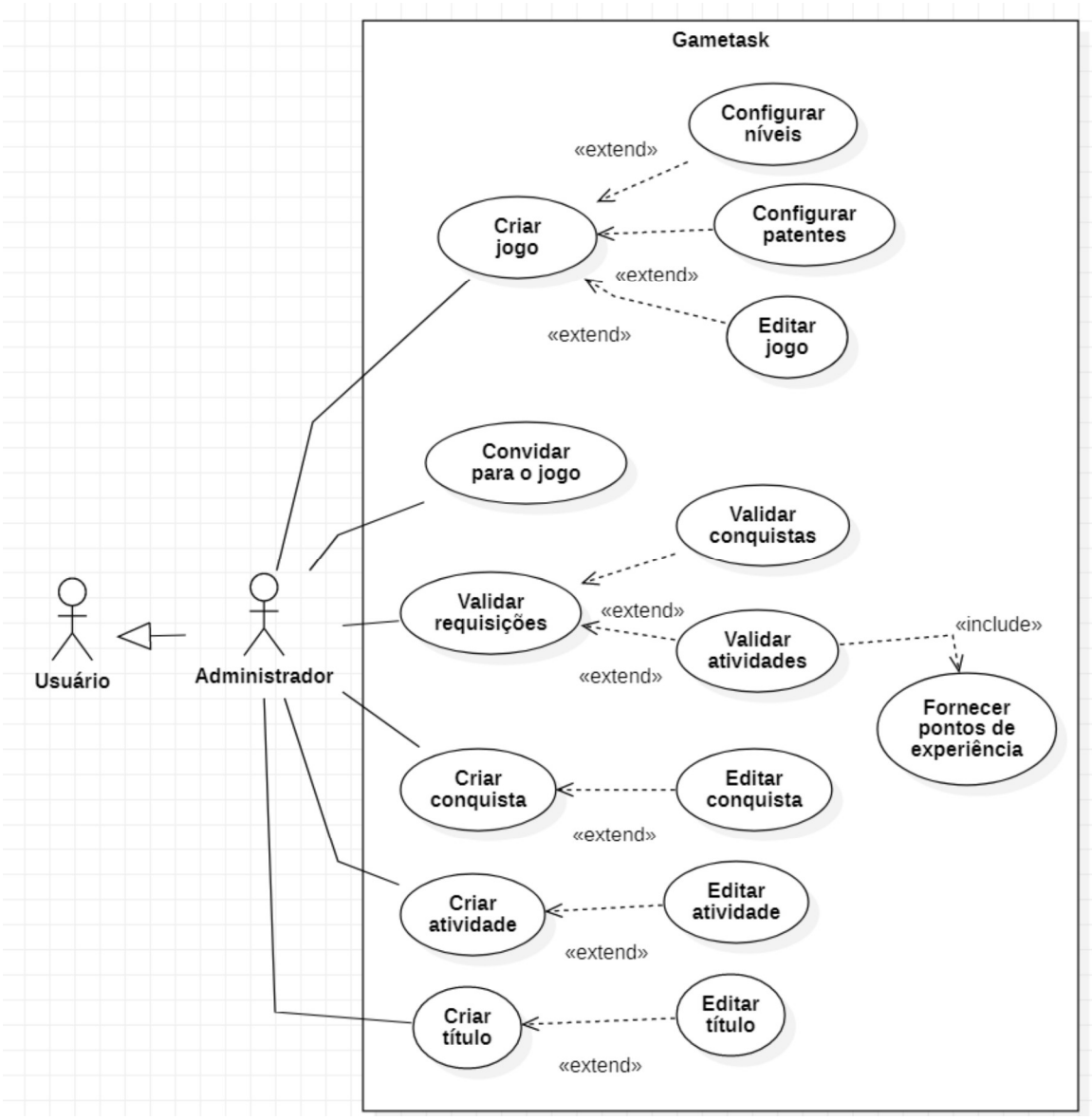
Fonte: Elaborado pelo autor (2021).

Figura 33 – Casos de uso do jogador



Fonte: Elaborado pelo autor (2021).

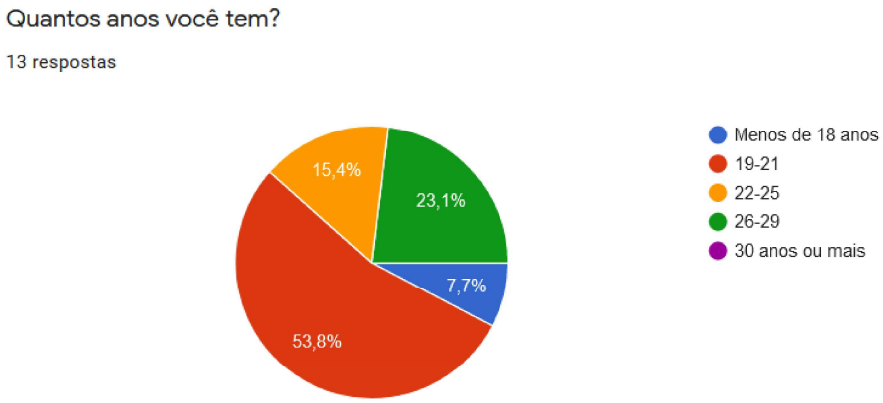
Figura 34 – Casos de uso do administrador



Fonte: Elaborado pelo autor (2021).

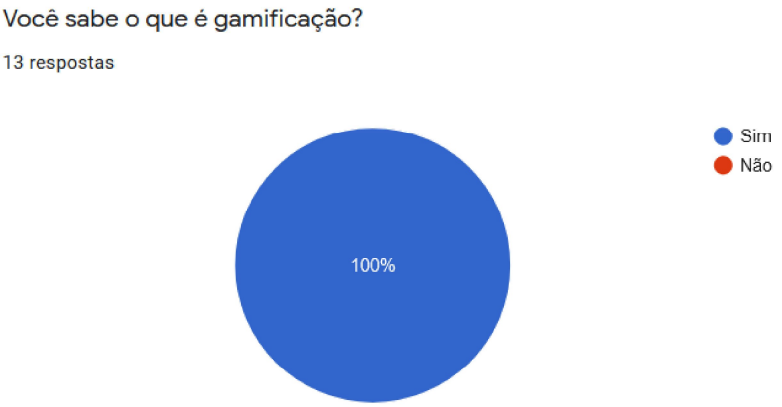
APÊNDICE B – RESPOSTAS DA PESQUISA DE VALIDAÇÃO DO GAMETASK

Figura 35 – Quantos anos você tem?



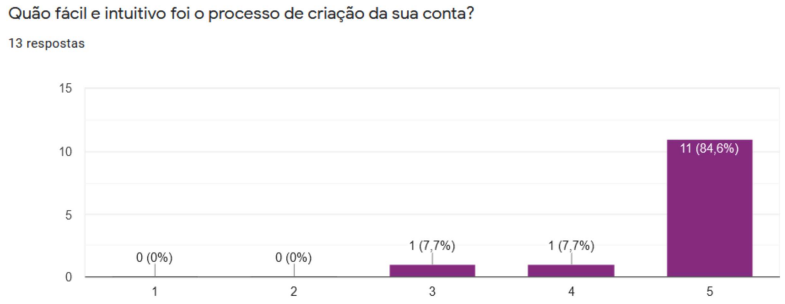
Fonte: Elaborado pelo autor (2021).

Figura 36 – Você sabe o que é gamificação?



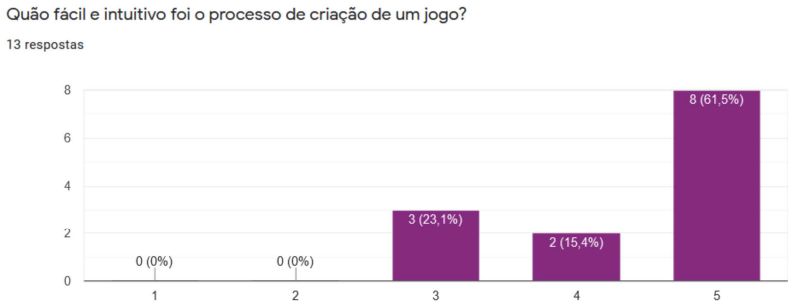
Fonte: Elaborado pelo autor (2021).

Figura 37 – Quão fácil e intuitivo foi o processo de criação da sua conta?



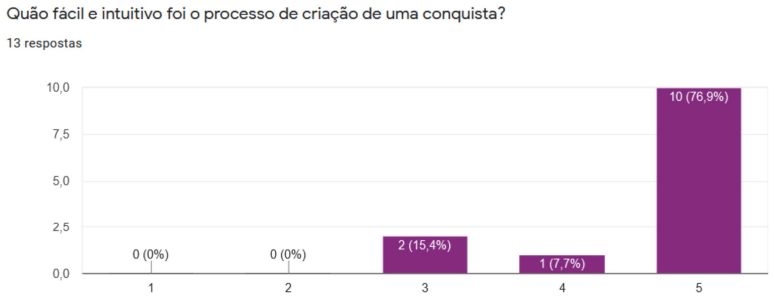
Fonte: Elaborado pelo autor (2021).

Figura 38 – Quão fácil e intuitivo foi o processo de criação de um jogo?



Fonte: Elaborado pelo autor (2021).

Figura 39 – Quão fácil e intuitivo foi o processo de criação de uma conquista?

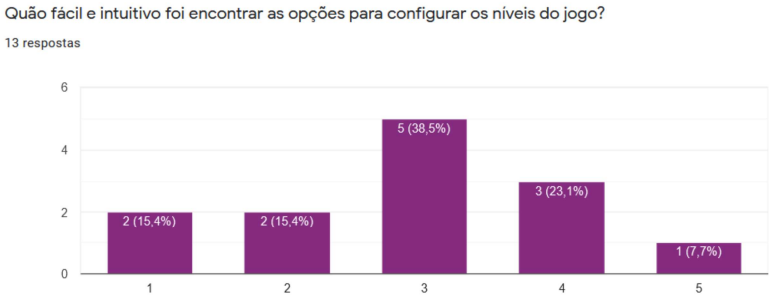


Fonte: Elaborado pelo autor (2021).

Figura 40 – Quão fácil e intuitivo foi o processo de criação de uma atividade?

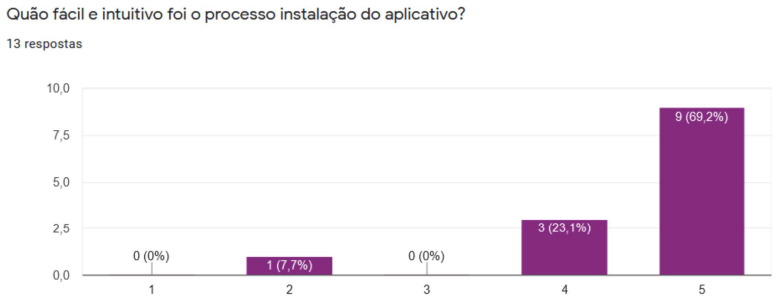
Fonte: Elaborado pelo autor (2021).

Figura 41 – Quão fácil e intuitio foi encontrar as opções para configurar os níveis do jogo?



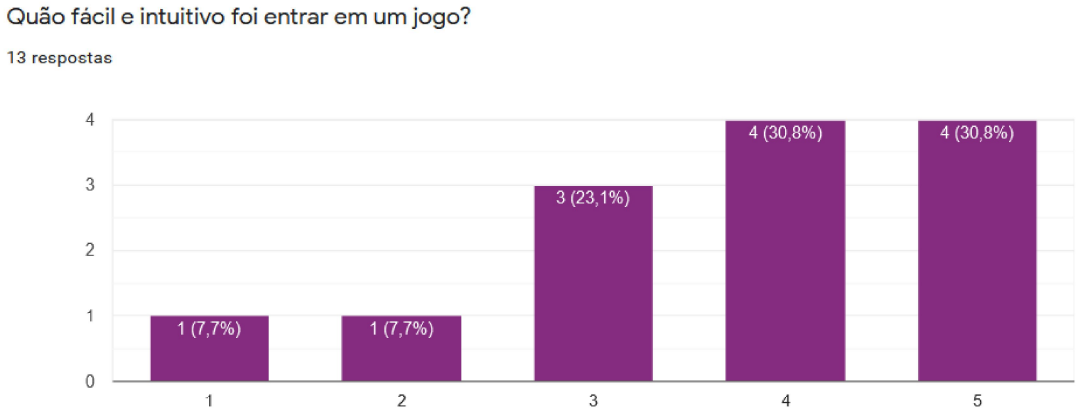
Fonte: Elaborado pelo autor (2021).

Figura 42 – Quão fácil e intuitio foi o processo de instalação do aplicativo?



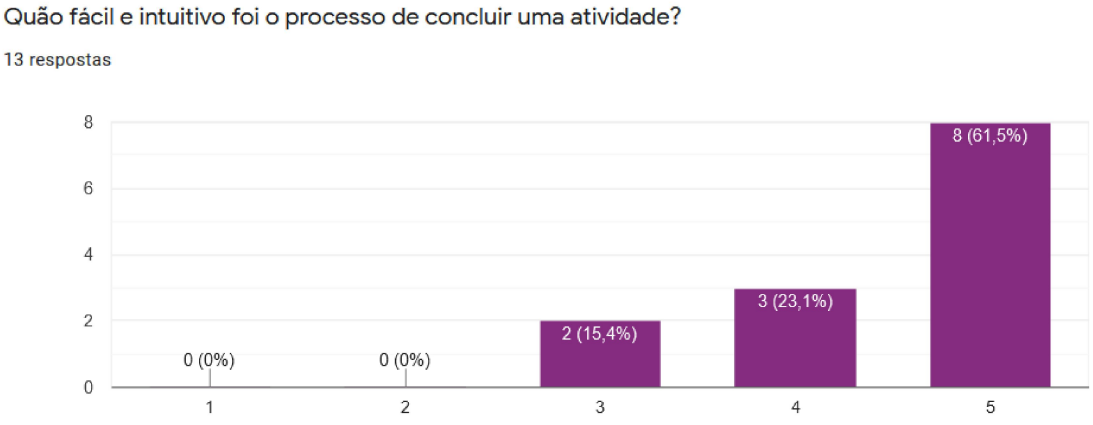
Fonte: Elaborado pelo autor (2021).

Figura 43 – Quão fácil e intuitio foi entrar em um jogo?



Fonte: Elaborado pelo autor (2021).

Figura 44 – Quão fácil e intuitivo foi o processo de concluir uma atividade?

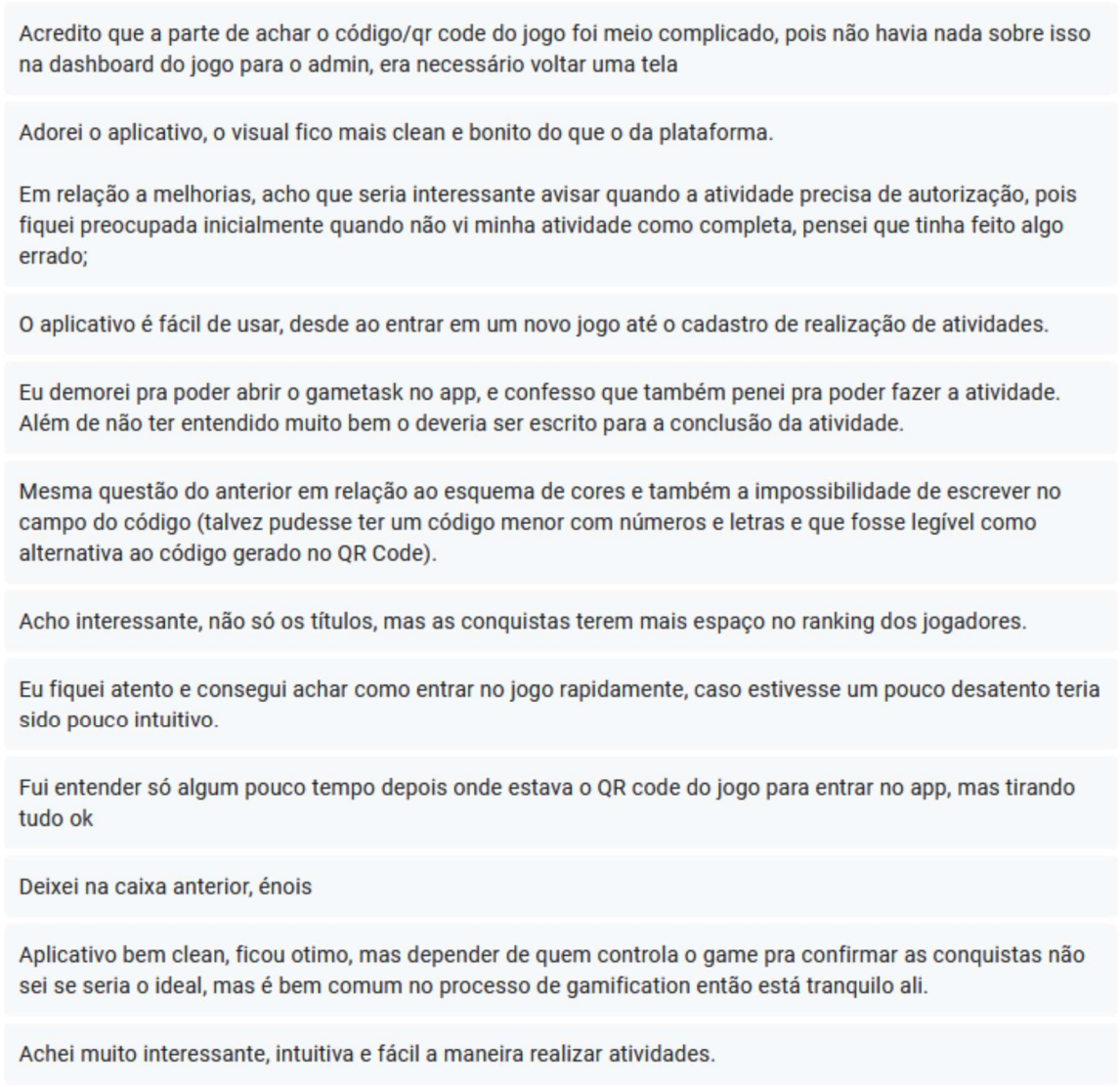


Fonte: Elaborado pelo autor (2021).

Figura 45 – Respostas descritivas sobre a plataforma *web*

Fonte: Elaborado pelo autor (2021).

Figura 46 – Respostas descritivas sobre o aplicativo móvel



Fonte: Elaborado pelo autor (2021).