Este projeto propõe a construção de uma aplicação para auxiliar o desenvolvimento de modo multijogador em jogos digitais, voltado principalmente a jogos educacionais, visando facilitar o desenvolvimento de jogos educacionais e incentivando novas formas de ensino.

Objetivos

Este trabalho tem como objetivo apresentar e desmontrar uma API que comporte a criação de uma plataforma que possa comportar múltiplos jogos educacionais e transformá-los em multijogador em rede de forma simples, armazenando dados das partidas dos alunos para que possam ser consultados e analisados posteriormente pelo professor.

Fundamentação Teórica

Jogos

Jogos digitais

Jogos digitais na educação

API

A sigla API significa Interface de Programação de Aplicações (em inglês, *Application Programming Interface*), sendo um conjunto de rotinas e padrões que permitem a construção e a integração de aplicações garantindo a segurança de dados (MEDEIROS, 2014).

Uma API permite que uma solução ou serviço se comunique com outros de maneira ágil sem a necessidade de conhecer a linguagem ou implementação na qual ela foi feita, apenas as formas de realizar a troca de informações que atenda os requisitos. Ela fornece um conjunto de operações definidas por suas entradas e saídas, e necessariamente permite reimplementações sem comprometer seus usuários.

APIs são disponibilizadas quando uma empresa de *software* tem a intenção de que outros desenvolvedores criem novos produtos associados ao seu serviço ou permita desenvolver sistemas melhores a partir de código reutilizável. Instituições costumam disponibilizar seus códigos acompanhados de uma documentação com instruções de uso para que os usuários possam integrar de maneira simples.

De onde surgiram as APIs?

A ideia de API surgiu muito antes do termo ser cunhado. Para descrever a origem da ideia das APIs é necessário discutir quem inventou as bibliotecas de sub-rotinas, pois não é possível existir uma API sem uma biblioteca de sub-rotinas.

O termo biblioteca de sub-rotinas (em inglês, *subroutine library*) apareceu pela primeira vez na publicação ‘*Planning and Coding of Problems for an Electronic Computing Instrument- Part II, Volume III*’ por Goldstine e Neumann (1948), antes de quaisquer computadores de propósito geral serem construídos, sendo o primeiro relato de um método para criar computadores capazes de armazenar um programa. A publicação possuia como ideia principal que a maioria dos programas utilizariam operações em comum, e que uma biblioteca de sub-rotinas com essas operações diminuiria a quantidade de código necessário para escrever um programa e a quantidade de erros gerados no processo.

A ideia original de Goldstine e Neumann (1948) no entanto era muito difícil de se tornar algo prático, pois necessitaria de uma extensa intervenção do operador de computador para fazer uso dessas sub-rotinas.

Já em 1949, Maurice Wilkes finalizou a construção do primeiro computador capaz de armazenar programas em memória da história, o EDSAC, anterior ao EDVAC, a máquina criada por Goldstine e Von Neumann em 1951. Os primeiros programas executados por ele eram meros testes e não necessitavam do uso de sub-rotinas, utilizavam uma arquitetura simples, onde as 30 primeiras palavras eram para inicialização (comparado a um *boot loader*) e o programa em si era escrito a partir da posição 30, o código era escrito em *assembly* com a ideia de afastar o usuário de lidar com o binário diretamente. O primeiro programa do EDSAC foi escrito por seu estudante de PhD David Wheeler, responsável também por criar o código de inicialização do EDSAC. (BLOCH, 2018)

Ao tentar criar o primeiro programa não trivial, tentando solucionar uma integral numérica da equação diferencial de Airy, Wilkes percebeu que grande parte do tempo gasto para escrever seus programas era buscando erros no seu próprio código, Wilkes contemplou como solução a implementação de bibliotecas de sub-rotinas, tarefa na qual repassou para Wheeler. Wheeler finalizou a criação de sua arquitetura de sub-rotinas em setembro de 1949, onde ele criou pseudo-instruções, destinadas não ao computador, mas sim ao compilador para a realocação de sub-rotinas e passagem de parâmetros, sem necessitar da intervenção de um operador. Com isso, Wheeler foi capaz de reduzir a quantidade e instruções para inicialização.

Sub-rotinas já eram capazes de chamar outras sub-rotinas dentro de si, mas sem permitir recursão. (BLOCH, 2018)

A biblioteca de sub-rotinas de David Wheeler consistia em um gaveteiro armazenando as fitas de papel perfurado que continham as sub-rotinas, cada gaveta era rotulada e possuía um conjunto de sub-rotinas do mesmo tipo, as quais eram copiadas mecânicamente para a fita do programa principal.

Foi então que Wilkes, Wheeler e Gill (1951), publicaram ’*The Preparation of Programs for an Electronic Digital Computer*’, o primeiro livro sobre programação de computadores do mundo, no qual apresentou ao mundo a primeira biblioteca de código reutilizável, a primeira API meticulosamente documentada e outros conceitos, se tornando uma das principais referências na escrita de programas até o surgimento de linguagens de alto nível. Em 1952, WHEELER apresentou um artigo com ideias chaves sobre a criação de

bibliotecas durante a conferência nacional da ACM:

A preparação de uma biblioteca de sub-rotinas exige um trabalho considerável, é muito maior do que o esforço necessário para codificar a sub-rotina em sua forma mais simples. Normalmente é necessário codificá-la seguindo os padrões da biblioteca, podendo reduzir sua eficiência de tempo e espaço.

Pode ser desejável codificá-la de tal maneira que a operação seja generalizada para algum propósito. Contudo, após ter sido codificada e testada, ainda resta a considerável tarefa de escrever uma descrição para que pessoas que não conhecem o interior do código possam utilizá-lo facilmente. Esta última

tarefa pode ser a mais difícil. (WHEELER, 1952, tradução nossa)

O termo *Application Program Interface* só foi cunhado a partir do artigo ’*Data structures and techniques for remote computer graphics*’, quando os autores Cotton e Greatorex (1952), dissertavam sobre a interação entre uma aplicação e o computador em si. Uma interface de aplicação consistente poderia ser mantida mesmo se o computador em si fosse substituído por outro, um sistema flexível e independente de hardware garantiria que a tecnologia avançasse sem tornar o sistema obsoleto, visto que na época para cada novo computador criado era preciso reimplementar as bibliotecas pré existentes e, com novos algoritmos, melhorar a performance de aplicações criadas.

Posteriormente, APIs invadiram múltiplos campos da computação, podendo citar como alguns exemplos a biblioteca padrão utilizada pela linguagem C, chamadas de sistemas em sistemas operacionais, interfaces de linhas de comando, Java, até a criação de *web* APIs.

O conceito de *web* APIs moderno surgiu a partir do ano 2000, o termo API REST apareceu na tese de doutorado, ’*Architectural Styles and the Design of Network-based Software Architectures* de Roy Fieldings descrevendo os primeiros conceitos sobre esse tipo de API. A primeira *web* API a surgir foi criada pela *Salesforce*, publicada em 7 de fevereiro de 2000 no evento IDG Demo, oferecendo processos de automação para clientes seguindo um modelo de Internet como um serviço (*IaaS*). Em seguida, empresas como a *eBay* e *Amazon* disponibilizaram suas próprias *web* APIs (LANE, 2019).

Após 2004, uma nova seleção de APIs surgiram e começaram a se popularizar, sem um valor comercial diretamente associado, eram caracterizadas pelo compartilhamento de informações entre pessoas, fotos e outros dados mais sociais. Pode-se citar como principais APIs, o *Delicious*, *Flickr*, *Facebook* e *Twitter*.

Já em 2006, a *Amazon* disponibilizou ao mundo sua API focada em oferecer serviços de armazenamento de dados de baixo custo, seguindo o modelo de pagamento conforme o uso, o *Amazon Simple Storage* (S3). Seis meses após o lançamento do S3, a empresa disponibilizou um novo serviço chamada de *Amazon Elastic Compute* (EC2), provendo servidores nos quais desenvolvedores poderiam realizar a implantação da infraestrutura de novas aplicações e empresas de tecnologia.

Ambos os serviços integram a plataforma *Amazon Web Services* (AWS) e foram a primeira aparição comercial da computação em nuvem, popularizando o modelo e revolucionando o funcionamento de APIs e da computação como um todo.

Classificações de APIs

Ciente de que APIs existem de diversas formas e propósitos, é interessante definir classificações possíveis a elas. A primeira forma de classificação a ser abordada é pelo mecanismo de transporte, se é baseado em hipertexto, código-fonte ou operações binárias:

a) APIs de serviços *web* - APIs baseadas em hipertexto tipicamente utilizando padrões de comunicação como REST, SOAP, JSON-RPC ou XML-RPC. Costumam ser utilizadas na construção de aplicações *web* como parte da arquitetura orientada a serviços (SOA);

b) APIs de código-fonte - Oferecem bibliotecas de objetos, classes, etc. Utilizadas no desenvolvimento de projetos para criar aplicações, normalmente seguem padrões como J2EE ou .NET;

c) APIs legado - Conjunto de recursos ou protocolos com a finalidade de se comunicar com aplicações mais antigas.

Podem ser classificadas também pelo tipo de acesso:

a) APIs públicas - Também conhecidas como APIs abertas, estão disponíveis para desenvolvedores ou outros usuários de forma pública para consumo e com poucas restrições, podendo exigir registro, uma chave de API, ou outro tipo de autenticação, ou até mesmo ser completamente aberta. Tem foco principal no acesso por usuários externos, normalmente a empresa ou desenvolvedor responsável possui um site ou

portal no qual disponibiliza a documentação para o uso;

b) APIs privadas - Também conhecidas como APIs internas, são omitidas de usuários externos e podem ser acessadas apenas por sistemas internos de uma empresa e as equipes responsáveis pelos mesmos;

c) APIs de parceiros - Um meio termo entre as anteriores, podem ser acessadas por pessoas de fora da organização desde que recebam permissões exclusivas para tal, o acesso é concedido por ou para os parceiros de negócio envolvidos;

d) APIs compostas - Combinam vários dados ou APIs de exposição diferentes. Desenvolvidas através de recursos de orquestração de APIs.

Pela finalidade:

a) APIs de dados - Fornecem acesso a ferramentas de CRUD (*create*, *read*, *update*, *delete*), conjuntos de dados provenientes de bancos de dados ou provedores de nuvem;

b) APIs de serviços internos - Expõem serviços ou processos internos ou algumas ações complexas;

c) APIs de serviços externos - Serviços de terceiros que podem ser integrados ao seu sistema;

d) APIs de experiência de usuário - Fornecem dados necessários para serem consumidos por seu público-alvo, sejam aplicativos móveis, sites ou portais internos.

E pela arquitetura escolhida, seja REST, SOAP, RPC, baseada em eventos/streaming, entre outras possíveis arquiteturas.

Vantagens e desvantagens

Pode-se destacar algumas vantagens no uso de APIs:

a) A troca de informações é feita de maneira segura, visto que os dados podem ser acessados após realizar uma sequência de requisições por um protocolo de autorização padrão;

b) Possibilidade de monitorar os acessos a API, conhecendo quem, quando e onde foram feitas as requisições e quais as informações foram solicitadas, trazendo auditabilidade a aplicação;

c) Restrição da quantidade de informações trafegadas de acordo com a necessidade ou autorização do sistema;

d) Possibilidade de reimplentar algoritmos e realizar refatorações sem impactar a forma de comunicação do usuário;

e) No caso de substituição de uma máquina, seja por quebra ou por melhoria tecnológica, não é necessário recriar todas as funcionalidades para a máquina nova;

f) Traz maior facilidade para o uso dos recursos por um terceiro que desconheça as minúcias do código-fonte;

g) Mais agilidade na manutenção da aplicação.

E citar como desvantagens:

a) Requer tempo e custo para a implementação;

b) Em caso de quedas de sistema, pode ocasionar a quebra do funcionamento de outros sistemas que dependem dessa API;

c) Necessita de manutenções periódicas para garantir a saúde do ecossitema;

d) Dificuldade na padronização da interface e na escrita de uma documentação clara e eficiente.