UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Tiago Henrique Pavaneli

Desenvolvimento de uma API e uma Aplicação Web para auxiliar na análise de dados providos por Aplicação Móvel

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Tiago Henrique Pavaneli

Desenvolvimento de uma API e uma Aplicação Web para auxiliar na análise de dados providos por Aplicação Móvel

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Thiago Pirola Ribeiro

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2023

Tiago Henrique Pavaneli

Desenvolvimento de uma API e uma Aplicação Web para auxiliar na análise de dados providos por Aplicação Móvel

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 09 de março de 2023:

Prof. Dr. Thiago Pirola Ribeiro
Orientador

Profa. Dra. Ana Cláudia Martinez

Prof. Dr. Rafael Dias Araújo

Uberlândia, Brasil
2023

Foi pensando nas pessoas que executei este projeto, por isso dedico este trabalho a todos aqueles a quem esta pesquisa possa ajudar de alguma forma.

Agradecimentos

Aos meus pais, por nunca terem medido esforços para me proporcionar um ensino de qualidade durante todo o meu período escolar.

Aos amigos, que me incentivaram nos momentos difíceis e compreenderam a minha ausência enquanto eu me dedicava à realização deste trabalho.

E ao professor Thiago Pirola Ribeiro, por ter sido meu orientador e ter desempenhado esta função com dedicação e amizade.

"A persistência é o caminho do êxito"
(Charles Chaplin)

Resumo

Neste trabalho foi desenvolvida uma Interface de Programação de Aplicações (API) para auxiliar na comunicação entre a aplicação móvel e o servidor para abastecer e retornar os dados necessários. Para exemplificar e validar a API, foi desenvolvida uma aplicação Web para uso colaborativo entre profissionais e pacientes portadores de Transtorno do Déficit de Atenção com Hiperatividade (TDAH) para tornar mais fácil a análise dos dados gerados por um aplicativo móvel utilizado pelos pacientes, fornecendo clarificações sobre a evolução dos mesmos em sua jornada de tratamento. As tecnologias utilizadas para o desenvolvimento foram: Arquitetura *Representational State Transfer (REST)*, Banco de Dados Não Somente SQL (NoSQL) (MongoDB), Angular, Node.js, Express.js, além de Folhas de Estilo em Cascata (CSS), JavaScript, TypeScript, Extensões Reativas para JavaScript (RxJS) e *JSON Web Tokens* (JWT). Os testes realizados puderam demonstrar e validar a API desenvolvida para a operacionalização da comunicação entre servidor, banco de dados e aplicação móvel.

Palavras-chave: API, Sistema web, RESTful, MEAN Stack, TDAH.

Lista de ilustrações

Figura 1 – Diagrama de funcionamento do MEAN Stack 24
Figura 2 – Princípios da Arquitetura REST 25
Figura 3 – Representação Gráfica da Comunicação com uma API RESTful 26
Figura 4 – Estrutura de comunicação entre aplicativo móvel, API de *backend* e página web. 35
Figura 5 – Diagramas de Classe dos *Schemas* do Mongoose para o Sistema proposto. 36
Figura 6 – Diagramas de relações entre os *Schemas* do Mongoose 37
Figura 7 – *Middleware* que executa a encriptação de senha. 37
Figura 8 – Gráfico demonstrando o tempo para geração de *hash* X custo. 38
Figura 9 – Sequência de fluxo do Node.js e Express 39
Figura 10 – Documentação interativa gerada pelo Swagger 40
Figura 11 – Exemplo de Objeto JSON enviado pelo Postman. 41
Figura 12 – Exemplo de Objeto JSON recebido pelo Postman. 41
Figura 13 – Objeto JSON armazenado no MongoDB. 42
Figura 14 – Exemplo da resposta de erro recebida pelo Postman. 42
Figura 15 – Exemplo de erro de conexão apresentado no navegador. 43
Figura 16 – Exemplo de erro a ser apresentado na tela do usuário. 44
Figura 17 – Códigos fonte da utilização da biblioteca *Ngx Toastr*. `Esquerda`: módulo principal da aplicação e `Direita`: componente de interceptação de erro. 44
Figura 18 – Funcionamento da autenticação por meio de JWT. 45
Figura 19 – Exemplo da Tela Completa do Sistema. 46
Figura 20 – Exemplo da responsividade do Sistema para dispositivos móveis. Após selecionar o paciente (esquerda), abre-se a tela do lado direito. Ambas as telas são excludentes, ocupando toda a tela do dispositivo móvel. . . 47
Figura 21 – Exemplo da Tela de Login de Usuário. 48
Figura 22 – Exemplo da tela de Inscrição/Cadastro. 48
Figura 23 – Exemplo da tela de Inscrição/Cadastro com campos inválidos. 49
Figura 24 – Menu suspenso de opções do usuário. 49
Figura 25 – Exemplo de opções da página de Administração. 50
Figura 26 – Modal de ação irreversível. 50
Figura 27 – Exemplo da geração de código para pacientes. 51
Figura 28 – Exemplo das opções da página para editar perfil. 51
Figura 29 – Exemplo para a comparação de uma imagem com uma parte de sua codificação em Base64. 52
Figura 30 – Exemplo da lista gerada dinâmicamente de Pacientes (versão para dispositivos móveis). 53

Figura 31 – Informações do paciente sem paciente selecionado (versão para dispositivos móveis). 53
Figura 32 – Exemplo de filtros para seleção (versão para dispositivos móveis). . . . 54
Figura 33 – Exemplo de gráficos disponíveis gerados dinamicamente. 54
Figura 34 – Exemplo de gráfico de dias utilizando a plataforma. 55
Figura 35 – Código referente a função que salva o PDF. 56

Lista de abreviaturas e siglas

ASCII Código padrão americano para troca de informações - *American Standard Code for Information Interchange*

API Interface de Programação de Aplicações - *Application Programming Interface*

ACID Atomicidade, Consistência, Isolamento e Durabilidade - *Atomicity, Consistency, Isolation, and Durability*

CRUD Criação, Consulta, Atualização e Destruição - *Create, Read, Update and Delete*

CSS Folhas de Estilo em Cascata - *Cascading Style Sheets*

CERN Organização Europeia para a Pesquisa Nuclear - *European Council for Nuclear Research*

HTML Linguagem de Marcação de Hipertexto - *Hypertext Markup Language*

HTTP Protocolo de Transferência de Hipertexto - *Hypertext Transfer Protocol*

IBM Máquinas de Negócios Internacionais - *International Business Machines*

IDE Ambiente de Desenvolvimento Integrado - *Integrated Development Environment*

ID identificação - *identification*

JSON Notação de Objeto JavaScript - *JavaScript Object Notation*

JWT *JSON Web Tokens*

MIT Instituto de Tecnologia de Massachusetts - *Massachusetts Institute of Technology*

MEAN MongoDB, Express.js, Angular e Node.js - *MongoDB, Express.js, Angular and Node.js*

NoSQL Não Somente SQL - *Not Only SQL*

ODM Mapeamento de Documento de Objeto - *Object Document Mapping*

PDF Formato de Documento Portátil - *Portable Document Format*

RxJS Extensões Reativas para JavaScript - *Reactive Extensions for JavaScript*

REST Representational State Transfer - *Transferência de estado representacional*

SQL Linguagem de Consulta Estruturada - *Structured Query Language*

TCT Método Cognitivo Tajima - *Tajima Cognitive Method*

TDAH Transtorno do Déficit de Atenção com Hiperatividade

UX Experiência do Usuário - *User Experience*

VSCode *Visual Studio Code*

Sumário

1 INTRODUÇÃO 13
1.1 Motivação 13
1.2 Problema 14
1.3 Hipótese 15
1.4 Objetivos 15
1.5 Organização da Monografia 15

2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 Conceitos Básicos e Tecnologias 16
2.1.1 Visual Studio Code 16
2.1.2 GitHub 17
2.1.3 HTML 17
2.1.4 CSS 18
2.1.5 Desenvolvimento: *Frontend*, *Backend* e *Full Stack* 19
2.1.6 Javascript 19
2.1.7 TypeScript 20
2.1.8 Node.js 21
2.1.9 MongoDB 21
2.1.10 Express.js 22
2.1.11 Angular 23
2.1.12 MEAN *Stack* 24
2.1.13 REST 24
2.1.14 RxJS 26
2.1.15 Bootstrap 27
2.1.16 ApexCharts 28
2.1.17 Ngx Toastr 28
2.1.18 PDF.js 28
2.1.19 HTML2Canvas 29
2.1.20 Bcrypt 29
2.2 Trabalhos Correlatos 30

3 DESENVOLVIMENTO E RESULTADOS 34
3.1 Tecnologias Utilizadas 34
3.2 Desenvolvimento 35
3.2.1 *Backend* 35
3.2.1.1 Base de Dados 35

3.2.1.2 Criptografia 37
3.2.1.3 Ambiente 38
3.2.2 *Frontend* 42
3.2.2.1 Testes e Mensagens 43
3.2.2.2 Autenticação 43
3.2.2.3 Responsividade, Cores e Menus 45
3.2.2.4 Utilização do Sistema 52

4 CONCLUSÃO 57
4.1 Trabalhos Futuros 58

REFERÊNCIAS 59

APÊNDICES 65

1 Introdução

Interface de Programação de Aplicações (API) é um componente crucial no desenvolvimento de software moderno. Uma API fornece um conjunto de regras e protocolos que permitem que diferentes componentes de software se comuniquem e interajam perfeitamente. Com APIs, os desenvolvedores de software podem integrar vários serviços, bancos de dados, e aplicativos em seus projetos de software sem precisar conhecer os detalhes subjacentes de cada componente. APIs podem ser usadas para buscar dados de sistemas remotos, processar informações e executar várias operações em nome do aplicativo (JIN; SAHNI; SHEVAT, 2018).

Neste projeto, foi utilizada uma API RESTful desenvolvida com Node.js e Express no *backend*. A razão para usar uma API é permitir uma comunicação entre os diferentes componentes do projeto. Nesse caso, a API serve como a interface que permite que o aplicativo móvel desenvolvido por outro aluno se comunique com o *backend*. Isso permite que o aplicativo acesse dados e serviços fornecidos pelo *backend* sem precisar conhecer os detalhes subjacentes da implementação. De forma similar, a API permite a comunicação entre o *backend* e a aplicação de *frontend* em Angular também desenvolvida nesse trabalho para que os responsáveis possam analisar os dados provindos do aplicativo móvel.

A API desse projeto foi desenvolvida especificamente para facilitar a comunicação entre o aplicativo móvel e o *backend*. O aplicativo móvel foi desenvolvido para ajudar pacientes com TDAH, fornecendo jogos que auxiliam em seu tratamento. A API permite que o aplicativo móvel acesse dados e serviços fornecidos pelo *backend*, como resultados de jogos e informações do paciente. Além disso, a API se comunica com a aplicação de *frontend* em Angular desenvolvida nesse projeto, que disponibiliza um painel para o responsável acessar os resultados dos jogos do paciente e gráficos para analisar o andamento do tratamento. A API foi projetada para ser eficiente e escalável, permitindo lidar com as solicitações do aplicativo móvel e do site.

1.1 Motivação

O Transtorno do Déficit de Atenção com Hiperatividade (TDAH) continua a ser o rótulo diagnóstico atual para crianças e adultos que apresentam problemas significativos de atenção e, geralmente, também com impulsividade e atividade excessiva (BARKLEY, 2015).

Segundo Mak et al. (2021), levando em consideração que a prevalência agrupada de TDAH foi avaliada em 15.9% para alunos cursando o primeiro ano da faculdade, a busca de

tratamentos e estudos sobre o tema tem se tornado abundante. Torna-se então necessário o desenvolvimento de ferramentas para lidar com o grande volume de dados gerados, como a do presente trabalho, que quantificam e auxiliam nas análises de profissionais nos estudos e tratamentos de portadores de TDAH.

O treino cognitivo é o conjunto de técnicas que contribuem para estimular funções cerebrais e a capacidade mental. Exercícios de treinamento cognitivo tem mostrado um impacto positivo no desenvolvimento de habilidades de atenção e memória (BARROSO et al., 2019).

Os exercícios de treinamento cognitivo são especialmente úteis quando o comprometimento cognitivo é observado e quando um treinamento cognitivo regular e personalizado é realizado. Estudos em participantes com comprometimento cognitivo mostraram que o treinamento cognitivo regular e diário pode melhorar alguns de seus sintomas cognitivos. Além disso, estudos também demonstraram que programas computadorizados de treinamento de memória de trabalho e funções executivas levam a melhores resultados do que métodos de treinamento cognitivo comuns em crianças com TDAH (RUIZ-MANRIQUE; TAJIMA-POZO; MONTAÑES-RADA, 2014).

Em um trabalho de Iniciação Científica, desenvolvido por outro aluno da FACOM/UFU, está sendo desenvolvido um aplicativo para dispositivos móveis que propõe a utilização de jogos/exercícios para o treinamento cognitivo focado em crianças com TDAH.

Levando em consideração a quantidade de dados gerados por meio desses exercícios desenvolvidos com a finalidade de treinamento cognitivo, torna-se útil o desenvolvimento de ferramentas utilizadas por profissionais para análise desses dados fornecidos por aplicativos móveis. O presente trabalho busca desenvolver uma solução web para auxiliar profissionais a realizarem essas análises.

1.2 Problema

Os aplicativos atuais produzem enormes volumes de dados. Enquanto a capacidade de coletar e armazenar novos dados cresce rapidamente, a capacidade de analisar esses dados não acompanha o mesmo ritmo (KEIM et al., 2006).

Levando em consideração a necessidade de especialistas de acompanhar vários casos ao mesmo tempo, analisar o progresso de vários pacientes, realizar acompanhamento individual, visualizando as respostas de entrada de cada exercício, sem o auxílio de uma ferramenta que agrupe todos esses dados e os separe por categoria, se torna uma tarefa complexa e não muito prática.

1.3 Hipótese

Desenvolver uma solução que auxilie nas análises, que serão feitas por profissionais, dos dados gerados pelo uso de jogos por pacientes portadores de TDAH com o uso de um aplicativo móvel, que tem o intuito de uma avaliação/treino computadorizado. A comunicação entre a solução e o aplicativo deverá ser por meio de uma API em um servidor para armazenamento dos dados de forma segura, rápida e em larga escala.

Deve-se permitir o fácil acesso e recuperação desses dados do servidor de forma a serem apresentados visualmente e comparativamente através de uma aplicação web acessada unicamente pelo especialista responsável, mantendo a comunicação e confiança entre especialista e paciente.

1.4 Objetivos

O objetivo deste trabalho é produzir uma solução para o armazenamento e acesso seguro de dados gerados por uma aplicação móvel de exercícios de treinamento cognitivos, permitindo a categorização dos dados e visualização por meio de uma Experiência do Usuário (UX) desenvolvida nesse projeto. Os objetivos específicos do trabalho são:

- Modelagem e Criação do Bancos de Dados NoSQL.
- Programação de uma API RESTful a ser consumida por um aplicativo móvel para armazenamento de dados e por uma aplicação web para visualização dos mesmos.
- Desenvolvimento de uma página web logável e dinâmica direcionada à base desenvolvida.
- Um sistema de visualização configurável dentro da página com opções para salvar e analisar os dados.

1.5 Organização da Monografia

No Capítulo 2 será abordada toda a fundamentação teórica com o objetivo de auxiliar na compreensão do trabalho e das tecnologias utilizadas. No Capítulo 3 serão apresentados os resultados das aplicações desenvolvidas junto aos exemplos e formas de utilização das mesmas. Finalizando com o Capítulo 4 que abrange as conclusões juntamente com as principais contribuições obtidas no trabalho e indicativos de incrementações que possam beneficiar as soluções desenvolvidas.

2 Fundamentação Teórica

Neste capítulo será apresentada a fundamentação teórica do trabalho que fornecerá a base para compreender o problema que está sendo abordado e a abordagem adotada para resolvê-lo. Este capítulo apresentará as teorias, modelos e frameworks relevantes que informam a pesquisa e fornecem um contexto para a análise e discussão subsequentes dos resultados.

2.1 Conceitos Básicos e Tecnologias

Diversos conceitos e tecnologias foram necessárias para o desenvolvimento desse trabalho. Os próximos tópicos abordarão os itens essenciais para o entendimento do Capítulo 3.

2.1.1 Visual Studio Code

Para o desenvolvedor de software, escolher o Ambiente de Desenvolvimento Integrado (IDE) certo é crucial para melhorar a produtividade e a qualidade do código. O *Visual Studio Code* (VSCode), um editor de código da empresa Microsoft[1], emergiu como um dos IDEs mais populares nos últimos anos e existem várias razões pelas quais se tornou a escolha preferida entre os desenvolvedores.

O VSCode fornece uma interface do usuário elegante e moderna que é intuitiva e personalizável. Isso permite que os desenvolvedores adaptem o ambiente às suas necessidades e preferências específicas, tornando mais fácil se concentrar na tarefa em questão. Em "O Programador Prático" (HUNT; THOMAS, 1999), os autores afirmam que ter um ambiente de trabalho eficiente e bem projetado é essencial para aumentar a produtividade.

Conta com uma vasta gama de plugins e extensões disponíveis através da sua aba de extensões. Isso significa que os desenvolvedores podem facilmente adicionar novos recursos e ferramentas ao ambiente para agilizar o fluxo de trabalho. É relevante enfatizar a importância de usar ferramentas que apoiem e melhorem o processo de desenvolvimento de software (MARTIN, 2008).

Fornece uma ampla gama de ferramentas de depuração e teste, o que ajuda os desenvolvedores a encontrar e corrigir erros de código mais rapidamente e efetivamente. Autores enfatizam a importância de ter ferramentas de depuração robustas como um componente crítico do desenvolvimento de software (AGANS, 2002), integrando-se perfeitamente a uma ampla gama de tecnologias e plataformas, tornando-se uma escolha

[1] Pode ser obtido em <https://code.visualstudio.com/download>

versátil para desenvolvedores trabalhando em uma variedade de projetos. É imprescindível destacar a necessidade de usar ferramentas flexíveis e adaptáveis a diferentes requisitos de projeto (FREEMAN et al., 2004).

Além disso, o VSCode é código aberto e gratuito, tornando-o acessível a desenvolvedores de todos os níveis de habilidade e *background*. Destacam-se então as vantagens do uso de software open-source, incluindo acessibilidade, colaboração e inovação (RAYMOND, 2001).

2.1.2 GitHub

O GitHub[2], da empresa Microsoft, é uma plataforma amplamente utilizada para desenvolvimento de software que oferece vários benefícios para desenvolvedores e equipes.

A centralização de código em um único repositório no GitHub torna mais fácil para as equipes colaborarem e acompanhar as alterações. Como Laura Lorenz explica em seu livro "*GitHub for Collaborative Software Development*"(MISTRÍK et al., 2010) essa centralização agiliza o processo de desenvolvimento e melhora a comunicação entre os membros da equipe.

O fornecimento do controle de versão garante que todos os membros da equipe estejam trabalhando com a versão mais recente do código e possibilita retornar a versões antigas do código ou analisar as diferenças entre cridas com a escrita de novos códigos. Chris Dannen enfatiza a importância do controle de versão no desenvolvimento de software e como o sistema do GitHub torna fácil acompanhar as alterações e trabalhar com outros (BELL; BEER, 2014).

O compartilhamento do código com a comunidade mais ampla, proporcionando uma oportunidade para *feedback* e possíveis novas colaborações por futuros desenvolveres mantendo o projeto. Ryan Ayers enfatiza o valor do GitHub como plataforma para compartilhamento de código e construção de novas parcerias na comunidade de desenvolvimento (UZAYR, 2022).

2.1.3 HTML

A Linguagem de Marcação de Hipertexto (HTML) é uma linguagem de marcação usada para criar páginas e aplicativos da web. O HTML foi criado pela primeira vez em 1989 por Tim Berners-Lee, um cientista da computação na Organização Europeia para a Pesquisa Nuclear (CERN).

O HTML é usado para definir a estrutura e o conteúdo de uma página da web, incluindo títulos, parágrafos, listas, links, imagens e muito mais. O HTML usa tags para

[2] <https://github.com>

definir elementos, com a *tag* cercada por colchetes angulares (por exemplo, `<p>` para um parágrafo) (DUCKETT, 2014b).

A linguagem evoluiu ao longo dos anos para incluir novos recursos e capacidades, como a capacidade de incluir multimídia como áudio e vídeo, e a adição de novos elementos como controles de formulários e novos elementos semânticos como cabeçalho e rodapé. A última versão do HTML, introduziu novos elementos e atributos para melhorar o significado semântico de uma página, tornando mais fácil para os mecanismos de busca entenderem o conteúdo de uma página (DUCKETT, 2011).

O uso do HTML é essencial para criar páginas da web acessíveis a todos os usuários, incluindo aqueles com deficiências. A linguagem permite que os desenvolvedores web adicionem texto alternativo para imagens, legendas para vídeos e descrições para gráficos e tabelas, tornando o conteúdo acessível ao software de leitor de tela para usuários com deficiência visual (HORTON; QUESENBERY, 2014).

2.1.4 CSS

As Folhas de Estilo em Cascata (CSS) é uma linguagem de estilização usada para descrever a aparência e formatação de um documento escrito em HTML. É um aspecto crucial do design de sites, permitindo que os desenvolvedores de sites separem o conteúdo e apresentação de um site.

O CSS foi introduzido pela primeira vez em 1996 como uma forma de melhorar a aparência das páginas da web além da formatação básica do HTML. Rapidamente ganhou popularidade e se tornou um padrão para o design de sites devido à sua capacidade de controlar o layout e os elementos visuais de um site de forma consistente e eficiente.

O CSS pode ser escrito diretamente no documento HTML ou em um arquivo CSS separado que é vinculado ao documento HTML. O uso de arquivos CSS separados se tornou uma boa prática, pois permite maior consistência e manutenibilidade dos estilos de um site. Isso ocorre porque as alterações feitas em um único arquivo CSS podem afetar várias páginas HTML, reduzindo a quantidade de código redundante e tornando mais fácil atualizar a aparência e o estilo de um site (MEYER, 2011).

O CSS fornece uma ampla gama de opções de formatação, incluindo a capacidade de definir estilos de fontes, cores de fundo, bordas e espaçamentos. Também permite um controle avançado de layout, como definir a posição de elementos, usar caixas flexíveis e usar consultas de mídia para ajustar os estilos com base no tamanho da tela e orientação do dispositivo. O uso de consultas de mídia em CSS tem se tornado cada vez mais importante nos últimos anos à medida que o número de dispositivos acessando a web continua a crescer (FRAIN, 2014).

2.1.5 Desenvolvimento: *Frontend*, *Backend* e *Full Stack*

Desenvolvimento de *Frontend*, *Backend* e *Full Stack* são três aspectos separados, porém inter-relacionados no desenvolvimento de software. O desenvolvimento de *frontend* se concentra na criação da interface do usuário e na experiência. Já o desenvolvimento de *backend* lida com a lógica do lado do servidor e armazenamento de dados. O desenvolvimento *Full Stack* engloba tanto o desenvolvimento *frontend* quanto o *backend*.

O desenvolvimento de *frontend* exige uma compreensão sólida de HTML, CSS e JavaScript. Desenvolvedores de *frontend* usam essas habilidades para construir aplicativos web atraentes visualmente e fáceis de utilizar e que rodam no navegador. O foco é criar uma interface que seja rápida, responsiva e acessível a todos os usuários (ACCOMAZZO; MURRAY; LERNER, 2017).

Por outro lado, o desenvolvimento de *Backend* envolve trabalhar com bancos de dados, lógica do lado do servidor e APIs. Desenvolvedores de *Backend* usam linguagens de programação para escrever código do lado do servidor e construir modelos de dados. Eles também criam e gerenciam APIs que permitem ao *frontend* comunicar com o *Backend* e acessar dados (BASS; CLEMENTS; KAZMAN, 2012).

O desenvolvimento *Full Stack* requer uma compreensão abrangente tanto do desenvolvimento de *frontend* quanto do de *Backend*. Desenvolvedores *Full Stack* trabalham tanto no lado do cliente quanto no lado do servidor de uma aplicação, permitindo-lhes construir e implantar aplicativos web completos. Eles devem ser capazes de escrever código limpo e eficiente e trabalhar com múltiplas tecnologias (IHRIG; BRETZ, 2014).

2.1.6 Javascript

JavaScript é uma linguagem de programação de alto nível, interpretada, que foi introduzida pela primeira vez em 1995. Foi criada por Brendan Eich enquanto ele trabalhava na empresa *Netscape Communications Corporation*. O JavaScript foi inicialmente destinado a ser uma linguagem de *script* para navegadores da web, mas evoluiu para uma linguagem poderosa que pode ser usada tanto para o desenvolvimento de *frontend* quanto de *Backend*. A grande maioria dos sites a utiliza e todos os principais navegadores da web possuem um mecanismo JavaScript dedicado para executá-la (GROSSKURTH; GODFREY, 2005).

É conhecida por sua tipagem dinâmica, funções de primeira classe tratadas como variáveis e capacidade de programação orientada a objetos. O JavaScript também é uma linguagem baseada em eventos, o que significa que sua execução é determinada por eventos, como interação do usuário com uma página da web e juntamente com HTML e CSS, JavaScript é uma das três principais tecnologias da web (GUHA; SAFTOIU; KRISHNAMURTHI, 2010).

JavaScript é uma linguagem fracamente tipada, o que significa que "o tipo de dados de um valor é determinado em tempo de execução, e não no momento em que o código é escrito". Isso permite uma alta flexibilidade no código e pode tornar mais fácil escrever aplicativos complexos (FLANAGAN, 2011).

Nos últimos anos, o aumento de frameworks de JavaScript do lado do servidor, como o Node.js, tornou possível construir aplicações web *Full Stack* usando apenas JavaScript. Isso ainda mais consolidou a posição do JavaScript como uma das linguagens de programação mais populares do Mundo.

O JavaScript tem uma biblioteca padrão pequena e simples, com poucos tipos de dados além de objetos básicos e funções. Apesar do seu pequeno tamanho, ele é capaz de lidar com "aplicativos complexos". Essa simplicidade tornou mais fácil para os desenvolvedores aprender e usar, levando à sua ampla adoção (HAVERBEKE, 2018).

2.1.7 TypeScript

Typescript[3] é uma linguagem de programação estaticamente tipada, desenvolvida e mantida pela Microsoft. Foi introduzida pela primeira vez em 2012 e, desde então, ganhou muita popularidade entre a comunidade de desenvolvedores.

A sintaxe do Typescript é um superconjunto do JavaScript, o que significa que qualquer código JavaScript válido também é um código Typescript válido. No entanto, o Typescript adiciona vários recursos, como anotações de tipo, interfaces e classes, o que torna o código mais estruturado e fácil de manter (BRADLEY, 2021).

Um dos principais benefícios de usar o Typescript é a melhor leitura e manutenção do código. Com a adição de anotações de tipo, os desenvolvedores podem capturar erros relacionados a tipos durante a compilação, em vez de tempo de execução. Isso ajuda a capturar bugs cedo no ciclo de desenvolvimento e reduz o tempo necessário para depurar o código.

Outra vantagem do Typescript é a sua interoperabilidade com bibliotecas JavaScript. Como o Typescript é um superconjunto do JavaScript, os desenvolvedores podem facilmente integrar bibliotecas JavaScript existentes aos seus projetos Typescript. Além disso, as anotações de tipo fornecidas pelo Typescript permitem aos desenvolvedores criar definições de tipo para bibliotecas JavaScript, que podem ser compartilhadas com outros desenvolvedores (NANCE, 2014).

[3] <https://www.typescriptlang.org>

2.1.8 Node.js

Node.js[4] é um ambiente de execução JavaScript de *Backend* de código aberto e multiplataforma. Foi introduzido pela primeira vez em 2009 por Ryan Dahl e desde então tornou-se uma das tecnologias mais populares para a construção de aplicativos do lado do servidor.

A ideia para o Node.js surgiu de sua frustração com as limitações dos servidores da web na época. Ele acreditava que os servidores da web deveriam ser capazes de lidar com mais conexões simultâneas, então ele se propôs a criar uma tecnologia que pudesse fazer isso. Em seu livro, Dahl explica que ele foi inspirado pelo design do ambiente de execução JavaScript do Google Chrome e buscou trazer esse tipo de desempenho para o lado do servidor (HUGHES-CROUCHER; WILSON, 2012).

Um dos recursos chave do Node.js é seu modelo de entrada e saída baseado em eventos e não bloqueado. Isso permite que o Node.js lide com um grande número de conexões simultâneas com baixo *overhead*. Como resultado, o Node.js é adequado para a construção de aplicativos de rede escaláveis e de alta performance, como servidores da web e aplicativos em tempo real.

Além de seus benefícios de desempenho, o Node.js também se tornou popular devido à sua grande e ativa comunidade. Há muitas ferramentas, bibliotecas e módulos disponíveis para o Node.js que tornam mais fácil para os desenvolvedores construir e implantar aplicativos. O grande número de módulos disponíveis para o Node.js o torna uma tecnologia altamente versátil que pode ser usada para uma ampla gama de aplicativos (BUTTIGIEG; JEVDJENIC, 2015).

Apesar de sua popularidade e sucesso, o Node.js não está isento de desafios. Um dos maiores desafios enfrentados pelo Node.js é a necessidade de manter a compatibilidade com versões anteriores, ao mesmo tempo em que adiciona novos recursos e melhorias (OXLEY et al., 2017). Isso pode ser um equilíbrio difícil que exige atenção cuidadosa aos detalhes e planejamento.

Apesar desses desafios, o Node.js continua sendo uma das tecnologias mais amplamente utilizadas para a construção de aplicativos de *Backend*. Sua combinação de desempenho, versatilidade e suporte à comunidade o torna a melhor escolha para desenvolvedores em todo o mundo.

2.1.9 MongoDB

MongoDB[5] é um sistema popular de gerenciamento de banco de dados NoSQL que tem ganhado ampla adoção nos últimos anos. Ele foi lançado em 2009 e desde então se

[4] <https://nodejs.org>
[5] <https://www.mongodb.com>

tornou um dos principais bancos de dados NoSQL em uso atualmente.

O MongoDB é baseado na ideia de armazenamento de dados orientado a documentos, o que é fundamentalmente diferente dos sistemas tradicionais de gerenciamento de banco de dados relacionais que usam tabelas estruturadas e linhas. Em um banco de dados orientado a documentos como o MongoDB, os dados são armazenados como documentos semi-estruturados ou não estruturados, como Notação de Objeto JavaScript (JSON), que podem ser facilmente lidos e escritos em uma variedade de linguagens de programação (BRADSHAW; CHODOROW; BRAZIL, 2019).

As principais diferenças entre bancos de dados NoSQL e Linguagem de Consulta Estruturada (SQL) são frequentemente encontradas em seu modelo de dados e capacidade de escalabilidade. Os bancos de dados SQL são baseados em um modelo relacional e usam um esquema fixo, o que os torna mais adequados para dados estruturados, como transações financeiras ou dados de clientes. Já os bancos de dados NoSQL como o MongoDB são projetados para alta escalabilidade e flexibilidade, tornando-os ideais para lidar com grandes quantidades de dados semi-estruturados ou não estruturados, como postagens em mídias sociais ou avaliações de clientes on-line (EDWARD; SABHARWAL, 2015).

Outra diferença chave entre bancos de dados NoSQL e SQL é na forma como eles lidam com consistência de dados e transações. Os bancos de dados SQL usam Atomicidade, Consistência, Isolamento e Durabilidade (ACID) para garantir consistência de dados, mas isso pode resultar em gargalos de desempenho em grande escala. Já os bancos de dados NoSQL como o MongoDB usam consistência eventual, o que permite maior escalabilidade e desempenho, mas pode sacrificar alguma consistência de dados (HOLMES; HARBER, 2019).

2.1.10 Express.js

Express.js[6] é um popular framework de aplicações web de código aberto para Node.js, projetado para construir APIs e aplicações web rápidas e escaláveis.

O Express.js foi criado por TJ Holowaychuk em 2010 e foi lançado sob a licença Instituto de Tecnologia de Massachusetts (MIT). Foi inspirado no framework Ruby on Rails e buscou trazer um nível semelhante de simplicidade na construção de aplicações web com Node.js (L, 2016).

O Express.js ganhou rapidamente popularidade entre os desenvolvedores Node.js devido à sua facilidade de uso e flexibilidade. Oferece um conjunto mínimo e flexível de recursos para aplicativos web e móveis e é o framework subjacente para várias aplicações web populares Node.js, como StrongLoop e Máquinas de Negócios Internacionais (IBM) API Connect.

[6] <https://expressjs.com>

Uma das características-chave do Express.js é o seu design modular, que permite aos desenvolvedores usar somente os componentes que precisam e facilmente estender o framework com um middleware personalizado. Isso torna o Express.js um framework altamente personalizável que pode ser adaptado para atender às necessidades específicas de qualquer aplicação web (MARDAN, 2014).

Outro aspecto importante do Express.js é o seu suporte a rotas, que é o processo de determinar como uma aplicação web deve responder a uma solicitação. O Express.js fornece um sistema de roteamento simples e intuitivo que facilita o tratamento de solicitações, criação de APIs e construção de páginas web dinâmicas (BROWN, 2014).

Apesar de suas muitas vantagens, o Express.js também tem suas limitações. Alguns desenvolvedores criticaram a falta de suporte embutido para determinados recursos, como engines de template e conectividade com banco de dados. No entanto, essas limitações podem ser facilmente superadas usando módulos ou plugins de terceiros.

2.1.11 Angular

Angular[7] é um popular framework de código aberto baseado em JavaScript para construção de aplicativos web. Ele foi lançado em 2010 pela Google e desde então tornou-se um dos frameworks amplamente utilizados para a construção de aplicativos web dinâmicos e complexos.

O Angular foi introduzido pela primeira vez em 2010 como AngularJS, um framework para construção de aplicativos web dinâmicos e complexos. AngularJS foi projetado para abordar os desafios enfrentados pelos desenvolvedores web na criação de aplicativos web complexos e interativos. Ele foi criado como uma resposta às limitações do HTML, que foi projetado principalmente para páginas web estáticas e não era adequado para aplicativos dinâmicos e interativos. O AngularJS ofereceu uma solução ao oferecer uma série de ferramentas e componentes que poderiam ser facilmente integrados aos aplicativos web existentes (FREEMAN, 2018).

Ao longo dos anos, o Angular evoluiu significativamente, com a introdução de novos recursos e melhorias visando torná-lo mais fácil de usar e mais poderoso. Em 2016, a Google lançou o Angular 2, uma reescrita completa do AngularJS, que se concentrou em melhorar o desempenho, escalabilidade e acessibilidade. O Angular 2 ofereceu uma arquitetura mais modular e escalável, desempenho melhorado e melhor suporte a dispositivos móveis (FAIN; MOISEEV, 2017).

A versão mais recente do Angular, o Angular 15, foi lançado em novembro de 2022. Os novos recursos do Angular 15 incluem APIs independentes estáveis, permitindo que os desenvolvedores do Angular criem aplicativos sem os módulos Ng. Ele também

[7] <https://angular.io>

oferece menos código clichê, desempenho aprimorado, API de composição de diretivas e muitas outras atualizações e recursos para desenvolvedores em termos de experiência e desempenho (TECHNOLOGY, 2023).

2.1.12 MEAN *Stack*

MongoDB, Express.js, Angular e Node.js (MEAN) formam o acrônimo para conceber esse padrão de projeto, é considerado o um procedimento de desenvolvimento, testado e comprovado. É uma coleção poderosa e versátil de tecnologias baseadas em JavaScript que podem ser usadas para construir aplicações web *Full Stack* (NIRGUDKAR; SINGH, 2017).

A Figura 1 ilustra como as tecnologias do padrão MEAN *Stack* comunicam entre si, o usuário utiliza a interface da aplicação *frontend* em Angular, essa aplicação faz solicitações HTTP ao servidor Node.js que lida com elas em tempo de execução, enquanto o framework do Express lida com as rotas de pedidos e respostas do MongoDB, repassando as informações pros Node.js que por sua vez retorna a solicitação HTTP ao Angular que fica responsável por exibir os dados na interface do usuário.

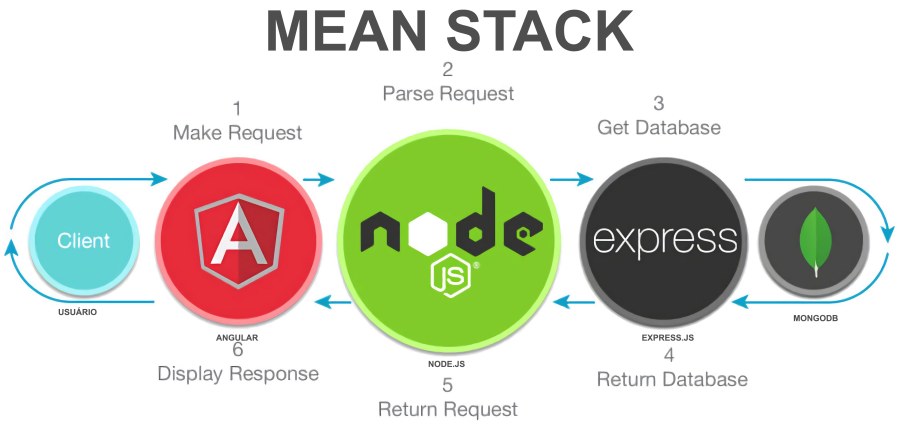

Figura 1 – Diagrama de funcionamento do MEAN Stack

Fonte:<https://roaringstudios.com/assets/img/why-mean-stack.jpg>

2.1.13 REST

Representational State Transfer (REST) é um estilo de arquitetura de software que define um conjunto de restrições para a criação de serviços da web. Foi introduzido pela primeira vez por Roy Fielding em sua tese de doutorado "Architectural Styles and the Design of Network-based Software Architectures" (FIELDING, 2000).

O padrão REST é baseado nos princípios do protocolo HTTP e foi projetado para criar serviços da web escaláveis, eficientes e de fácil manutenção. Um serviço que segue o

padrão REST é chamado de RESTful.

A Figura 2 ilustra os princípios-chave do padrão REST, que são ausência de estado, arquitetura cliente-servidor, sistema em camadas, capacidade de cache e código sob demanda (opcional). A ausência de estado significa que cada solicitação de um cliente para um servidor contém todas as informações necessárias para concluir a solicitação e o servidor não armazena nenhum contexto de cliente em seu lado entre as solicitações. Isso torna os serviços REST altamente escaláveis, pois não precisam acompanhar as sessões do cliente.

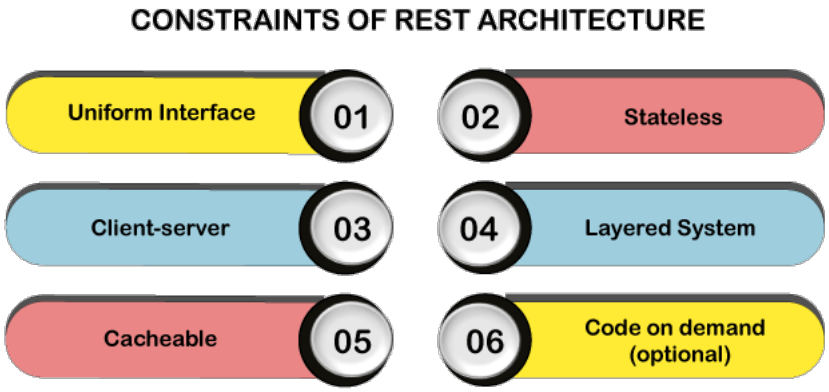

Figura 2 – Princípios da Arquitetura REST

Fonte: <https://static.javatpoint.com/blog/images/what-is-rest.png>

A arquitetura cliente-servidor separa os pedidos da interface do usuário dos pedidos de armazenamento de dados, possibilitando a evolução da interface do usuário independentemente do *Backend*.

O padrão REST opera em um sistema em camadas, o que significa que cada camada se comunica apenas com a camada imediatamente abaixo dela, em vez de diretamente com outras camadas. Isso reduz o acoplamento entre as camadas, facilitando a evolução do sistema ao longo do tempo.

Capacidade de cache é uma propriedade de um serviço RESTful que permite aos clientes armazenar em cache as respostas, reduzindo o número de idas e voltas necessárias para recuperar dados. Isso resulta em melhor desempenho e utilização de rede reduzida.

Finalmente, código sob demanda é uma restrição opcional que permite aos clientes baixar código executável em resposta a uma solicitação. Isso pode ser usado para implementar interfaces de usuário dinâmicas que podem ser adaptadas às necessidades do cliente.

O padrão REST usa vários métodos para Criação, Consulta, Atualização e Destruição (CRUD) de recursos. Esses métodos são baseados nos métodos HTTP (GET, POST,

PUT, DELETE) e são usados para manipular recursos em um servidor. Por exemplo, GET é usado para recuperar um recurso, POST é usado para criar um recurso, PUT é usado para atualizar um recurso e DELETE é usado para excluir um recurso.

A Figura 3 ilustra a comunicação de uma API RESTful entre cliente e servidor, o primeiro envia um um método HTTP para API que por sua vez faz a requisição ao servidor, pega a resposta e devolve o objeto no formato correto para o cliente.

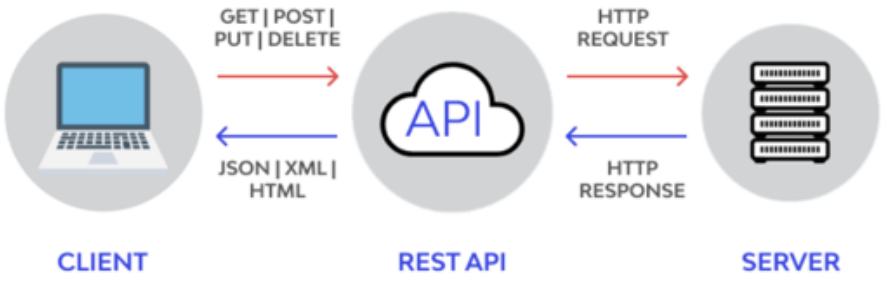

Figura 3 – Representação Gráfica da Comunicação com uma API RESTful

Fonte: <https://miro.medium.com/v2/resize:fit:640/format:webp/1*DjapCLRrKsU2gYGUJXm7Jg.png>

Em suma, REST é um estilo de arquitetura de software amplamente adotado para a construção de serviços da web. Seus princípios de ausência de estado, arquitetura cliente-servidor, sistema em camadas, capacidade de armazenamento em cache e código sob demanda o tornam escalável, eficiente e fácil de manter. Os serviços REST podem ser acessados usando métodos HTTP padrão e podem ser integrados a uma ampla variedade de aplicativos e tecnologias.

2.1.14 RxJS

RxJS[8] é uma biblioteca de programação reativa para JavaScript e TypeScript. Foi lançada em 2011 pela Microsoft e agora é um projeto de código aberto mantido por um grupo de indivíduos e empresas.

O RxJS foi criado para ajudar os desenvolvedores a gerenciar fluxos de dados assíncronos, como entrada de usuários ou dados de um servidor, de maneira mais eficiente e expressiva. Ele fornece uma série de *Observables*, Operadores e Agendadores que permitem aos desenvolvedores processar, transformar e manipular fluxos de dados de maneira funcional e reativa (DANIELS; ATENCIO, 2017).

[8] <https://rxjs.dev>

As raízes do RxJS podem ser rastreadas de volta ao .NET *Reactive Framework*, que foi introduzido em 2009. O .NET *Reactive Framework* foi projetado para tornar mais fácil para os desenvolvedores construir aplicativos assíncronos e baseados em eventos, e foi bem recebido na comunidade .NET. Isso levou à criação do RxJS, que trouxe o poder da programação reativa à comunidade JavaScript (MANSILLA, 2018).

Desde então, o RxJS tornou-se uma ferramenta popular para construir aplicativos da web e foi adotado por muitas empresas, incluindo Netflix, Microsoft e Google. Além disso, o RxJS foi integrado a vários frameworks *frontend* populares, como Angular e React, tornando mais fácil para os desenvolvedores usá-lo em seus projetos.

Apesar de sua popularidade, a programação reativa pode ser um conceito complexo e desafiador de compreender, especialmente para desenvolvedores que são novos nisso (FARHI, 2017).

2.1.15 Bootstrap

O Bootstrap[9] é um framework *frontend* gratuito que foi lançado em 2011m, sendo desenvolvido por designers e desenvolvedores do Twitter, Mark Otto e Jacob Thornton, para simplificar o processo de design de suas ferramentas internas (SPURLOCK, 2013).

O Bootstrap rapidamente ganhou popularidade entre os desenvolvedores da web devido à sua facilidade de uso e personalização. Ele foi projetado para permitir que os desenvolvedores criem sites responsivos e focados em dispositivos móveis com uma aparência consistente. A abordagem amigável e o design intuitivo da estrutura tornaram-na uma escolha popular para designers e desenvolvedores (JOBSEN; COCHRAN; WHITLEY, 2016).

O Bootstrap passou por várias atualizações importantes desde seu lançamento inicial, cada um adicionando novos recursos e melhorias. Por exemplo, o Bootstrap 3 introduziu suporte para design plano e acessibilidade aprimorada, enquanto o Bootstrap 4 adicionou suporte para flexbox e sistemas de grid aprimorados.

Bootstrap é amplamente utilizado hoje por uma grande comunidade de desenvolvedores em todo o mundo. Muitas grandes empresas, como Amazon, LinkedIn e NASA, usam Bootstrap para projetar e construir seus sites. A estrutura também é uma escolha popular para desenvolver temas e modelos do WordPress (IMSIROVIC, 2017).

Um dos principais benefícios do Bootstrap é sua extensa documentação e suporte da comunidade. Existem muitos recursos disponíveis, como tutoriais, fóruns e plugins de terceiros, que podem ajudar desenvolvedores e designers a implementar suas ideias de maneira rápida e fácil.

[9] <https://getbootstrap.com/>

2.1.16 ApexCharts

O ApexCharts[10] é uma biblioteca JavaScript de código aberto para criar gráficos bonitos e interativos. Ele fornece uma ampla variedade de gráficos como gráficos de linhas, gráficos de barras, gráficos de pizza e muitos outros, tornando-o uma solução ideal para visualizar dados. A biblioteca é altamente personalizável, permitindo que os desenvolvedores criem gráficos que atendam perfeitamente às suas necessidades.

Uma ótima opção para quem deseja criar gráficos com menos código e sem precisar usar várias bibliotecas. A biblioteca também fornece uma API simples e intuitiva, tornando-o fácil de usar mesmo para aqueles que têm pouca experiência com bibliotecas de gráficos (ZHU, 2013).

Uma boa solução para visualização de dados porque é rápida, leve e altamente responsiva oferecendo excelente desempenho, mesmo ao trabalhar com grandes conjuntos de dados, o que é um fator crítico para a visualização de dados (YUK; DIAMOND, 2014).

2.1.17 Ngx Toastr

Ngx Toastr é uma biblioteca Angular para notificações não intrusiva e uma ferramenta essencial para a construção de aplicativos amigáveis. Ele é usado para exibir informações na forma de uma mensagem em tela e fornece várias opções de personalização para adaptar a aparência da mensagem para corresponder ao design do site (KUNZ, 2018).

Uma biblioteca altamente personalizável para exibição de mensagens, tornando-se uma excelente escolha para desenvolvedores da web que desejam exibir notificações importantes para os usuários de maneira visualmente atraente e fornece uma API simples e direta, o que facilita a integração em um aplicativo Angular (FREEMAN, 2020).

Ngx Toastr é uma biblioteca altamente personalizável e fácil de usar para exibir notificações em aplicativos Angular. Sua API simples e várias opções de personalização o tornam uma excelente escolha para desenvolvedores web que desejam melhorar a UX em seus aplicativos.

2.1.18 PDF.js

PDF.js[11] é uma biblioteca de software de código aberto para renderizar arquivos Formato de Documento Portátil (PDF) em navegadores da web. É desenvolvido pela Mozilla Foundation sendo considerada uma das soluções mais abrangentes e sofisticadas para renderização de PDF na web.

[10] <https://apexcharts.com>
[11] <https://mozilla.github.io/pdf.js>

De acordo com um estudo conduzido pela Mozilla Hacks, "Pdf.js fornece uma maneira conveniente para os usuários visualizarem PDFs sem a necessidade de instalar software adicional em seus computadores." (HACKS, 2013). Isso elimina a necessidade de os usuários instalarem visualizadores de PDF separados, que podem consumir muito tempo e exigir atualizações frequentes.

O PDF.js é um ótimo exemplo de como os padrões da web podem ser usados para criar aplicativos poderosos e eficientes executados em um navegador (SIKOS, 2014). Sua capacidade de funcionar perfeitamente com diferentes sistemas operacionais e navegadores da web fornece uma experiência de usuário consistente em diferentes plataformas, tornando mais fácil para os usuários acessar PDFs de uma variedade de dispositivos (LINGRAS; TRIFF; LINGRAS, 2016). Esse recurso torna o PDF.js uma solução versátil para uso pessoal e profissional.

2.1.19 HTML2Canvas

HTML2Canvas[12] é uma biblioteca JavaScript que permite aos desenvolvedores capturar e converter elementos HTML em canvas. Essa biblioteca tem sido amplamente utilizada em muitos aplicativos da web, permitindo a criação de capturas de tela, impressões e PDFs de alta qualidade a partir de páginas da web.

É uma ferramenta útil para capturar e converter conteúdo em um formato de imagem. A biblioteca é fácil de usar e pode ser instalada em um projetos que utilizam Javascript. Uma vez instalada, os desenvolvedores podem usar a API para renderizar elementos HTML específicos na tela, incluindo texto, imagens e até vídeos (DUCKETT, 2014a).

Ela fornece um conjunto robusto de opções para personalizar a saída, como ajustar o tamanho da tela, definir a qualidade da imagem e aplicar transformações aos elementos capturados (DUCKETT, 2014b). Esse nível de flexibilidade permite que os desenvolvedores criem capturas de tela exclusivas e dinâmicas do conteúdo da web. A biblioteca pode ser usada em combinação com outras bibliotecas JavaScript, como PDF.js, para criar PDFs de alta qualidade a partir de páginas da web. Isso pode ser particularmente útil em aplicativos da web que exigem relatórios ou documentos imprimíveis (KOZLOWSKI, 2013).

2.1.20 Bcrypt

Bcrypt é um algoritmo popular de *hash* de senha amplamente utilizado para proteger senhas de usuários em aplicativos da web. O principal objetivo do uso de um algoritmo

[12] <https://html2canvas.hertzen.com>

de *hash* de senha é armazenar as senhas com segurança de forma que, mesmo que o banco de dados seja comprometido, o invasor não conseguirá obter as senhas em texto simples.

O Bcrypt foi criado por Niels Provos e David Mazières em 1999 sendo baseado na cifra de bloco de chave simétrica *Blowfish*. O algoritmo usa uma função de derivação de chave para gerar um *hash* da senha, que é armazenado no banco de dados. A principal vantagem do Bcrypt é que ele é computacionalmente caro, tornando difícil para um invasor forçar os *hashes* e quebrar as senhas.

Um dos principais recursos do Bcrypt é sua capacidade de lidar com *Salt*, um valor aleatório que é adicionado à senha antes do *hash*. *Salting*, como é chamado o uso do *Salt*, ajuda a impedir que invasores usem tabelas pré-computadas de *hashes* para quebrar as senhas, pois o *Salt* adiciona uma camada adicional de complexidade ao *hash* (FERGUSON; SCHNEIER; KOHNO, 2010).

Outro recurso do Bcrypt é sua capacidade de se adaptar ao aumento do poder computacional. O algoritmo permite que o fator de custo, ou o número de iterações usadas no processo de derivação de chaves, seja aumentado à medida que os computadores se tornam mais rápidos. Isso significa que, mesmo que um invasor obtenha acesso a um hardware mais rápido, o tempo necessário para quebrar os *hashes* ainda aumentará (STUTTARD; PINTO, 2011).

2.2 Trabalhos Correlatos

Tem havido um interesse crescente no campo da visualização de dados baseada na web nos últimos anos, particularmente com o surto da pandemia de COVID-19. A necessidade de informações em tempo real e de fácil acesso sobre a disseminação do vírus levou a um aumento no desenvolvimento de painéis de informações interativos baseados na web. Vários estudos foram realizados sobre o uso de várias linguagens de programação e ferramentas para a construção desses painéis.

Nesse contexto, no trabalho de Maksimova e Kolev (2020) “COVID-19 MoniToR”, os autores usaram uma combinação de linguagens de programação de *Backend* (Java, Python e PHP) enconjunto com linguagens de *frontend* (HTML, CSS e JavaScript), para desenvolver painéis web de visualização de dados. O uso de APIs de código aberto, como NovelCOVID API ou COVID API, fornecem informações em tempo real sobre a disseminação do COVID-19, que podem ser facilmente integradas ao aplicativo da web.

O uso de bibliotecas e estruturas JavaScript, como ApexCharts.js, com a capacidade de resposta e interatividade, que permitem a representação de dados dinâmica e escalável, foram utilizadas e comprovadas eficientes para construir painéis de visualização de dados interativos e responsivos.

O artigo de Nugraha e Ahmed (2019) "MEAN stack to enhance the advancement of parking application: A narrative review" discute os problemas de estacionamento, principalmente no horário de pico, devido ao aumento do número de carros particulares. Para resolver esse problema foram desenvolvidos sistemas de estacionamento inteligentes que podem fornecer informações em tempo real sobre vagas de estacionamento. O artigo realizou um estudo empírico de várias reservas de sistemas de estacionamento, monitoramento, e sistemas de orientação e revisou 22 artigos para analisar a tecnologia do sistema usada para construir o aplicativo de estacionamento inteligente. A revisão descobriu que uma abordagem utilizando MEAN *Stack* poderia ser eficaz na produção de um aplicativo em tempo real com alto desempenho, flexibilidade e escalabilidade.

A conclusão alcançada foi de que o avanço futuro do estacionamento inteligente é direcionado para sistemas não relacionais e baseados em *smartphones*. Os benefícios do estacionamento inteligente incluem a redução do tempo e dos recursos desperdiçados durante o processo de encontrar uma vaga de estacionamento manualmente. A maioria dos artigos revisados no artigo usou a linguagem JavaScript. Os sistemas de estacionamento inteligente baseados em MEAN *Stack* mostraram uma tendência positiva, e o avanço tecnológico nessa área possibilitou o desenvolvimento de aplicativos baseados em reservas, informações e orientações aos motoristas. A tecnologia não relacional provou ser confiável para aplicativos rápidos, escaláveis e em tempo real, tornando-a uma boa escolha para um aplicativo de estacionamento inteligente baseado em MEAN *Stack*.

No caso de estudo de Ruiz-Manrique, Tajima-Pozo e Montañés-Rada (2014) "ADHD Trainer: the mobile application that enhances cognitive skills in ADHD patients", os exercícios de treinamento cognitivo demonstraram ser eficazes na melhora dos sintomas cognitivos em indivíduos com comprometimento cognitivo, incluindo aqueles com TDAH. O tratamento padrão para o TDAH inclui medicamentos, tratamento psicossocial e comportamental e exercícios de treinamento cognitivo. Nos últimos anos, programas computadorizados de treinamento de memória de trabalho e funções executivas demonstraram melhores resultados do que os métodos de treinamento cognitivo comuns em crianças com TDAH.

Apesar dos potenciais benefícios dos exercícios de treinamento cognitivo, existem riscos associados ao uso excessivo de dispositivos eletrônicos e ao risco de desenvolver vício em videogame ou internet, especialmente em crianças com TDAH. No entanto, novas tecnologias como os videogames podem ser utilizadas como ferramentas terapêuticas para treinar funções executivas em crianças e adolescentes.

Uma abordagem promissora, demonstrada no estudo de caso, é o Método Cognitivo Tajima (TCT), que tem vantagens importantes em relação às terapias de treinamento cognitivo tradicionais, incluindo maior motivação em crianças para concluir a terapia de treinamento cognitivo e facilidade de acesso ao aplicativo. O Método TCT usa jogos pro-

jetados para serem semelhantes aos videogames comuns que as crianças gostam, fornecer valor de entretenimento e *feedback* sobre desempenhos relativos às pontuações próprias e dos colegas, o que melhora o senso de agência e autoeficácia das crianças.

Tendo em vista o número limitado de técnicas de treinamento cognitivo desenvolvidas para pacientes com TDAH nos últimos anos e as baixas taxas de conclusão do treinamento cognitivo em crianças com TDAH, o desenvolvimento de um sistema de treinamento cognitivo computadorizado diário pode melhorar alguns de seus sintomas cognitivos e ser útil no tratamento do vício em videogames (RUIZ-MANRIQUE; TAJIMA-POZO; MONTAÑÉS-RADA, 2014). Portanto, desenvolver um sistema de treinamento cognitivo para tratar o TDAH que leve em consideração as vantagens e os riscos das novas tecnologias e exercícios de treinamento cognitivo, bem como os potenciais benefícios do Método TCT, pode fornecer uma abordagem eficaz para melhorar os sintomas cognitivos em indivíduos com TDAH.

O artigo "Intelligent mobile malware detection using permission requests and API calls" (ALAZAB et al., 2020) propõe um modelo para detecção inteligente de malware móvel usando solicitações de permissão e chamadas de APIs. Os autores sugerem três estratégias de agrupamento para selecionar as chamadas de APIs mais valiosas que podem maximizar a probabilidade de identificar aplicativos de malware para Android, ou seja, o grupo ambíguo, o grupo de risco e o grupo disruptivo. O estudo usa um conjunto de dados de 27.891 aplicativos Android e os resultados indicam que o modelo proposto é eficaz na detecção de aplicativos móveis de malware e atinge uma medida F de 94,3%. O documento destaca a gravidade do malware móvel e sua ameaça à confidencialidade, integridade e disponibilidade dos sistemas móveis. Os autores de malware móvel estão constantemente atualizando e refinando suas técnicas para evitar a detecção, o que torna necessária a detecção automatizada e inteligente.

Malware móvel refere-se a qualquer tipo de código malicioso que afeta a integridade e a funcionalidade dos sistemas móveis sem o conhecimento ou consentimento do usuário. O documento destaca a crescente sofisticação do malware móvel, que inclui vários tipos, como ransomware, trojans, worms, spyware, rootkits e botnets. O documento observa o aumento de ataques de malware móvel e as limitações dos atuais scanners antimalware contra técnicas de ofuscação, que os autores de malware móvel usam para contornar as barreiras de segurança. Os autores sugerem a necessidade urgente de um sistema automatizado e inteligente utilizando APIs para identificar e detectar aplicativos maliciosos para removê-los dos mercados oficiais e não oficiais e torná-los indisponíveis para download posterior.

O artigo "Analysis and assessment of a knowledge based smart city architecture providing service APIs" (BADII et al., 2017) afirma que as soluções de cidades inteligentes estão transformando dados em serviços para usuários e operadores da cidade, explorando

dados e soluções de análise de dados. Esses serviços geralmente integram dados abertos e privados, dados estáticos e em tempo real de administrações e operadores privados. APIs de cidades inteligentes podem oferecer diferentes funcionalidades, dependendo da arquitetura das soluções escolhidas para passar dos dados aos serviços, limitando ou possibilitando a possibilidade de exploração de dados agregados e reconciliados por algoritmos de raciocínio, possibilitando a produção de serviços sofisticados. A exploração efetiva de dados e relacionamentos semânticos para fornecer serviços inteligentes estimulou a criação de uma arquitetura de cidade inteligente com agregação de dados baseada em computação semântica. A arquitetura proposta incluía uma camada de agregação de dados focada em trazer dados para uma base de conhecimento para a cidade, uma solução para executar diferentes análises de dados com o suporte de ferramentas de gerenciamento de processos e a formalização de APIs de Smart City por meio das quais todos os aplicativos e painéis web e móveis podem acessar os dados e a base de conhecimento com raciocínio e inferência espacial e temporal.

O artigo apresentou uma comparação de uma gama de soluções de arquiteturas modernas para agregação de dados para cidades inteligentes e para fornecer APIs de uma cidade inteligente, oferecendo a comparação em termos de cobertura de requisitos e flexibilidade. A análise destacou as vantagens de apresentar dados agregados semânticos interoperáveis, o que possibilita a integração espacial, temporal e raciocínio conceitual diretamente explorado por APIs de cidades inteligentes.

A solução proposta também foi avaliada em termos de custos computacionais e de rede na nuvem, identificando requisitos que podem ser usados para estimar os recursos necessários para lidar com uma variedade de diferentes cidades inteligentes. A arquitetura e solução de cidade inteligente Sii-Mobility, desenvolvido explorando a ontologia Km4City (Km4City, n.d.), foi apresentado e está acessível como código aberto no Github. Atualmente é adotado pelos projetos de pesquisa e desenvolvimento H2020 da Comissão Europeia (Research and innovation - European Commission, 2020) e pelo projeto GHOST (Italian Ministry of Education, Universities and Research, n.d.) do ministério da educação, universidades e pesquisa da Itália na área metropolitana de Cagliari.

3 Desenvolvimento e Resultados

Este capítulo se inicia com uma breve explicação de cada tecnologia utilizada no projeto, seguido de uma apresentação de imagens de telas do projeto como justificativa para o uso dessas ferramentas para o desenvolvimento e por fim uma avaliação dos resultados atingidos.

3.1 Tecnologias Utilizadas

A IDE escolhida para desenvolver o trabalho foi o VSCode por facilitar escrever, depurar e formatar o código através de suas extensões. O projeto foi armazenado no GitHub em dois repositórios distintos, uma para o *backend* e outra para o *frontend*, para fácil compartilhamento e futuras colaborações. Também foi muito importante o uso do GitHub para versionar o código e restaurar para versões antigas em caso de *bugs*.

O MEAN *Stack* foi utilizado nesse projeto devido ao seu desenvolvimento JavaScript *Full Stack* e tecnologias flexíveis e por sua interatividade e quantidade de bibliotecas disponíveis. Também foi utilizado o Typescript pela sua tipagem estática opcional, recursos orientados a objetos e um sistema de inferência de tipo que facilitou a depuração do código e diminuiu de forma relevante o tempo de produção do projeto. É importante lembrar que o Typescript é transpilado — traduzido em tempo de compilação — para Javascript durante a *build*.

Na camada de *backend*, o MongoDB foi escolhido por ser um banco de dados NoSQL altamente escalável, flexível e dinâmico. Seu modelo de dados orientado a documentos permitiu o armazenamento contínuo e intuitivo das estruturas de dados hierárquicas encontradas durante o projeto. O uso do MongoDB permitiu um desenvolvimento fácil e rápido, bem como alterações dinâmicas de esquema, sem a necessidade de migrações de dados demoradas. Além disso, a rica funcionalidade de consulta e os recursos de indexação do MongoDB permitiram uma recuperação de dados eficiente, facilitando o desempenho em tempo real do projeto.

Para utilização do MongoDB dentro do projeto, a Mongoose foi utilizada como uma biblioteca de Mapeamento de Documento de Objeto (ODM) fornecendo uma maneira simples e conveniente de modelar os dados armazenados e executar operações nesses dados. A definição do modelo baseado em esquema do Mongoose permitiu a aplicação de tipos de dados e regras de validação. Além disso, o *middleware* do Mongoose e a funcionalidade de criação de consultas permitiram operações de banco de dados sofisticadas, simplificando o processo de interação com o banco de dados e reduzindo a quantidade de código necessária.

Ainda na camada de *backend*, Node.js e Express criaram uma solução do lado do servidor. O Node.js como ambiente de tempo de execução, forneceu a capacidade de realizar processamento de alto desempenho no lado do servidor. Express, como a estrutura da web de *backend* construída sobre o Node.js, permitiu a criação da API de padrão REST e a implementação de lógica de *backend* complexa. O alto desempenho e escalabilidade dessa combinação a tornaram a escolha ideal para lidar com os requisitos de tempo real desse projeto, enquanto seu design minimalista e modular permitiu uma fácil customização e integração com outras ferramentas e bibliotecas.

3.2 Desenvolvimento

O projeto foi baseado na estrutura de comunicação entre um aplicativo móvel, API de *backend* e página da web. Como ilustrado na Figura 4, a aplicação em Angular executa em um servidor e envia requisições para a API do *backend*, assim como o aplicativo *mobile*.

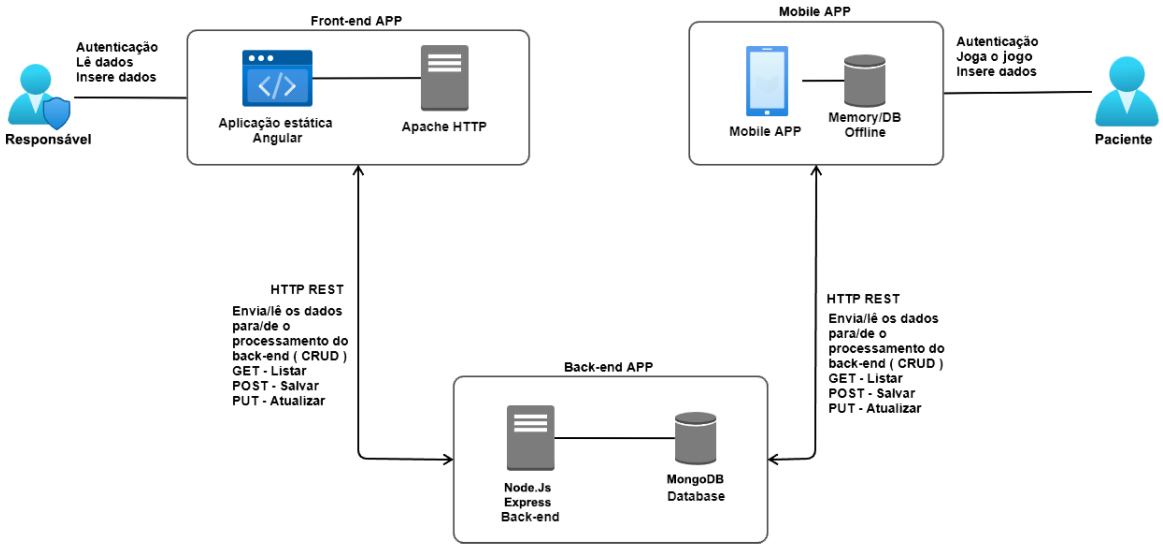

Figura 4 – Estrutura de comunicação entre aplicativo móvel, API de *backend* e página web.

3.2.1 *Backend*

Nessa seção serão descritas as técnicas e implementações realizadas no *Backend* para que o Sistema trabalhe conforme descrito. As explicações foram separadas em partes referentes à Base de Dados, Criptografia e ao Ambiente para que o Sistema executasse.

3.2.1.1 Base de Dados

Para assegurar a comunicação descrita na seção 3.2, foi necessária criar uma estrutura de banco de dados usando os *Schemas* da biblioteca Mongoose. *Schemas* são as

definições da estrutura de uma coleção do MongoDB que especificam as propriedades, tipos de dados e regras de validação dos documentos que podem ser armazenados naquela coleção.

Durante o restante do texto e no Sistema/Projeto desenvolvido, `usuário` é quem utilizará o Sistema que, normalmente será um especialista ou médico, sendo este responsável pelos pacientes. Os pacientes não terão acesso ao Sistema, apenas ao aplicativo móvel que se comunicará com a API do Sistema.

O usuário tem em seu *Schema* além dos campos comuns (nome, email, senha, etc.), um campo para armazenar um *array* de códigos gerados e um campo de privilégio. O paciente tem um campo de código de login que é utilizado na criação de seus dados no banco para relacionar o paciente ao usuário do Sistema por meio do campo de responsável.

No aplicativo móvel que está sendo desenvolvido são propostos jogos variados que serão desenvolvidos ao longo do tempo. Por ainda não estarem implementados os jogos, nesse projeto pensou-se em criar uma estrutura que não necessitasse de alterações tanto de estrutura de banco de dados quanto de interfaces para cada jogo que fosse desenvolvido.

Com isso, propõe-se que o jogo tenha uma estrutura de dados dinâmica para a geração de seus resultados, possibilitando na criação de cada jogo, uma entrada diferente para as propriedades que serão analisadas pelo usuário do Sistema para aquele jogo. Os resultados de cada jogo serão relacionados a cada tipo de jogo e ao paciente que o jogou por meio da identificação (ID) de cada um no banco de dados (Figura 5).

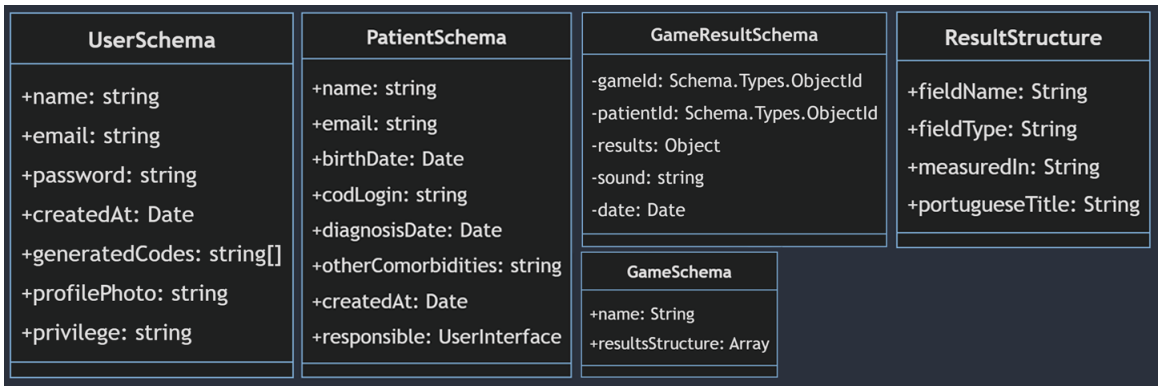

Figura 5 – Diagramas de Classe dos *Schemas* do Mongoose para o Sistema proposto.

A Figura 6 ilustra como cada estrutura de *Schema* se relaciona. Os resultados dos jogos (`GameResult`) estão ligados a cada jogo (`Game`) pelo seu ID e estrutura de resultados (`resultsStructure`), também estão ligados ao paciente pelo ID do mesmo. O paciente (`Patient`) é ligado ao usuário (`User`) pelo código de login utilizado, tendo sido gerado pelo usuário.

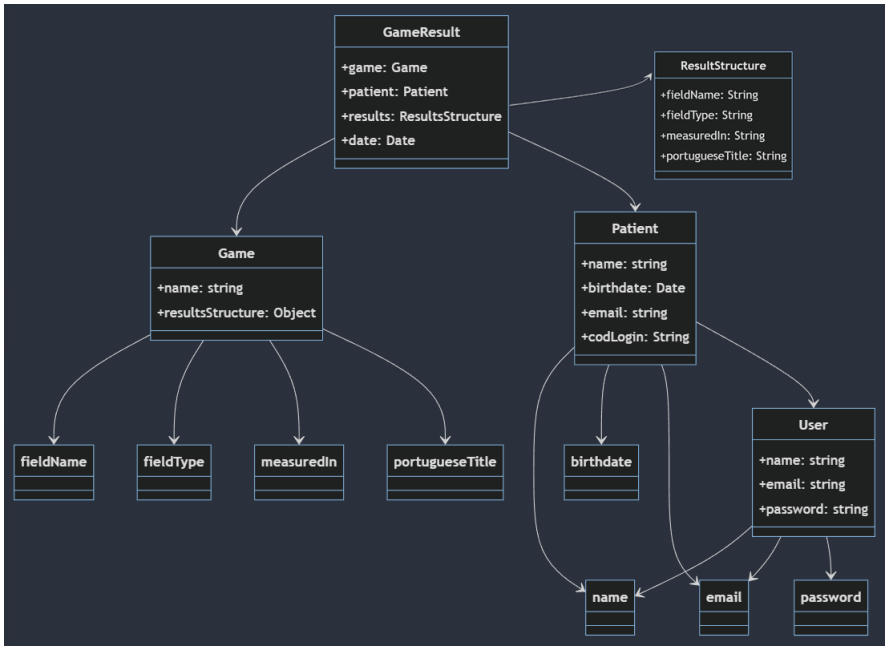

Figura 6 – Diagramas de relações entre os *Schemas* do Mongoose

3.2.1.2 Criptografia

Ao criar um usuário no banco de dados, tornou-se necessário encriptar as senhas para um login mais seguro e, para isso foi escolhida a biblioteca Bcrypt, com uma função *hash* criptográfica altamente segura e amplamente utilizada, sendo escolhida por sua capacidade de garantir a confidencialidade e proteção de informações confidenciais do usuário. A utilização da função de *Salt* e de derivação de chave para criar um *hash* da senha, faz com que seja computacionalmente inviável reverter o processo e descobrir a senha original, assim, mesmo em caso de violação de dados, as informações confidenciais dos usuários permanecem protegidas.

O *Schema* de usuário tem essa particularidade que é uma função de *middleware* que utiliza o método "pre", executando todas as vezes que o um novo usuário é salvo e encriptando seu registro de senha salvo no servidor (Figura 7).

```
UserSchema.pre<UserInterface>('save', async function (next) {
    const user = this;
    if (!user.isModified('password')) {
        return next();
    }
    const salt = await bcrypt.genSalt(10);
    user.password = await bcrypt.hash(user.password, salt);
    next();
});
```

Figura 7 – *Middleware* que executa a encriptação de senha.

Observa-se (Figura 7) que foi utilizado `genSalt(10)` e isto significa que o custo

computacional para descriptografar o item gerado será de 2^{10} ciclos de *clock*. A biblioteca Bcrypt permite utilizar o custo de até 31 (2^{31} = 2.147.483.648), porém o custo para a geração da criptografia também aumenta. Na Figura 8 pode ser observado um exemplo que o tempo para geração de *hash* de uma senha cresce exponencialmente à medida que o custo aumenta (ARIAS, 2021).

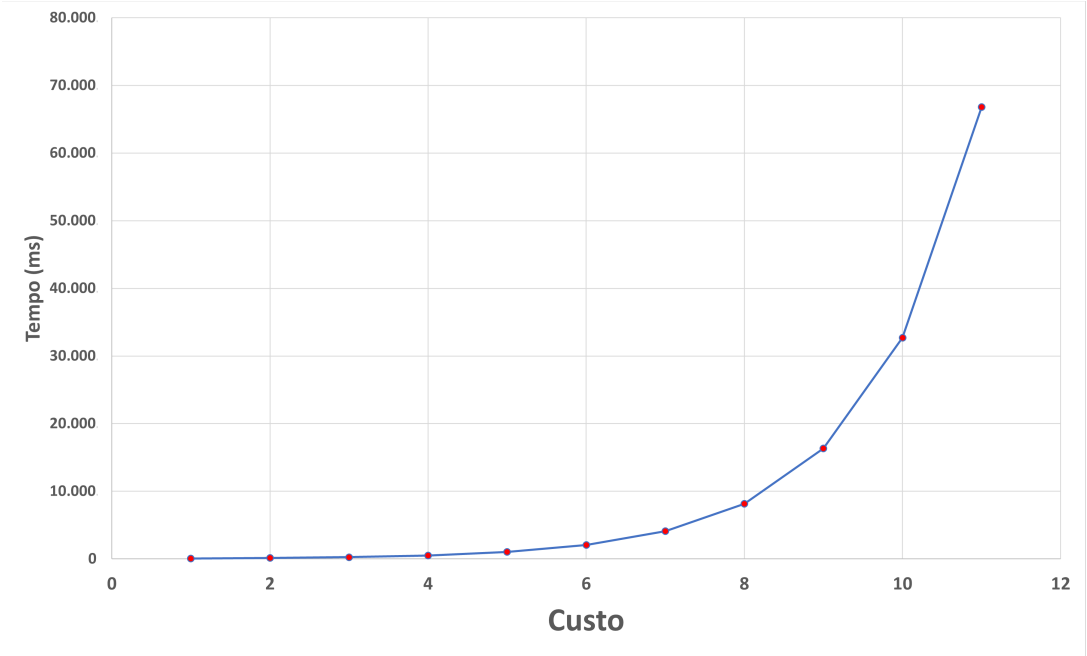

Figura 8 – Gráfico demonstrando o tempo para geração de *hash* X custo.

Fonte: Gerado a partir dos dados de (ARIAS, 2021).

3.2.1.3 Ambiente

O Node.js é um ambiente de tempo de execução JavaScript que fornece uma API de baixo nível para lidar com solicitações HTTP, mas não fornece abstrações de alto nível para criar aplicativos. O Express, por outro lado, é uma estrutura de aplicativo executada sobre o Node.js fornecendo um conjunto de abstrações de alto nível para a construção de aplicativos e foi utilizado dessa forma no projeto para criar o sistema de roteamento que permite lidar com solicitações HTTP recebidas, definir as rotas e *endpoints* do aplicativo. O sistema de *middleware* do Express foi utilizado na autenticação, validação e tratamento de erros.

No diagrama da Figura 9 é possível observar as sequências dos *middlewares* utilizados, o *middleware* de autenticação é responsável por verificar as credenciais do usuário e conceder ou negar acesso ao recurso solicitado. Se o usuário for autenticado, a requisição é passada para o *middleware* de validação, que é responsável por validar os dados na requisição para garantir que eles atendam ao formato requerido. Se ocorrer um erro durante o processo de validação, este é capturado pelo *middleware* de tratamento de erros, que

gera uma resposta ao erro e a envia de volta ao cliente por meio do Node.js. Caso não ocorra erro, a requisição é passada de volta ao Express, que gera uma resposta e a envia de volta ao cliente por meio do Node.js.

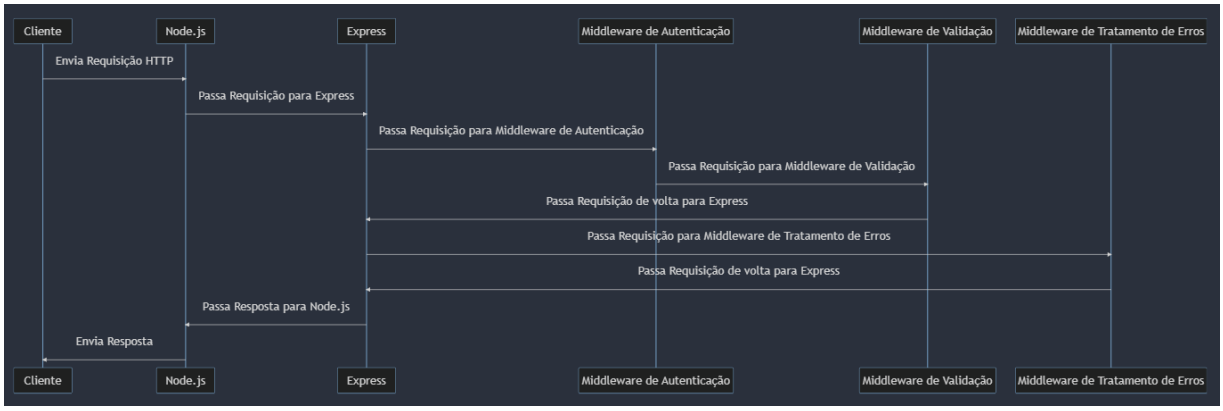

Figura 9 – Sequência de fluxo do Node.js e Express

Para a documentação dos *endpoints* da API foi utilizada a biblioteca Swagger que permite a geração de documentação interativa para APIs RESTful, essa documentação interativa pode ser vista ao rodar o projeto acessando *http://localhost:3333/api-docs/*, como demonstrado na Figura 10. Está anexado nesse trabalho também o documento PDF gerado utilizando a documentação interativa do Swagger no Apêndice (seção 4.1).

Após a finalização do código do *backend*, foi criada uma conta no site do MongoDB Atlas[1] e foi realizada a configuração de conexão para testes. A ferramenta utilizada para testar a API foi o Postman[2], que é uma ferramenta de desenvolvimento de APIs que simplifica o processo de desenvolvimento e teste. O Postman fornece recursos úteis, como solicitações em várias etapas, ambiente e variáveis globais, servidores fictícios, monitoramento e testes automatizados. É importante notar que os testes foram realizados por meio do Postman, uma aplicação terceirizada, e não diretamente na aplicação, uma vez que esta receberá dados do aplicativo móvel que ainda está em desenvolvimento.

Como exemplo, será demonstrada a criação de um novo tipo de jogo. Na Figura 11 é possível visualizar como enviar, por meio do Postman, um objeto JSON, contendo todos os dados necessários, no corpo da requisição para o *endpoint* "`/game`", utilizando o método POST. Observa-se que os campos que constam no Objeto JSON são os mesmos do *Schema* `ResultStructure` da Figura 5.

A Figura 12 ilustra um exemplo de uma resposta do servidor sendo apresentada conforme esperado: contendo *Status* 201 – código de *status* HTTP que indica que uma solicitação foi bem-sucedida e um recurso foi criado –, a mensagem de sucesso (programada) e a indicação que não houve erro.

[1] <https://cloud.mongodb.com/>
[2] <https://www.postman.com>

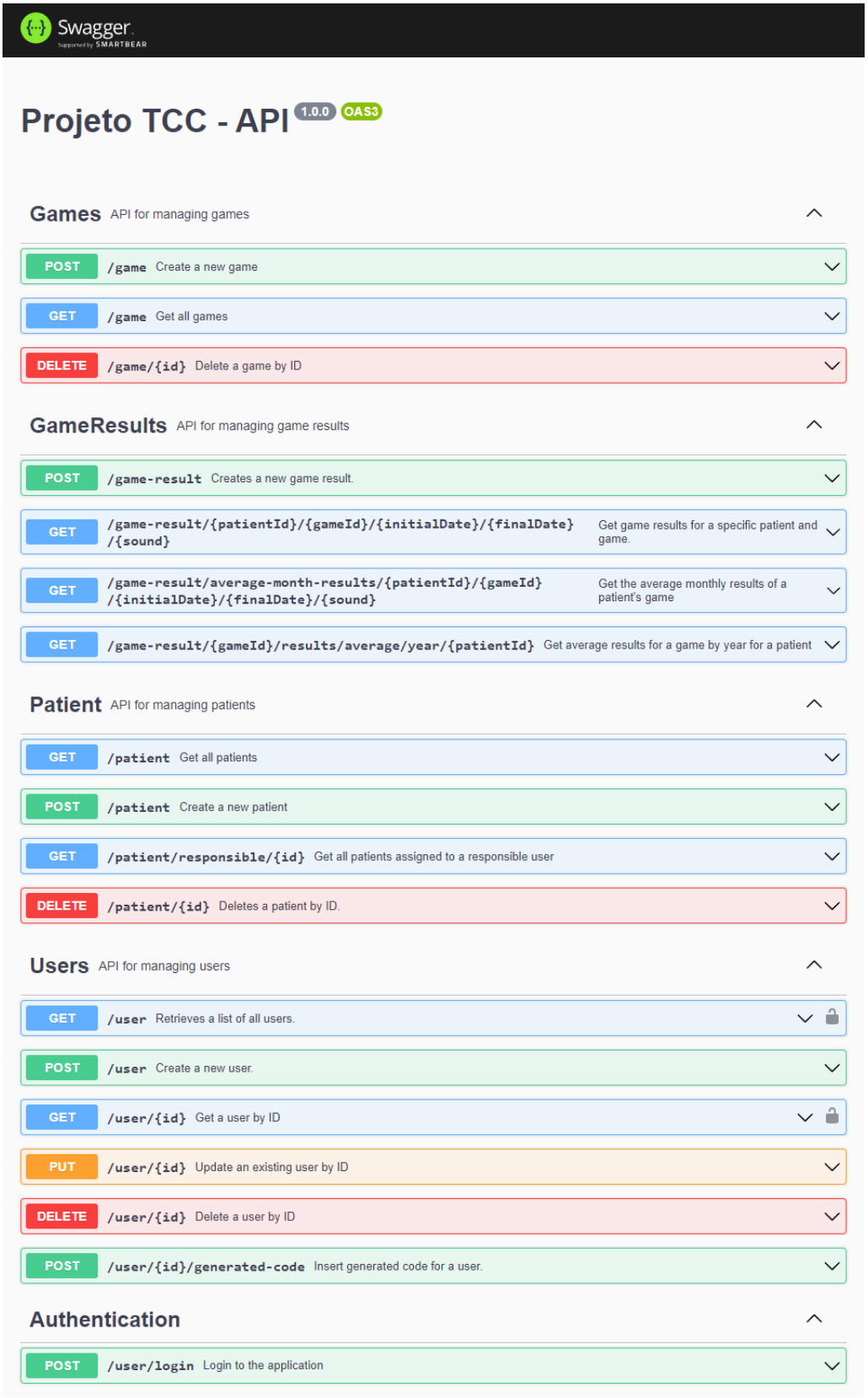

Figura 10 – Documentação interativa gerada pelo Swagger

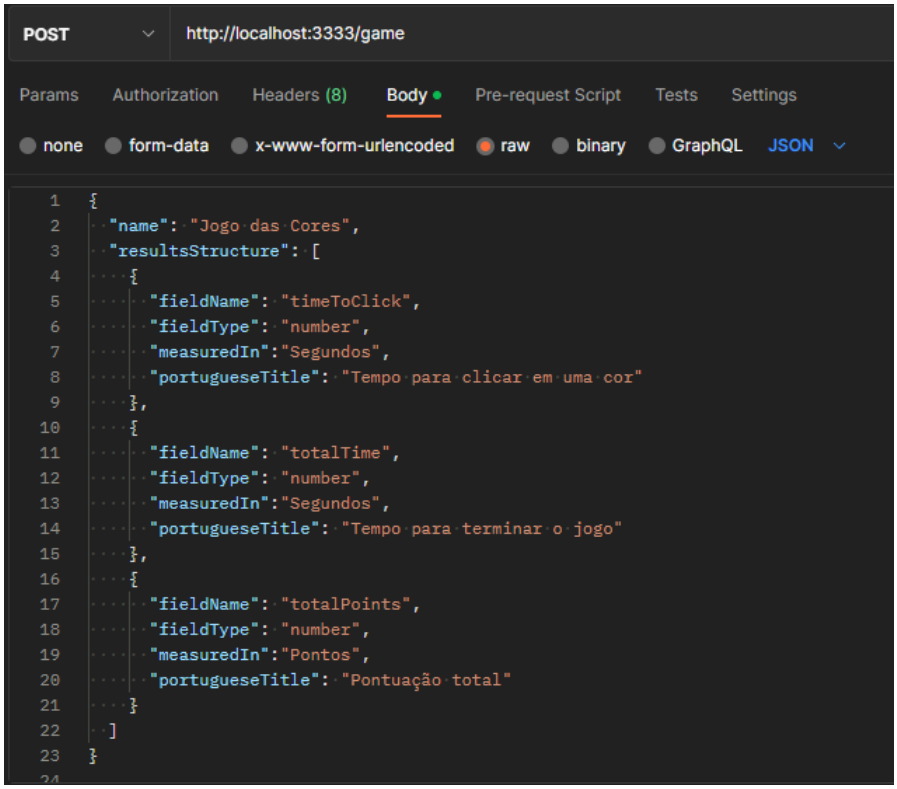

Figura 11 – Exemplo de Objeto JSON enviado pelo Postman.

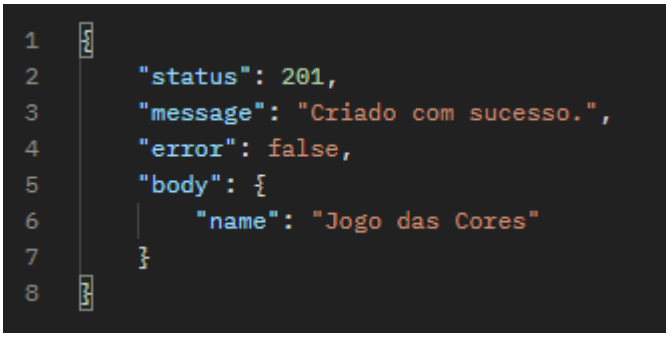

Figura 12 – Exemplo de Objeto JSON recebido pelo Postman.

Para verificar se os dados foram de fato enviados ao servidor, é possível conferir o *cluster* no site do MongoDB Atlas. Conforme mostrado na Figura 13, é possível constatar que o objeto foi armazenado com sucesso no banco de dados.

Caso ocorra algum erro na tentativa do envio do objeto JSON, uma resposta de erro é retornada. Essa resposta contém o código de status HTTP que indica que uma solicitação inválida foi realizada, a mensagem com o erro especificado e a indicação de erro (Figura 14). É importante citar que as mensagens de erro que são devolvidas pelos *middlewares*, foram padronizadas de acordo com o padrão RFC 7807 (NOTTINGHAM; WILDE, 2016), também conhecido como "Detalhes de problemas para APIs HTTP". O uso de um formato de erro padronizado ajuda a garantir que os erros possam ser analisados

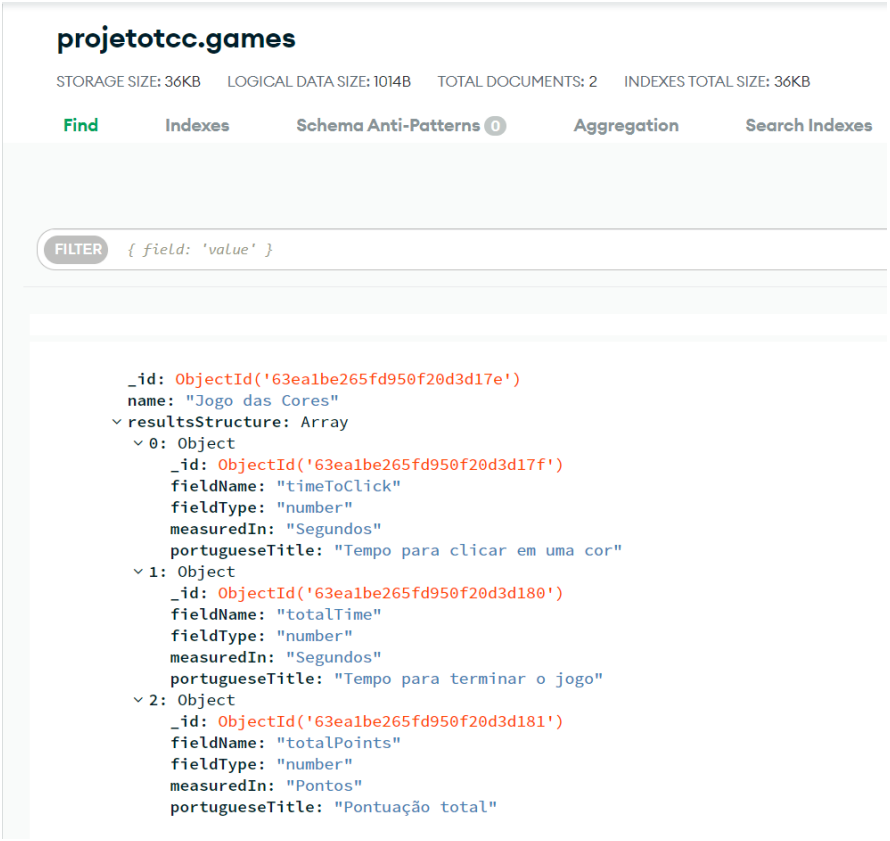

Figura 13 – Objeto JSON armazenado no MongoDB.

e processados de forma consistente pelos clientes de uma API (AMUNDSEN, 2020). Essa padronização será importante, pois esses erros de resposta serão interceptados futuramente pela aplicação web.

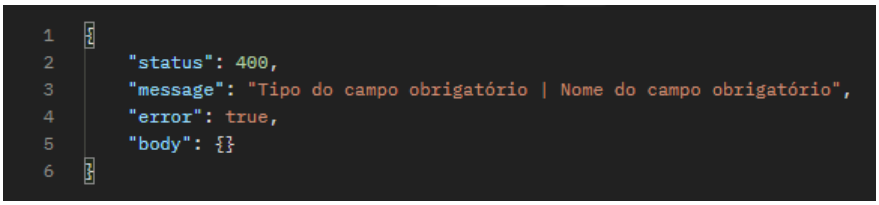

Figura 14 – Exemplo da resposta de erro recebida pelo Postman.

3.2.2 *Frontend*

Nessa seção serão descritas as técnicas e implementações realizadas no *Frontend* para que o Sistema trabalhe conforme descrito. As explicações foram separadas em partes referentes à Testes e Mensagens, Autenticação, Responsividade, Cores e Menus e Utilização do Sistema para melhor entendimento.

3.2.2.1 Testes e Mensagens

Partindo para a interação com o *Frontend*, com o projeto executando localmente pode-se visualizar as telas e testar a conexão com o *Backend*. Desligando o servidor propositalmente para simular uma falha de conexão é possível visualizar o erro de conexão na aba de ferramentas de desenvolvedor do navegador (Figura 15).

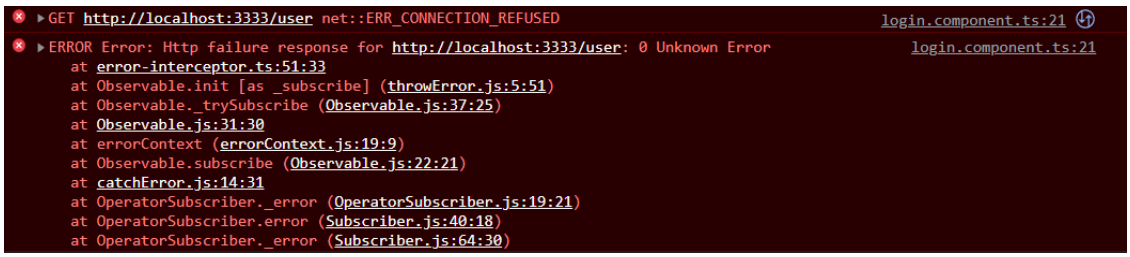

Figura 15 – Exemplo de erro de conexão apresentado no navegador.

Este teste é especialmente importante por permitir visualizar o interceptador de erro do Angular. Esse interceptador de erro foi codificado utilizando-se a biblioteca *Ngx Toastr* sendo utilizada para criar notificações *toast* em Angular como ilustrado na Figura 16. Uma notificação *toast* é uma mensagem que aparece na tela do dispositivo acessado pelo usuário, geralmente em resposta a um evento ou ação realizada pelo usuário. A biblioteca fornece um conjunto de componentes do Sistema predefinidos que podem ser usados para personalizar as mensagens de notificação e, neste caso, foram utilizadas para exibir em tela qualquer erro de conexão com o servidor e respostas de sucesso das entradas do usuário.

A Figura 17 ilustra como os interceptadores de erros foram utilizados para detectar erros ou exceções lançados pelo aplicativo e tratá-los adequadamente, notificando o usuário sobre quaisquer problemas encontrados durante a execução do aplicativo. Na Figura 17 à esquerda está o módulo principal da aplicação, no qual a biblioteca do *Ngx Toastr* foi importada com as configurações padrão de duração da mensagem exibida em tela. Já na Figura 17 à direita, é possível visualizar o componente de interceptação de erro que utiliza a biblioteca para apresentar os erros em tela. É importante destacar que essa programação é considerada uma boa prática, pois fornece informações claras para o usuário em caso de mau funcionamento da aplicação. Sem essa abordagem, o usuário poderia ficar sem saber o motivo pelo qual a aplicação não está funcionando corretamente.

3.2.2.2 Autenticação

O diagrama da Figura 18 representa um fluxo simplificado de como o *token* JWT funciona durante a autenticação. O usuário faz o login com suas credenciais sendo validadas pelo servidor. O servidor gera um *token* JWT contendo uma chave secreta e o retorna ao usuário. Esse *token* contém as informações do usuário sendo armazenado no navegador

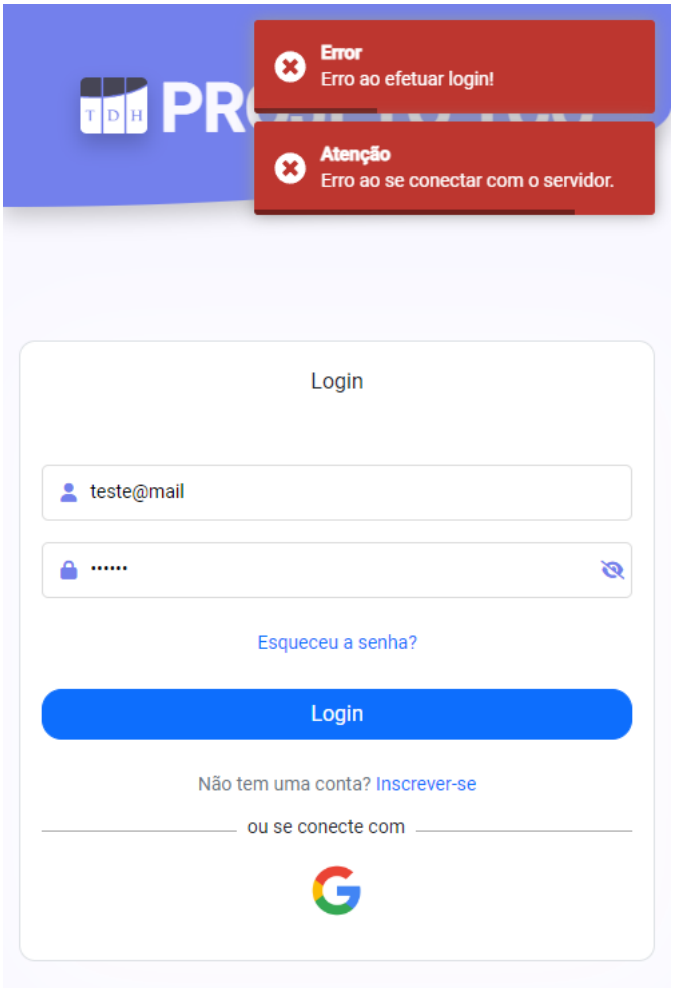

Figura 16 – Exemplo de erro a ser apresentado na tela do usuário.

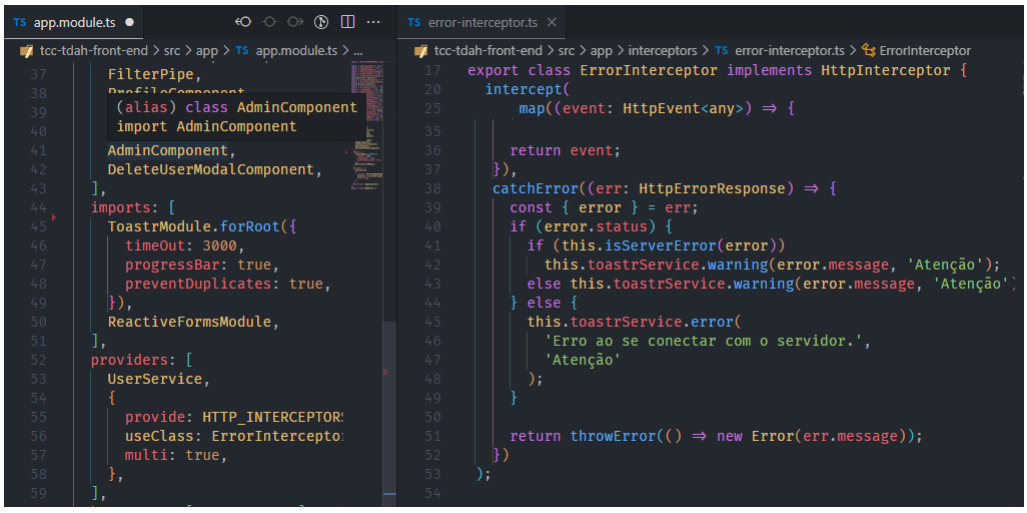

Figura 17 – Códigos fonte da utilização da biblioteca *Ngx Toastr*. `Esquerda`: módulo principal da aplicação e `Direita`: componente de interceptação de erro.

por meio do `localStorage`. O usuário então faz uma solicitação para um recurso protegido, e nesse momento, o servidor de recursos valida o *token* JWT antes de conceder acesso e servir o recurso protegido.

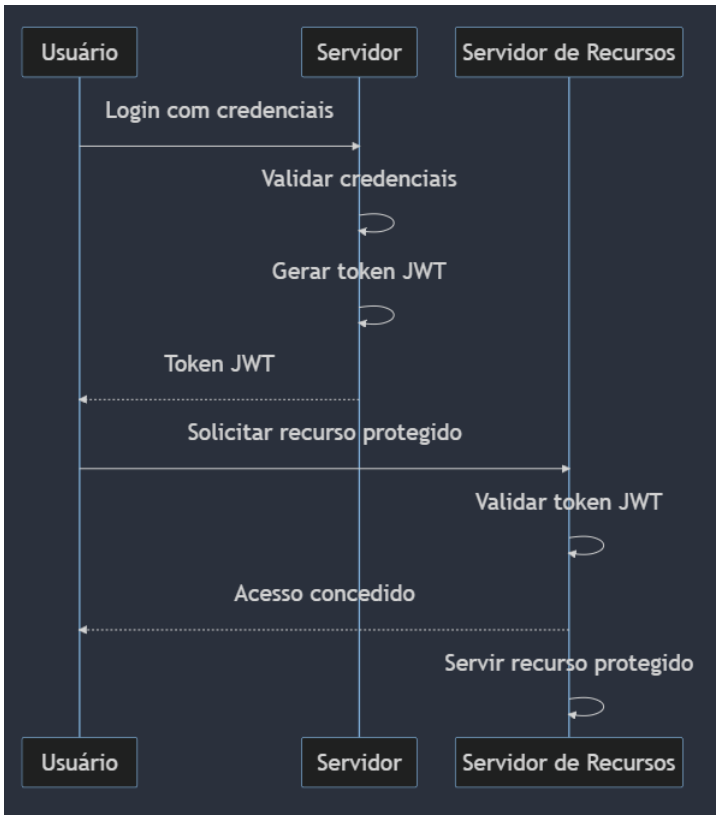

Figura 18 – Funcionamento da autenticação por meio de JWT.

3.2.2.3 Responsividade, Cores e Menus

Na camada de design do Sistema foi utilizada a biblioteca Bootstrap, que forneceu um conjunto abrangente de componentes de interface do usuário pré-projetados, reduzindo o tempo e o esforço necessários para o desenvolvimento, além de HTML e CSS para estruturar e estilizar os componentes e o *layout*.

No quesito responsividade, a biblioteca Bootstrap foi utilizada novamente, em conjunto com CSS *media queries* e lógicas da propriedade de *display* para atingir um resultado responsivo para qualquer tamanho de tela em dispositivos diversificados. Na Figura 19 observa-se o Sistema completo em uma tela grande. Os nome do Sistema está como "Projeto TCC" por ainda não ter sido definido um nome para o aplicativo móvel que está sendo desenvolvido, porém quando este for definido, a alteração nesse Sistema será simples.

A Figura 20 ilustra o comportamento do Sistema em telas menores, como a de dispositivos móveis, sendo adicionado um botão a esquerda que selecionar ou a lista de pacientes ou as informações, conforme o caso.

Figura 19 – Exemplo da Tela Completa do Sistema.

A paleta de cores do Sistema não foi escolhida por acaso. De acordo com a teoria das cores, essas junção de cores remete a calma, lealdade e inteligência (WHITE, 2022). Com isso as cores predominantes são azul, branco e cinza, consideradas cores frias.

A Figura 21 ilustra a tela de login, na qual o usuário poderá se logar utilizando um e-mail e senha, sendo utilizado o método explicado na subseção 3.2.2.2. Caso o usuário não tenha um e-mail e/ou senha cadastrados, ele pode se direcionar até a tela de cadastro (clicando em `inscreva-se`), para que possa cadastrar seu nome, sobrenome, e-mail e senha, as strings dos campos nome e sobrenome serão concatenadas para envio do campo `name` no objeto do banco de dados, após esse passo o usuário fica habilitado a voltar a tela de login e entrar no Sistema.

A tela de cadastro (Figura 22) contém várias verificações que foram feitas utilizando a validação dos formulários reativos do Angular assim, qualquer tipo de informação inserida que não seguir os padrões, apresentará em tela mensagens que indicará o motivo do usuário não poder seguir adiante (Figura 23). Em ambas as telas de login e cadastro se encontra pronto o botão para interação com a conexão com uma conta Google, porém a lógica e comunicação com a API do Google está entre os tópicos que deverão ser feitos em projetos futuros.

Após efetuar o login, o usuário é redirecionado para a tela principal do Sistema. Ao deslocar-se para a extrema direita da página, encontrará um ícone circular com a saudação "Olá", seguida pelo nome do usuário. Ao clicar neste ícone, será exibido um menu suspenso

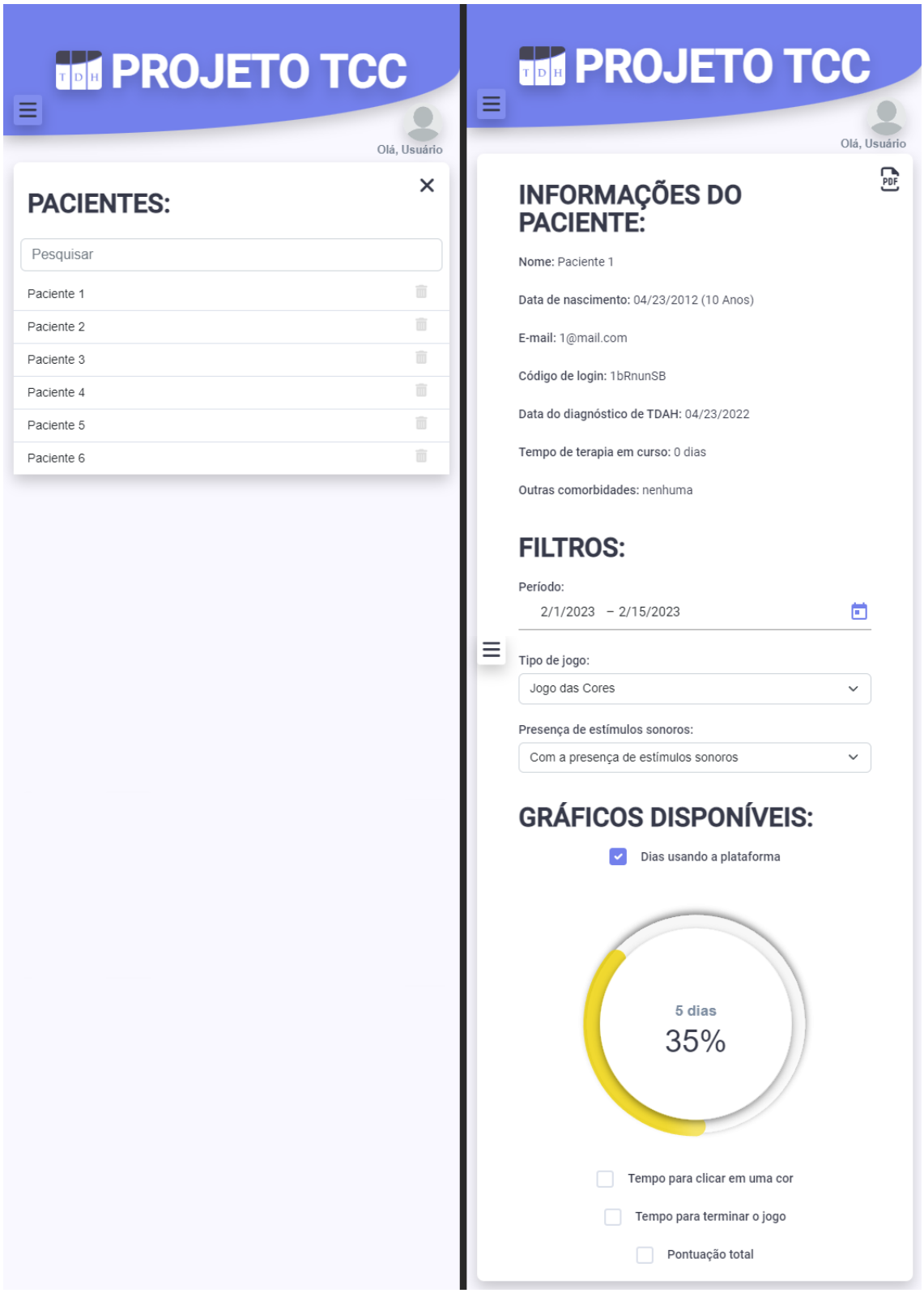

Figura 20 – Exemplo da responsividade do Sistema para dispositivos móveis. Após selecionar o paciente (esquerda), abre-se a tela do lado direito. Ambas as telas são excludentes, ocupando toda a tela do dispositivo móvel.

Figura 21 – Exemplo da Tela de Login de Usuário.

Figura 22 – Exemplo da tela de Inscrição/Cadastro.

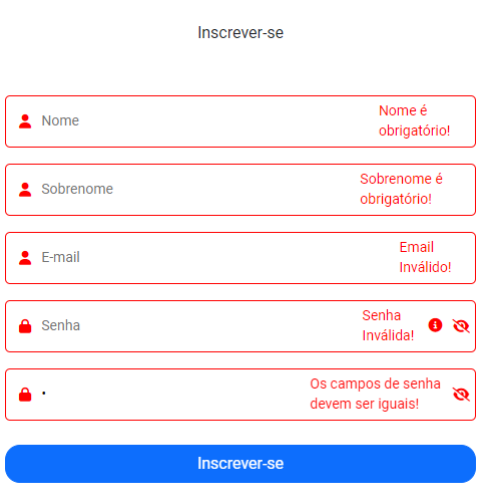

Figura 23 – Exemplo da tela de Inscrição/Cadastro com campos inválidos.

(Figura 24), animado por meio de uma combinação de elementos de lista em HTML e propriedades de transição e transformação em CSS. Neste menu, o usuário terá acesso a uma lista de opções, cada uma levando a uma nova página por meio dos componentes e rotas do Angular. Além disso, nesta mesma área, o usuário poderá efetuar *logout* do Sistema. Vale destacar que, no caso de um usuário com privilégios de administrador, uma opção adicional será exibida, chamada `Administração`.

Figura 24 – Menu suspenso de opções do usuário.

Ao clicar na opção de `Administração`, o usuário com perfil de administrador é encaminhado para uma página com uma lista onde, por exemplo, pode remover outros usuários (Figura 25).

Ao clicar no ícone da lixeira (`deletar`) em um usuário, uma janela de confirmação (Figura 26) aparecerá na tela. Essa janela foi criada utilizando a função de modais do Angular, por meio de um componente separado que usa o encapsulamento da funcionalidade do modal em um componente e serviço, permitindo reutilizar o modal em todo o aplicativo quando uma opção irreversível é mostrada ao usuário.

Outra opção que o usuário pode escolher no menu suspenso é `Gerar Código`. Essa opção leva a página de geração de código de paciente (Figura 27). Esse código é utilizado pelo paciente no momento de seu cadastro no aplicativo móvel. Isso vinculará o paciente a

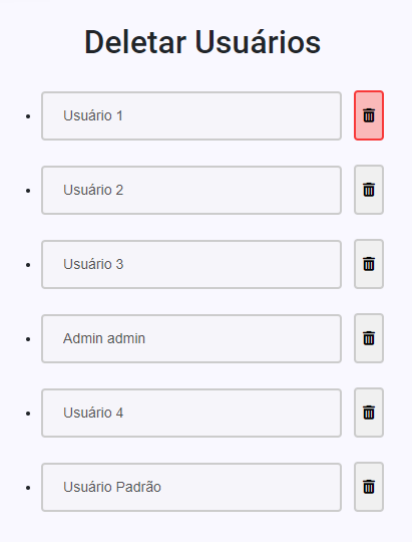

Figura 25 – Exemplo de opções da página de Administração.

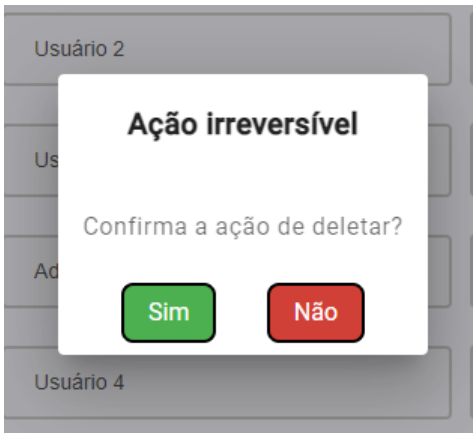

Figura 26 – Modal de ação irreversível.

este usuário/profissional e, a partir desse momento, todas as ações realizadas pelo paciente no aplicativo móvel serão mensuradas e encaminhadas para esse usuário.

O código é gerado de forma aleatória como uma *string* de oito números e letras maiúsculas e minusculas. É importante ressaltar que esse código é armazenado dentro do *array* de *strings* `generatedCodes` no banco de dados no objeto do usuário responsável e, cada vez que um paciente utilizar o código, este é removido desse *array*, tornando-o um código de uso único, o paciente é conectado ao responsável pelo ID através do campo `responsible` no objeto do banco de dados. A proposta inicial é que o paciente realize o cadastramento do código no momento da geração ou seja, durante a consulta.

No menu suspenso, há também a opção de `perfil`, que leva o usuário às configurações de perfil (Figura 28). No perfil, o usuário poderá atualizar seu nome, e-mail, senha e adicionar uma foto. Os formulários reativos do Angular foram programados para ficar desabilitados até que o botão de edição seja pressionado. Para escolher uma foto, o usuá-

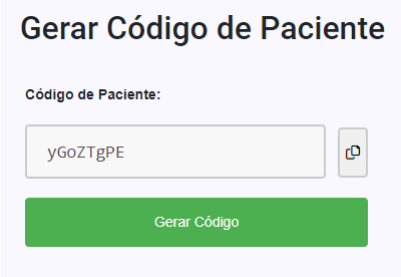

Figura 27 – Exemplo da geração de código para pacientes.

rio pode clicar na opção correspondente e a caixa de diálogo de escolha de arquivo será aberta, aceitando apenas arquivos de imagem. Alternativamente, o usuário pode arrastar uma imagem até a área designada na tela.

Figura 28 – Exemplo das opções da página para editar perfil.

Essa imagem será tratada pela biblioteca *HTML2Canvas* que manterá suas proporções, porém reduzirá a largura e altura para o tamanho ideal do círculo de perfil no qual a foto aparecerá em tela. Após isso, essa imagem será transformada em um código de imagem de Base64, permitindo que ela seja enviada ao servidor como um *string* para ser armazenada. A codificação em Base64 consiste em primeiramente converter a imagem em um formato binário, então esse código binário é dividido em pedaços de seis bits e

cada pedaço é mapeado ao caractere de Código padrão americano para troca de informações (ASCII) correspondente, formando uma longa *string* que pode ser decodificada e renderizada em tela novamente. Na Figura 29 é ilustrada a esquerda uma imagem e a direita uma pequena parte da *string* formada pela conversão em formato Base64, a *string* inteira é demasiadamente longa, então foi ilustrada apenas essa parte.

Figura 29 – Exemplo para a comparação de uma imagem com uma parte de sua codificação em Base64.

A última opção do menu suspenso do usuário, `sair`, faz o *logout*, removendo o *token* JWT previamente armazenado localmente e o redireciona novamente para a tela de login.

3.2.2.4 Utilização do Sistema

Após o login, na página principal, existem duas seções principais. A primeira seção à esquerda apresenta uma lista de todos os pacientes cadastrados e vinculados ao usuário logado (Figura 30). Ao passar o mouse sobre cada nome, o item é destacado e um ícone de exclusão é exibido à direita. Ao clicar no ícone de exclusão, uma janela de confirmação de exclusão será exibida na tela, a mesma janela já apresentada na Figura 26.

Essa lista é gerada dinamicamente com a resposta do servidor por meio da função `ngFor` do Angular que replica um bloco de código HTML com um laço de repetição. Nessa lista também está codificado um *pipe* – um código de transformação – que permite não só a pesquisa e seleção de pacientes, como também, comprime a lista com os resultados e destaca as letras inseridas compatíveis de cada nome.

A outra seção da página principal contém as `informações de paciente` que são preenchidas dinamicamente com os resultados do paciente selecionado na lista. Nesse ponto, não são exibidas as informações e `filtros` até que um paciente seja selecionado. A exibição dos `gráficos disponíveis` somente ocorre quando os `filtros` forem válidos. Antes da seleção do paciente essa parte aparece conforme a Figura 31.

Os `filtros` (Figura 32) são compostos por: um período que pode ser selecionado por meio de um calendário de intervalo de data, o tipo de jogo e a presença de estímulos sonoros. Cada alteração nos `filtros` desencadeia a chamada de uma função que verifica

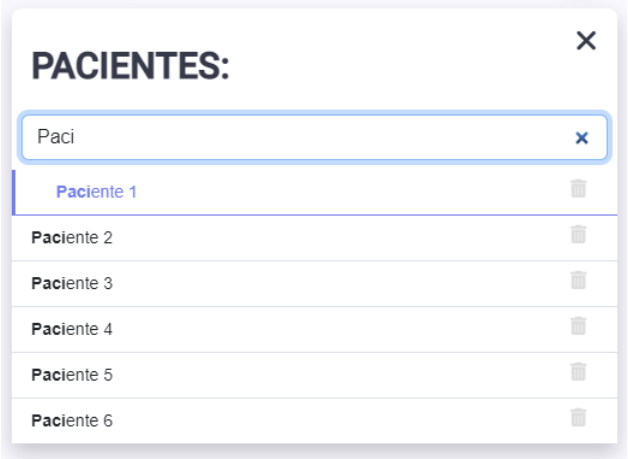

Figura 30 – Exemplo da lista gerada dinâmicamente de Pacientes (versão para dispositivos móveis).

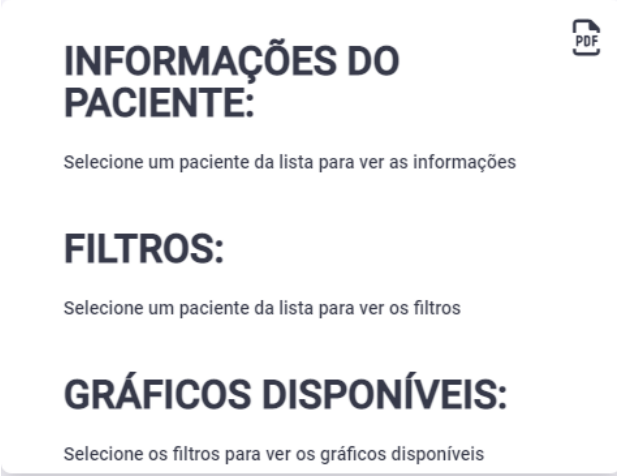

Figura 31 – Informações do paciente sem paciente selecionado (versão para dispositivos móveis).

se os `filtros` são válidos e, se forem, realiza a requisição dos resultados ao *backend* e altera a página de acordo com o que foi recebido.

Depois de selecionar os `filtros`, o usuário tem a opção de escolher quais gráficos serão exibidos na tela. Cada título de gráfico disponível é acompanhado de uma caixa de seleção no lado esquerdo, que pode ser marcada ou desmarcada conforme a preferência do usuário. Os gráficos são gerados dinamicamente a partir dos resultados do *backend*, que agrupa as informações por período (mês ou ano), caso o período selecionado abranja muitos meses.

Os gráficos disponíveis, exemplificados na Figura 33, são configuráveis no *backend*, permitindo que sejam cadastradas quaisquer quantidades de gráficos, devendo ser preenchido cada um com suas rspectivas informações: nome, informações a serem exibidas e a medida s ser utilizada. A estrutura flexível do *backend* permite que os gráficos sejam

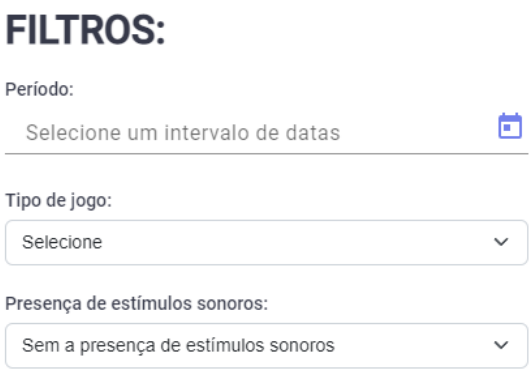

Figura 32 – Exemplo de filtros para seleção (versão para dispositivos móveis).

adicionados dinamicamente na tela, sem limitações no *frontend*. A biblioteca ApexCharts é utilizada para renderizar os gráficos que são adicionados ou removidos da tela conforme a seleção do usuário por meio das funções *render* e *destroy* da biblioteca.

Figura 33 – Exemplo de gráficos disponíveis gerados dinamicamente.

O gráfico de `Dias usando a plataforma` (Figura 34) é uma exceção aos outros gráficos, pois é fixo para qualquer jogo selecionado, o bloco de HTML que gera o gráfico não é inserido dinamicamente pelo *backend*, porém também tem suas informações preenchidas pelo *backend*. Trata-se de um gráfico circular que calcula a porcentagem de dias em que o paciente utilizou a plataforma no período selecionado através das datas que o

paciente enviou resultados. A cor do gráfico varia de vermelho a verde, indicando a proximidade com o percentual máximo de uso da plataforma. Nesse caso, o termo `plataforma` refere-se ao aplicativo móvel.

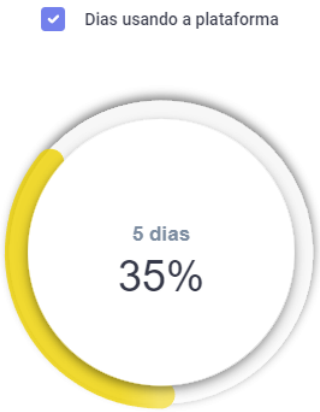

Figura 34 – Exemplo de gráfico de dias utilizando a plataforma.

Juntamente às informações do paciente, à direita, após selecionar todas as opções desejadas, o usuário pode clicar no símbolo do PDF e baixar todas as informações selecionadas no formato PDF para utilizar *offline* ou, por qualquer outro motivo que desejar. Vale ressaltar que o usuário pode, a qualquer momento, encontrar essas informações no Sistema e, fazer o download é apenas uma opção extra.

Para salvar o arquivo PDF foi utilizada a biblioteca *PDF.js* que permite aos desenvolvedores renderizar, manipular e anotar documentos PDF em qualquer aplicativo baseado em Javascript (ENGVALL; MUNTHE-DAHL, 2022). Neste caso a biblioteca foi utilizada juntamente com a biblioteca *HTML2Canvas*, com o código demonstrado na Figura 35, para converter o bloco de HTML para uma imagem e inserir dentro do documento PDF, que pode ser salvo com a data do dia em questão, no disco rígido do usuário.

```
async downloadPdf() {
  if (this.filtersForm.valid) {
    const data = document.getElementById('main');
    let canvas: any;
    if (data) {
      canvas = await html2canvas(data, { scale: 2 });
    }

    const pdf = new jsPDF('p', 'mm', 'a4');
    pdf.addImage(canvas.toDataURL('image/png'), 'PNG', 0, 0, 211, 298);
    pdf.save(`Relatório-${new Date().toLocaleDateString()}.pdf`);
  } else {
    this.toastr.warning(
      'Selecione um paciente e preencha os filtros para gerar o relatório'
    );
  }
}
```

Figura 35 – Código referente a função que salva o PDF.

4 Conclusão

Esse trabalho apresentou uma solução web para auxiliar profissionais que necessitam analisar dados de pacientes portadores de TDAH, mais especificamente, profissionais que utilizarão o aplicativo em desenvolvimento para auxiliar no treinamento cognitivo focado em crianças com TDAH.

Após a visualização da aplicação *frontend* e da API do *backend* trabalhando em conjunto, foi possível verificar o Sistema integrado e trabalhando em conjunto, interceptando e tratando erros, salvando dados e principalmente um Sistema que tem uma grande utilidade.

A API do *backend* foi apresentada com código altamente dinâmico e de fácil alteração e manutenção, podendo trazer um escalabilidade ao projeto caso a implementação de novas funções se mostrem necessárias, tanto por outros programadores quanto por estudantes que queiram contribuir com o projeto.

A interface visual ficou responsiva e dinâmica e preparada para receber diferentes as informações da API. Ela cumpre com o papel de facilitar o diagnóstico e análise de melhora do paciente pelo responsável trazendo uma enorme quantidade de dados em forma gráfica, tornado fácil a observação e comparação dos mesmos.

Os experimentos e testes realizados puderam comprovar e validar as hipóteses de que era possível construir uma solução para a necessidade da análise de dados, os quais, vindos de um dispositivo móvel, mostraram que por meio de um projeto *Full Stack* de desenvolvimento Javascript foi possível chegar a essa resposta, com um projeto com dados armazenados com a segurança do Bcrypt, dinâmico e responsivo.

Manter todos esses dados em um banco de dados, também trás a possibilidade de estudos de caso de milhares de pacientes simultaneamente e também a opção de avaliar e guardar esses dados por muitos anos sem perda.

Algumas limitações a se ressaltar no projeto foram:

- Devido a falta do aplicativo móvel para geração de dados dos pacientes, todos os dados tiveram que ser fabricados e não houve uma maneira de testar a comunicação com esse aplicativo, demonstrando apenas uma simulação de como acontecerá.
- Para a comunicação com o aplicativo móvel que conterá os treinos cognitivos, o desenvolvedor da aplicação móvel deverá estudar a API e a solução web para utilizar de acordo com a sua necessidade.
- Por não tem como prever todos os endpoints e JSONs que serão necessários durante

a comunicação das aplicações, talvez pequenas mudanças e manutenções no código serão necessárias.

4.1 Trabalhos Futuros

Como proposta para trabalhos futuros destacam-se:

- Fazer o *build* da API e da aplicação web Angular e alocar em um servidor para uso público.
- Fazer uma conexão de login e inscrição ao site por meio de APIs de contas de e-mail como a do Google, ou de redes sociais como o Facebook, facilitando ainda mais a criação de contas no aplicativo web.
- Implementar um sistema de recuperação de senha pelo e-mail cadastrado.
- Fazer um estudo sobre a velocidade de armazenamento e recuperação de dados em grande volume das aplicações.
- Conduzir uma avaliação da UX para se certificar que a opinião de especialistas sobre a acessibilidade e usabilidade do Sistema.

Referências

ACCOMAZZO, A.; MURRAY, N.; LERNER, A. **Fullstack React: The Complete Guide to ReactJS and Friends**. [S.l.]: Fullstack.io, 2017. ISBN 9780991344628. Citado na página 19.

AGANS, D. **Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems**. [S.l.]: AMACOM, 2002. ISBN 9780814426784. Citado na página 16.

ALAZAB, M.; ALAZAB, M.; SHALAGINOV, A.; MESLEH, A.; AWAJAN, A. Intelligent mobile malware detection using permission requests and api calls. **Future Generation Computer Systems**, v. 107, p. 509–521, 2020. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X19321223>. Citado na página 32.

AMUNDSEN, M. **Design and Build Great Web APIs**. [S.l.]: Pragmatic Bookshelf, 2020. ISBN 9781680508130. Citado na página 42.

ARIAS, D. **Hashing in Action: Understanding bcrypt**. 2021. Disponível em: <https://auth0.com/blog/hashing-in-action-understanding-bcrypt/>. Citado na página 38.

BADII, C.; BELLINI, P.; CENNI, D.; DIFINO, A.; NESI, P.; PAOLUCCI, M. Analysis and assessment of a knowledge based smart city architecture providing service apis. **Future Generation Computer Systems**, v. 75, p. 14–29, 2017. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X17302273>. Citado na página 32.

BARKLEY, R. A. History of ADHD. **Attention-deficit hyperactivity disorder: A handbook for diagnosis and treatment**, The Guilford Press, p. 3–50, 2015. Disponível em: <https://psycnet.apa.org/record/2014-57877-001>. Citado na página 13.

BARROSO, S. M.; PEREIRA, F. E.; LOPES, D. G.; MACHADO, J. R.; JÚNIOR, J. H. C. Treinamento cognitivo de atenção e memória de universitários com jogos eletrônicos. **Psico**, v. 50, n. 4, p. e29466, 2019. Disponível em: <https://revistaseletronicas.pucrs.br/index.php/revistapsico/article/view/29466>. Citado na página 14.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. [S.l.]: Pearson Education, 2012. ISBN 9780321815736. Citado na página 19.

BELL, P.; BEER, B. **Introducing GitHub: A Non-technical Guide**. [S.l.]: O'Reilly, 2014. ISBN 9781491949740. Citado na página 17.

BRADLEY, S. **Design Patterns in TypeScript: Common GoF (Gang of Four) Design Patterns Implemented in TypeScript**. [S.l.]: Amazon Digital Services LLC - Kdp, 2021. (Software Engineering). ISBN 9798747711983. Citado na página 20.

BRADSHAW, S.; CHODOROW, K.; BRAZIL, E. **MongoDB: The Definitive Guide : Powerful and Scalable Data Storage**. [S.l.]: O'Reilly Media, Incorporated, 2019. (The expert's voice in open source). ISBN 9781491954461. Citado na página 22.

BROWN, E. **Web Development with Node and Express: Leveraging the JavaScript Stack**. [S.l.]: O'Reilly Media, 2014. ISBN 9781491902301. Citado na página 23.

BUTTIGIEG, S.; JEVDJENIC, M. **Learning Node.js for Mobile Application Development**. [S.l.]: Packt Publishing, 2015. ISBN 9781782175049. Citado na página 21.

DANIELS, P.; ATENCIO, L. **RxJS in Action**. [S.l.]: Manning, 2017. ISBN 9781617293412. Citado na página 26.

DUCKETT, J. **HTML and CSS: Design and Build Websites**. John Wiley & Sons, 2011. Disponível em: <https://wtf.tw/ref/duckett.pdf>. Citado na página 18.

_____. **JavaScript and jQuery: Interactive Front-End Web Development**. [S.l.]: Wiley, 2014. ISBN 9781118531648. Citado na página 29.

_____. **Web Design with HTML, CSS, JavaScript and jQuery Set**. [S.l.]: John Wiley & Sons, 2014. Citado 2 vezes nas páginas 18 e 29.

EDWARD, S.; SABHARWAL, N. **Practical MongoDB: Architecting, Developing, and Administering MongoDB**. [S.l.]: Apress, 2015. (Expert's voice in open source). ISBN 9781484206478. Citado na página 22.

ENGVALL, V. N.; MUNTHE-DAHL, M. **Internship in web development at ØB Innovation**. Dissertação (B.S. thesis) — Høgskolen i Molde-Vitenskapelig høgskole i logistikk, 2022. Disponível em: <https://himolde.brage.unit.no/himolde-xmlui/handle/11250/3023913>. Citado na página 55.

FAIN, Y.; MOISEEV, A. **Angular 2 Development with TypeScript**. [S.l.]: CreateSpace Independent Publishing Platform, 2017. ISBN 9781548494711. Citado na página 23.

FARHI, O. **Reactive Programming with Angular and ngrx: Learn to Harness the Power of Reactive Programming with RxJS and ngrx Extensions**. [S.l.]: Apress, 2017. ISBN 9781484226209. Citado na página 27.

FERGUSON, N.; SCHNEIER, B.; KOHNO, T. **Cryptography Engineering: Design Principles and Practical Applications**. [S.l.]: Wiley, 2010. ISBN 9780470474242. Citado na página 30.

FIELDING, R. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) — University of California, Irvine, 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Citado na página 24.

FLANAGAN, D. **JavaScript: The Definitive Guide**. [S.l.]: O'Reilly Media, Incorporated, 2011. (Definitive Guide Series). ISBN 9780596805524. Citado na página 20.

FRAIN, B. **Responsive Web Design with HTML5 and CSS3**. Packt Publishing Ltd, 2014. Disponível em: <https://www.packtpub.com/product/responsive-web-design-with-html5-and-css3/9781849693189>. Citado na página 18.

FREEMAN, A. **Pro Angular 6**. [S.l.]: Apress, 2018. ISBN 9781484236482. Citado na página 23.

_____. **Pro Angular 9: Build Powerful and Dynamic Web Apps**. [S.l.]: Apress, 2020. ISBN 9781484259979. Citado na página 28.

FREEMAN, E.; ROBSON, E.; BATES, B.; SIERRA, K. **Head First Design Patterns: A Brain-Friendly Guide**. [S.l.]: O'Reilly Media, Incorporated, 2004. (Head First Series). ISBN 9780596800741. Citado na página 17.

GROSSKURTH, A.; GODFREY, M. W. A reference architecture for web browsers. In: IEEE. **21st IEEE International Conference on Software Maintenance (ICSM'05)**. 2005. p. 661–664. Disponível em: <https://ieeexplore.ieee.org/document/1510168>. Citado na página 19.

GUHA, A.; SAFTOIU, C.; KRISHNAMURTHI, S. The essence of javascript. In: SPRINGER. **European conference on Object-oriented programming**. 2010. p. 126–150. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-642-14107-2_7>. Citado na página 19.

HACKS, M. **Pdf.js: A Portable Document Format (PDF) Library for the Web**. 2013. [Online; acessado em 30 de janeiro de 2023]. Disponível em: <https://hacks.mozilla.org/2013/06/pdfjs/>. Citado na página 29.

HAVERBEKE, M. **Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming**. [S.l.]: No Starch Press, 2018. ISBN 9781593279509. Citado na página 20.

HOLMES, S.; HARBER, C. **Getting MEAN with Mongo, Express, Angular, and Node**. [S.l.]: Manning, 2019. ISBN 9781638355243. Citado na página 22.

HORTON, S.; QUESENBERY, W. **A Web for Everyone: Designing Accessible User Experiences**. Rosenfeld Media, 2014. Disponível em: <https://rosenfeldmedia.com/books/a-web-for-everyone/>. Citado na página 18.

HUGHES-CROUCHER, T.; WILSON, M. **Node: Up and Running: Scalable Server-Side Code with JavaScript**. [S.l.]: O'Reilly Media, Incorporated, 2012. (O'Reilly Series). ISBN 9781449398583. Citado na página 21.

HUNT, A.; THOMAS, D. **The Pragmatic Programmer: From Journeyman to Master**. [S.l.]: Pearson Education, 1999. ISBN 9780132119177. Citado na página 16.

IHRIG, C.; BRETZ, A. **Full Stack JavaScript Development With MEAN: MongoDB, Express, AngularJS, and Node.JS**. [S.l.]: SitePoint, 2014. ISBN 9781457192340. Citado na página 19.

IMSIROVIC, A. **Bootstrap 4 Cookbook**. [S.l.]: Packt Publishing, 2017. ISBN 9781785888762. Citado na página 27.

Italian Ministry of Education, Universities and Research. **Governing the smart city**. n.d. <https://sites.unica.it/ghost/home/>. [Online; acessado em 18 de março de 2023]. Citado na página 33.

JIN, B.; SAHNI, S.; SHEVAT, A. **Designing Web APIs: Building APIs That Developers Love**. [S.l.]: O'Reilly Media, 2018. ISBN 9781492026877. Citado na página 13.

JOBSEN, B.; COCHRAN, D.; WHITLEY, I. **Bootstrap 4 Site Blueprints**. [S.l.]: Packt Publishing, 2016. ISBN 9781785881497. Citado na página 27.

KEIM, D. A.; MANSMANN, F.; SCHNEIDEWIND, J.; ZIEGLER, H. Challenges in visual data analysis. In: IEEE. **Tenth International Conference on Information Visualisation (IV'06)**. 2006. p. 9–16. Disponível em: <https://ieeexplore.ieee.org/document/1648235>. Citado na página 14.

Km4City. **Open Urban Platform for a Sentient Smart City**. n.d. <https://www.km4city.org/>. [Online; acessado em 18 de março de 2023]. Citado na página 33.

KOZLOWSKI, P. **Mastering Web Application Development with AngularJS**. [S.l.]: Packt Publishing, 2013. (Community experience distilled). ISBN 9781782161837. Citado na página 29.

KUNZ, G. **Mastering Angular Components: Build component-based user interfaces using Angular, 2nd Edition**. [S.l.]: Packt Publishing, 2018. ISBN 9781788295581. Citado na página 28.

L, R. **Express.Js: Guide Book on Web Framework for Node.Js**. [S.l.]: CreateSpace Independent Publishing Platform, 2016. ISBN 9781533320018. Citado na página 22.

LINGRAS, P.; TRIFF, M.; LINGRAS, R. **Building Cross-Platform Mobile and Web Apps for Engineers and Scientists: An Active Learning Approach**. [S.l.]: Cengage Learning, 2016. ISBN 9781305855892. Citado na página 29.

MAK, A. D. P.; LEE, S.; SAMPSON, N. A.; ALBOR, Y.; ALONSO, J.; AUERBACH, R. P.; BAUMEISTER, H.; BENJET, C.; BRUFFAERTS, R.; CUIJPERS, P.; EBERT, D. D.; GUTIERREZ-GARCIA, R. A.; HASKING, P.; LAPSLEY, C.; LOCHNER, C.; KESSLER, R. C. Adhd comorbidity structure and impairment: Results of the who world mental health surveys international college student project (wmh-ics). **Journal of Attention Disorders**, SAGE Publications Inc, v. 26, n. 8, p. 1078–1096, 2021. Disponível em: <https://journals.sagepub.com/doi/abs/10.1177/1087054711057275>. Citado na página 13.

MAKSIMOVA, R.; KOLEV, K. Information graphers-covid-19 monitor. In: IEEE. **2020 International Conference Automatics and Informatics (ICAI)**. 2020. p. 1–4. Disponível em: <https://ieeexplore.ieee.org/document/9311326>. Citado na página 30.

MANSILLA, S. **Reactive Programming with RxJS 5: Untangle Your Asynchronous JavaScript Code**. [S.l.]: Pragmatic Bookshelf, 2018. ISBN 9781680505535. Citado na página 27.

MARDAN, A. **Express.js Guide: The Comprehensive Book on Express.js**. [S.l.]: Azat Mardan, 2014. Citado na página 23.

MARTIN, R. **Clean Code: A Handbook of Agile Software Craftsmanship**. [S.l.]: Pearson Education, 2008. (Robert C. Martin Series). ISBN 9780136083252. Citado na página 16.

MEYER, E. A. **CSS: The Definitive Guide**. O'Reilly Media, Inc., 2011. Disponível em: <https://www.oreilly.com/library/view/css-the-definitive/9781098117603/>. Citado na página 18.

MISTRÍK, I.; GRUNDY, J.; HOEK, A. van der; WHITEHEAD, J. **Collaborative Software Engineering**. [S.l.]: Springer Berlin Heidelberg, 2010. ISBN 9783642102936. Citado na página 17.

NANCE, C. **TypeScript Essentials**. [S.l.]: Packt Publishing, 2014. (Community experience distilled). ISBN 9781783985777. Citado na página 20.

NIRGUDKAR, N.; SINGH, P. The mean stack. **International Research Journal of Engineering and Technology**, p. 2395–56, 2017. Disponível em: <https://tinyurl.com/5n8d4y9z>. Citado na página 24.

NOTTINGHAM, M.; WILDE, E. **Problem Details for HTTP APIs**. RFC Editor, 2016. RFC 7807. (Request for Comments, 7807). Disponível em: <https://www.rfc-editor.org/info/rfc7807>. Citado na página 41.

NUGRAHA, D.; AHMED, F. Y. H. Mean stack to enhance the advancement of parking application: A narrative review. **Journal of Physics: Conference Series**, IOP Publishing, v. 1167, n. 1, p. 012075, feb 2019. Disponível em: <https://dx.doi.org/10.1088/1742-6596/1167/1/012075>. Citado na página 31.

OXLEY, T.; RAJLICH, N.; HOLOWAYCHUK, T.; YOUNG, A. **Node.js in Action**. [S.l.]: Manning, 2017. ISBN 9781638355175. Citado na página 21.

RAYMOND, E. **The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary**. [S.l.]: O'Reilly Media, 2001. (O'Reilly Series). ISBN 9780596001087. Citado na página 17.

Research and innovation - European Commission. **Horizon 2020**. 2020. <https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-2020_en>. [Online; acessado em 18 de março de 2023]. Citado na página 33.

RUIZ-MANRIQUE, G.; TAJIMA-POZO, K.; MONTAÑES-RADA, F. Case report:"adhd trainer": the mobile application that enhances cognitive skills in adhd patients. **F1000Research**, Faculty of 1000 Ltd, v. 3, 2014. Disponível em: <https://f1000research.com/articles/3-283/v1>. Citado na página 14.

RUIZ-MANRIQUE, G.; TAJIMA-POZO, K.; MONTAÑÉS-RADA, F. Case report: "adhd trainer": the mobile application that enhances cognitive skills in adhd patients. **F1000Research**, v. 3, p. 283, 2014. Disponível em: <https://f1000research.com/articles/3-283/v1>. Citado 2 vezes nas páginas 31 e 32.

SIKOS, L. **Web Standards: Mastering HTML5, CSS3, and XML**. [S.l.]: Apress, 2014. ISBN 9781484208847. Citado na página 29.

SPURLOCK, J. **Bootstrap: Responsive Web Development**. [S.l.]: O'Reilly Media, 2013. ISBN 9781449344597. Citado na página 27.

STUTTARD, D.; PINTO, M. **The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws**. [S.l.]: Wiley, 2011. ISBN 9781118175248. Citado na página 30.

TECHNOLOGY, B. **What's New in Angular 15**. Bacancy Technology, 2023. [Online; acessado 30 de janeiro 2023]. Disponível em: <https://www.bacancytechnology.com/blog/whats-new-in-angular-15>. Citado na página 24.

UZAYR, S. **Mastering Git: A Beginner's Guide**. [S.l.]: CRC Press, 2022. (Mastering Computer Science). ISBN 9781000552911. Citado na página 17.

WHITE, A. **The Elements of Graphic Design: Space, Unity, Page Architecture, & Type**. [S.l.]: Allworth, 2022. ISBN 9781621537625. Citado na página 46.

YUK, M.; DIAMOND, S. **Data Visualization For Dummies**. [S.l.]: Wiley, 2014. (–For dummies). ISBN 9781118502921. Citado na página 28.

ZHU, N. **Data Visualization with D3.js Cookbook**. [S.l.]: Packt Publishing, 2013. ISBN 9781782162179. Citado na página 28.

Apêndices

Projeto TCC - API

Overview

Tags

Games

API for managing games

GameResults

API for managing game results

Patient

API for managing patients

Users

API for managing users

Paths

GET `/game` Get all games

Responses

Code	Description	Links
200	OK	No Links
204	No Content	No Links
500	Internal Server Error	No Links

POST `/game` Create a new game

Responses

Code	Description	Links
201	Created	No Links
500	Internal Server Error	No Links

DELETE `/game/{id}` Delete a game by ID

Parameters

Type	Name	Description	Schema
path	**id** *required*	The ID of the game to delete	string (uuid)

Responses

Code	Description	Links
200	OK	No Links
204	No Content	No Links
500	Internal Server Error	No Links

POST `/game-result` Creates a new game result.

Creates a new game result.

Responses

Code	Description	Links
201	The created game result. *Content* **application/json**	No Links
500	Internal server error. *Content* **application/json**	No Links

GET `/game-result/{patientId}/{gameId}/{initialDate}/{finalDate}/{sound}` Get game results for a specific patient and game.

Retrieves all game results for a specific patient and game within a date range, filtered by sound.

Parameters

Type	Name	Description	Schema
path	**gameId** *required*	The ID of the game to retrieve results for.	string
path	**finalDate** *required*	The final date of the range to retrieve results for.	string (date)
path	**patientId** *required*	The ID of the patient to retrieve game results for.	string

Type	Name	Description	Schema
path	**sound** *optional*	The sound to filter the results by (optional).	string
path	**initialDate** *required*	The initial date of the range to retrieve results for.	string (date)

Responses

Code	Description	Links
200	The list of game results for the specified patient and game. *Content* **application/json**	No Links
204	The requested game results were not found.	No Links
500	An error occurred while processing the request.	No Links

GET `/game-result/average-month-results/{patientId}/{gameId}/{initialDate}/{finalDate}/{sound}` Get the average monthly results of a patient's game

Retrieve the average monthly results of a patient's game for a given period of time.

Parameters

Type	Name	Description	Schema
path	**gameId** *required*	The ID of the game to retrieve the results for	string
path	**finalDate** *required*	The end date of the period to retrieve the results for (YYYY-MM-DD)	string (date-time)
path	**patientId** *required*	The ID of the patient to retrieve the results for	string
path	**sound** *required*	The sound to retrieve the results for (e.g. 'sound1', 'sound2')	string
path	**initialDate** *required*	The start date of the period to retrieve the results for (YYYY-MM-DD)	string (date-time)

Responses

Code	Description	Links
200	The average monthly results for the given patient and game *Content* **application/json**	No Links
204	No content available for the given parameters	No Links
500	Internal server error	No Links

GET `/game-result/{gameId}/results/average/year/{patientId}` Get average results for a game by year for a patient

Returns the average results for a game by year for a given patient ID and game ID within a specified date range.

Parameters

Type	Name	Description	Schema
path	**gameId** *required*	The ID of the game.	string
query	**finalDate** *required*	The final date for the search in YYYY-MM-DD format.	string
path	**patientId** *required*	The ID of the patient.	string
query	**sound** *optional*	The sound used in the game.	string
query	**initialDate** *required*	The initial date for the search in YYYY-MM-DD format.	string

Responses

Code	Description	Links
200	Average results for a game by year for a patient	No Links
204	No content found for the specified query.	No Links
500	Internal server error.	No Links

GET `/patient` Get all patients

Responses

Code	Description	Links
200	Returns an array of patients *Content* **application/json**	No Links
204	No patients found	No Links
500	Server error	No Links

POST `/patient` Create a new patient

Responses

Code	Description	Links
201	Returns the created patient *Content* **application/json**	No Links
204	User not found	No Links
500	Server error	No Links

GET `/patient/responsible/{id}` Get all patients assigned to a responsible user

Parameters

Type	Name	Description	Schema
path	**id** *required*	ID of the responsible user	string

Responses

Code	Description	Links
200	Returns an array of patients *Content* **application/json**	No Links
204	No patients found for the responsible user	No Links
500	Server error	No Links

DELETE `/patient/{id}` Deletes a patient by ID.

Deletes a patient by ID.

Parameters

Type	Name	Description	Schema
path	**id** *required*	ID of the patient to delete.	string

Responses

Code	Description	Links
200	The deleted patient. *Content* **application/json**	No Links
204	No patient found for the given ID.	No Links
500	Internal server error. *Content* **application/json**	No Links

POST `/user/login` Login to the application

Responses

Code	Description	Links
200	Returns a JSON Web Token *Content* **application/json** *Properties* **Name** / **Description** / **Schema** token *optional* / JSON Web Token / string	No Links
401		No Links
500		No Links

GET `/user` Retrieves a list of all users.

Responses

Code	Description	Links
200	A list of users. *Content* **application/json**	No Links
204	No users found.	No Links

Type	Name	Scopes
apiKey	bearerAuth	

POST `/user` Create a new user.

Responses

Code	Description	Links
201	User created successfully. *Content* **application/json**	No Links
500	Internal server error. *Content* **application/json**	No Links

GET `/user/{id}` Get a user by ID

Parameters

Type	Name	Description	Schema
path	**id** *required*	User ID	string

Responses

Code	Description	Links
200	Success *Content* **application/json**	No Links
204	No content found for the specified ID	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/user/{id}` Update an existing user by ID

Parameters

Type	Name	Description	Schema
path	**id** *required*	ID of the user to update	string

Responses

Code	Description	Links
200	User updated successfully	No Links
204	No user found for the given ID	No Links
500	Internal server error	No Links

DELETE `/user/{id}` Delete a user by ID

Delete a user by ID

Parameters

Type	Name	Description	Schema
path	**id** *required*	ID of the user to delete	string

Responses

Code	Description	Links
200	User deleted successfully *Content* **application/json** *Properties* **Name** / **Description** / **Schema** _id *optional* — ID of the deleted user — string email *optional* — Email of the deleted user — string password *optional* — Hashed password of the deleted user — string profilePhoto *optional* — URL of the profile photo of the deleted user — string createdAt *optional* — Date of creation of the deleted user — string updatedAt *optional* — Date of last update of the deleted user — string	No Links
204	No content found with the specified ID	No Links
500	Server error	No Links

POST `/user/{id}/generated-code` Insert generated code for a user.

Inserts an array of generated codes for a specific user. The generated codes array will be limited to 10 items, keeping only the last 10 codes added.

Parameters

Type	Name	Description	Schema
path	**id** *required*	The ID of the user to insert the generated codes for.	string

Responses

Code	Description	Links
200	The updated user with the new generated codes array. *Content* **application/json**	No Links
204	The requested user was not found.	No Links
500	An error occurred while processing the request.	No Links