UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO PEDRO DE BARROS BARBOSA

FAVA:
APLICAÇÃO DE SIMULAÇÃO E VALIDAÇÃO DE AUTÔMATOS FINITOS PARA AUXÍLIO DE ENSINO E APRENDIZAGEM.

CAMPINA GRANDE - PB

2023

JOÃO PEDRO DE BARROS BARBOSA

FAVA:
APLICAÇÃO DE SIMULAÇÃO E VALIDAÇÃO DE AUTÔMATOS FINITOS PARA AUXÍLIO DE ENSINO E APRENDIZAGEM.

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Andrey Elísio Monteiro Brito

CAMPINA GRANDE - PB

2023

JOÃO PEDRO DE BARROS BARBOSA

FAVA:
APLICAÇÃO DE SIMULAÇÃO E VALIDAÇÃO DE AUTÔMATOS FINITOS PARA AUXÍLIO DE ENSINO E APRENDIZAGEM.

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Andrey Elísio Monteiro Brito

Orientador – UASC/CEEI/UFCG

Wilkerson de Lucena Andrade

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 28 de junho de 2023.

CAMPINA GRANDE - PB

FAVA:
FINITE AUTOMATA SIMULATION AND VALIDATION APPLICATION FOR LEARNING-TEACHING AID.

ABSTRACT

The courses that contain the automata's theory subject bring with them a necessary mathematical and formal nature. However, when the study of this subject is not planned in conjunction with the practical exercises, it usually becomes a factor that compromises the student's understanding on it. Still, the resolution and correction of such exercises, when done in a confusing manner without proper instruction, can be lengthy, tedious, and prone to errors.

To try solving this struggle and to support teaching and learning, this work proposes an online web platform for visually and interactively constructing finite automata, deterministic or not, its testing, validation, and application of certain algorithms on them. It also includes an environment focused on the resolution of exercises created by teachers for students, providing them with quick and constant feedbacks.

FAVA: Aplicação de simulação e validação de autômatos finitos para auxílio de ensino e aprendizagem.

João Pedro de Barros Barbosa
joaopedro.bb98@gmail.com
joao.pedro.barbosa@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

Andrey Elísio Monteiro Brito
andrey@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

RESUMO

Os cursos que apresentam a teoria dos autômatos trazem consigo uma natureza matemática e formal necessária. Porém, quando o estudo desse tema não é planejado de forma conjunta à realização de exercícios práticos, geralmente se torna fator que compromete a compreensão por parte do aluno. Ainda assim, a resolução e correção de tais exercícios, quando feitas de forma confusa, sem a devida instrução, podem ser longas, tediosas e propensas a erros.

Para mitigar essa dificuldade e apoiar o ensino e aprendizagem, este trabalho propõe uma plataforma *web online* visual e interativa para construção de autômatos finitos, determinísticos ou não, sua testagem, validação e aplicação de determinados algoritmos sob eles; além de contar com um ambiente focado na realização de exercícios criados por professores para os alunos, fornecendo-os feedbacks rápidos e constantes.

PALAVRAS-CHAVE

Autômatos finitos, Simulador, Validador formal, linguagens formais, software de auxílio acadêmico.

REPOSITÓRIOS

Frontend: https://github.com/joaobb/fava

Backend: https://github.com/joaobb/fava-api

Biblioteca: https://github.com/joaobb/fava-logics

1. INTRODUÇÃO

A ciência da computação é cercada de sistemas de alta complexidade, havendo a necessidade de redução desses a modelos computacionais, a fim de facilitar a maneabilidade, o estabelecimento de teorias, e, por fim, a compreensão acerca da natureza desses processos computacionais. Para tal, durante o curso de bacharelado em Ciência da Computação, são estudadas diversas formas de mitigar esta complexidade. Uma delas é o autômato finito: modelo amplamente utilizado para descrever e analisar sistemas que exibem comportamento finito e sequencial, como em atividades de processamento textual, *design* e validação de sistemas de *hardware* e *software*, dentre outras tantas aplicações. Também sendo notável por sua capacidade de apresentar os conceitos-base de sistemas de execução em formato concorrente, por meio dos autômatos finitos não determinísticos, facilitando a compreensão do funcionamento destes [3].

Entretanto, mesmo representando uma simplificação, o ensino e a aprendizagem desses modelos de computadores são morosos, dada sua natureza matemática, formal e, principalmente, abstrata, podendo apresentar um desafio significativo para estudantes e professores no exercício de suas atividades. Uma solução para isso é a realização de exercícios práticos sobre o tema, possibilitando aos alunos o aprendizado da teoria a partir da prática, trazendo algum nível de materialidade para algo antes totalmente abstrato. Porém, quando feita manualmente, o número de elementos que compõe um modelo de autômato finito tende a exigir uma complexa e detalhista validação de assertividade, tornando seu estudo, tal qual seu ensino, longo, tedioso e por fim, propenso a erros. Consequentemente, o nível de ambição pelo entendimento é reduzido, bem como o compromisso dos alunos para com o assunto.

Diante do exposto, este trabalho tem como intuito apresentar uma plataforma criada objetivando a mitigação de tais dificuldades — tanto de ensino, quanto de aprendizagem —, a partir do oferecimento de um ambiente visual e iterativo para construção de autômatos finitos, determinísticos ou não; e para a testagem, validação e aplicação de exercícios práticos criados por docentes, focando em seus discentes diretos ou o público geral. Estes exercícios, sendo avaliados e graduados automaticamente quando respondidos, fornecendo feedbacks rápidos e constantes aos seus respondentes. Além disso, provêm aos professores uma área de gerenciamento e visualização de estatísticas de submissões e avaliações de exercícios. Assim, a plataforma atinge tanto docentes, quanto discentes.

Para elucidação de requisitos, foram realizadas entrevistas com integrantes dos corpos discente e docente da disciplina de Teoria da Computação, ministrada no curso de Ciência da Computação da Universidade Federal de Campina Grande (UFCG). Ambos deram parecer positivo quanto a utilidade da ferramenta, cada um com seus interesses específicos.

Este artigo está organizado em cinco partes. Após esta introdução, a seção dois apresenta os fundamentos teóricos, descrevendo brevemente o modelo computacional dos autômatos finitos, assim como o algoritmo base utilizado para testes de equivalências entre estes autômatos, e, por fim, lista e analisa algumas soluções semelhante à apresentada neste trabalho. Já na seção três, a solução em si é descrita, detalhando suas funcionalidades (textualmente e a partir de recortes visuais do

estado da arte da aplicação) e a arquitetura utilizada por essa. Na quarta seção são expostos os relatos das experiências tidas e das dificuldades encontradas durante a produção deste artigo e da plataforma foco. Por fim, a quinta seção discute possíveis trabalhos futuros a serem realizados a fim de expandir a plataforma.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Autômatos finitos

De acordo com Hopcroft [14], o autômato é um modelo matemático de um sistema, com enfoque em algum detalhe que possua uma quantidade finita de estados. Esses são fragmentos de memória relevantes para a história do sistema, alternados a partir de reações do sistema a entradas externas, para validar ou não uma sequência de ações.

O modelo possui inúmeros casos de utilização, como o de processamento textual, seja de linguagem natural ou de programação — na construção de compiladores, por exemplo —, e o de verificação e validação de corretude de circuitos digitais ou protocolos, desde que possuam um número finito de estados em sua composição.

O modelo se divide em duas principais categorias, determinísticos e não-determinísticos, cada uma com suas características próprias, porém logicamente equivalentes.

2.2 Equivalência de autômatos

Um dos problemas clássicos da teoria dos autômatos é o teste de equivalência entre eles, uma vez que pode provar que dois modelos construídos de formas distintas podem ter o mesmo resultado logicamente. Estes testes desempenham um papel fundamental na verificação e validação de sistemas baseados em autômatos finitos, podendo validar refatorações e otimizações de sistemas, por exemplo, e assim garantir uma maior confiabilidade e corretude nesses sistemas.

Para tal, Hopcroft e Karp [36] demonstraram uma solução com tempo de execução em tempo quase linear, mais eficiente quando comparada aos outros métodos, que chegam a alcançar tempos de execução exponenciais, como o de preenchimento de tabela, por exemplo.

O algoritmo se baseia na divisão dos autômatos em conjuntos de estados relacionados e na validação destes grupos em tempo de execução, determinando se seus integrantes, e consequentemente os autômatos, são equivalentes.

Esta formação de grupos decorre da iteração dos autômatos partindo de seus estados iniciais e da união dos estados resultantes da aplicação das suas funções de transição para cada símbolo do seu alfabeto.

Dois autômatos são tomados como equivalentes quando todos os grupos formados por essa iteração tem todos os seus elementos como de aceitação OU todos de não aceitação. Caso haja algum cruzamento destes dois tipos de estados no mesmo grupo, os autômatos serão não equivalentes.

Porém, esse algoritmo, em seu formato original, limita-se apenas a autômatos finitos determinísticos. Mesmo que seja possível a sua aplicação em autômatos não determinísticos, por meio da conversão entre os modelos, tornar-se-ia inviável, quando comparado às outras soluções disponíveis, devido ao acréscimo ao seu tempo de execução que esta nova operação ocasionaria.

2.3 Trabalhos relacionados

Uma das soluções encontradas para tentar sanar tal dificuldade no ensino e aprendizado foi a de simular esses autômatos. Então, nos últimos 60 anos foram desenvolvidos inúmeros simuladores focados em autômatos e outras linguagens formais [7], apresentando variados propósitos, funcionalidades e capacidades, acompanhando o estado da arte do poder computacional.

Portanto, é evidente que a utilização de simuladores de autômatos é uma ferramenta fundamental para um ensino e aprendizagem destes sistemas formais de forma mais tangível.

Abaixo, estão apresentadas algumas das principais soluções similares à aqui desenvolvida, assim como seus pontos específicos.

- *JFlap* [18]

Software que permite a experimentação de diversos tipos de modelos de linguagens formais a partir de criação, simulação e testagem visuais. Abrange autômatos finitos determinísticos (AFD) e não-determinísticos (AFN), gramáticas livres de contexto, Máquinas de Turing, entre outros. Vem sendo desenvolvido continuamente desde 1993 pelos membros da *University of Duke* para possuir a maior gama de funcionalidades sobre o tema. Assim, justifica-se o fato de este ser o simulador mais amplamente divulgado e utilizado [7][13] dentre os disponíveis. Entretanto, por se tratar de um *software* Java, requer o *download* e execução desse e do *runtime* Java (JRE) em um computador para se tornar funcional, dificultando a utilização para usuários.

- *OpenFlap* [27]

Aplicação *web*, de acesso livre, baseada no JFlap, voltada para avaliação automatizada de exercícios e integração com ambientes de aprendizado (LMS) e livros didáticos interativos a partir da infraestrutura fornecida pelo *OpenDSA* [26]. Suporta construção, simulação, testagem e avaliação de autômatos finitos determinísticos, não-determinísticos, autômatos de pilha e Máquinas de Turing. Essa solução é bastante completa, porém é de difícil acesso e integração com métodos de ensino que não sigam o modelo de livros didáticos interativos ou utilizem o *OpenDSA*.

- *Automaton Simulator* [31]

Aplicação *web* que permite a construção e simulação de autômatos determinísticos, não-determinísticos, gramáticas livres de contexto e máquinas de Turing. Desenvolvido na Universidade da Califórnia em Davis (UCDAVIS), não é tão amigável ao usuário quanto as alternativas anteriores, já que não conta com visualização gráfica do autômato, apenas a sua representação em

texto. Assim, requer do usuário uma curva de aprendizado para o entendimento da estrutura a ser utilizada na ferramenta.

2.3.1 Limitações e conclusão

As alternativas citadas acima possuem as funcionalidades básicas semelhantes com as do projeto proposto neste documento: a criação visual de um autômato e sua testagem a partir de um conjunto de palavras. Porém, falta-lhes o componente de auxílio acadêmico ao discente e docente e a sua fácil integração ao método de ensino, principal diferencial da aplicação desenvolvida.

3. SOLUÇÃO

A FAVA (*Finite Automata Validation Application*), ou Aplicação de Validação de Autômatos Finitos, é uma plataforma *web* disponibilizada livremente para auxiliar no ensino-aprendizagem acerca do modelo computacional autômato finito (acessível em https://fava.vercel.app). É focada em prover um ambiente interativo de construção de autômatos integrado a um repositório de exercícios práticos populado pelos próprios usuários da plataforma.

A solução foi pensada e concedida para suprir as principais características apresentadas por [21] para a criação de uma ferramenta e visualização de linguagens formais com real impacto em seus usuários, como a facilidade de interação, permitindo a criação e edição de autômatos organicamente em um ambiente *sandbox* a partir de *clicks* de *mouse*. Outra característica é a visualização animada do funcionamento do autômato, exibindo todo o caminho percorrido nesse para a simulação do processamento de palavras definidas pelo usuário, passo a passo, sendo controlado pelo usuário, a fim de atender estudantes com tempos de aprendizagem distintos, facilitando o entendimento do modelo construído.

Já para os professores, a plataforma auxilia principalmente na organização e acompanhamento de seus alunos por meio da criação de turmas para agrupamento e gerenciamento. Além disso, por realizar todas as correções das atividades práticas propostas de maneira automática, reduz a zero o tempo antes empregado nesta atividade.

Esta correção é feita baseada no teste de equivalência entre os autômatos-base da questão (construído pelo professor no momento da criação do exercício) e o submetido pelo aluno ao tentar resolvê-la. Para a realização deste teste, é utilizado o algoritmo *Hopcroft–Karp extended* (HKe) [1], uma extensão do algoritmo mencionado na seção dois que capacita a sua aplicação a testes entre todas as classes de autômatos finitos, determinísticos ou não, sem grandes alterações à boa eficiência do algoritmo original. Para prevenir fraudes, toda essa lógica de avaliação é realizada no *backend*, camada do sistema não acessível diretamente pelo usuário e protegida por uma combinação de e-mail e senha.

A plataforma foi desenvolvida com grande foco na expansibilidade, seguindo padrões, estruturas e tecnologias estabelecidas pela comunidade, como a linguagem *JavaScript* [15] e seu superconjunto *Typescript* [16], para a manutenção de um padrão mais sólido de contrato de interfaces entre os componentes da plataforma; o uso das as tecnologias *ReactJS* [30], *ExpressJS* [9] e *PostgreSQL*; além da disponibilização de todo o código-fonte via *GitHub* [12]. Facilita-se, assim, o desenvolvimento de novos módulos e operações por parte de usuários interessados, abordagem que permite a continuidade do aprimoramento da plataforma e o maior englobamento de temas acerca dos autômatos finitos, causando mais impacto no ensino-aprendizagem.

3.1 Funcionalidades

A plataforma apresenta uma variedade de funcionalidades que visa proporcionar uma experiência mais lúdica e interativa para o aprendizado do aluno. Estas funções foram requisitadas durante entrevistas com seis alunos e com dois professores interessados na solução.

3.1.1 Criação de autômato visual e iterativamente:

A partir do ambiente *sandbox* fornecido pela plataforma, qualquer usuário, independentemente de estar cadastrado no sistema ou não, tem a capacidade de criar, de forma visual e interativa, um autômato finito, adicionando estados e suas transições conectivas por meio de *clicks* na área de criação. É possível, também, editar ou remover estes estados e suas transições, marcando-os como iniciais ou não, e como sendo de aceitação ou não, como visto na figura 1, além de conseguir definir os símbolos destas transições.

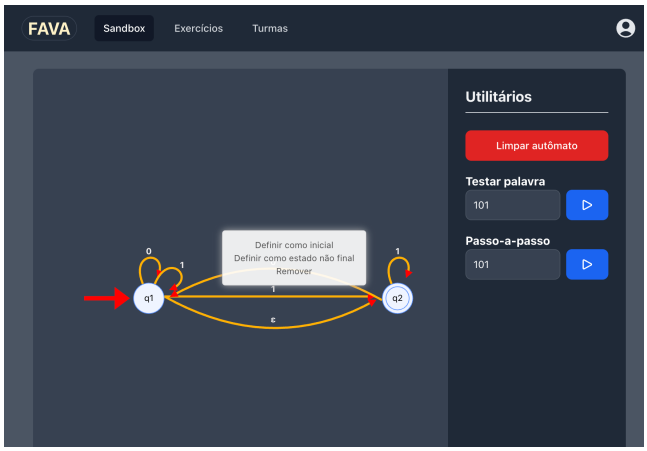

Figura 1 - Edição de estado na criação de autômato no ambiente *sandbox*.

3.1.2 Teste de aceitação de palavra por autômato:

Após a criação do autômato, o sistema possibilita ao usuário a realização de testes de aceitação de palavras por ele. Isso ocorre a partir da inserção da palavra a ser testada e resulta num estado de aceitação ou rejeição dessa, apresentado visualmente como na figura 2. Isso permite verificar a conformidade do autômato com um conjunto de palavras da linguagem desejada e vice-versa.

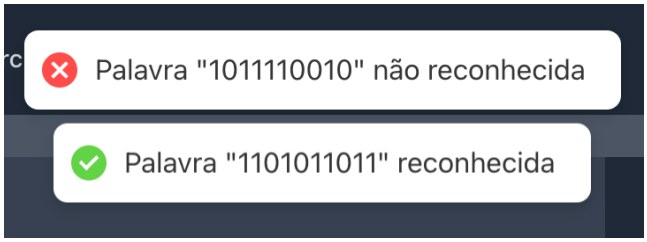

Figura 2 - Resultados possíveis do teste de aceitação de palava por autômato.

3.1.3 Teste passo-a-passo de aceitação de palavra por autômato:

Além do teste direto de aceitação, o sistema fornece a opção de acompanhar todo processo de aceitação ou rejeição de uma palavra passo-a-passo, percorrendo todas as transições e estados decorrente do processamento da palavra pelo autômato, sempre identificando visualmente os estados atuais e transições utilizadas. Isso permite uma compreensão mais detalhada do funcionamento do autômato criado em diferentes cenários.

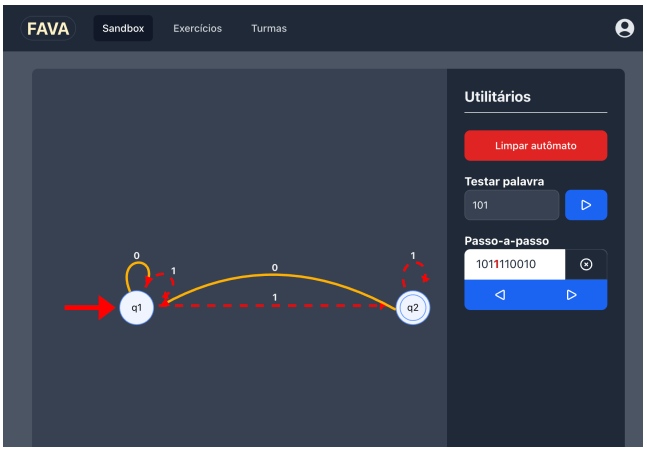

Figura 3 - Teste passo-a-passo (*debug*) de aceitação de palavra por autômato.

3.1.4 Cadastro de usuário:

Para que haja o armazenamento dos exercícios criados e resolvidos e a identificação individual de utilitários do sistema, é oferecido o cadastro de usuários, realizado a partir do fornecimento de informações básicas, como nome, *e-mail* e senha.

O usuário define, no momento de sua criação, o papel que performará no sistema, optando entre aluno ou professor, escolha que determinará qual categoria de funcionalidades a que terá acesso.

3.1.5 Criação de exercício:

Os usuários cadastrados como professores são capazes de criar exercícios por meio da construção de um autômato na área de *sandbox* e a inserção de sua descrição textual, explicando brevemente o funcionamento do modelo construído. Essa descrição fornece aos estudantes orientações e contexto sobre o exercício proposto, auxiliando-os na compreensão e, consequentemente, sua resolução.

Os exercícios podem ser disponibilizados de duas formas configuráveis durante a criação: público, onde qualquer usuário cadastrado na plataforma terá acesso e poderá tentar sua resolução; ou privado para uma turma, de forma a ser disponibilizando apenas para os usuários inscritos na turma selecionada pelo professor.

Figura 4 - Criação de exercício com declaração visual e descrição textual de autômato.

3.1.6 Filtragem de exercícios:

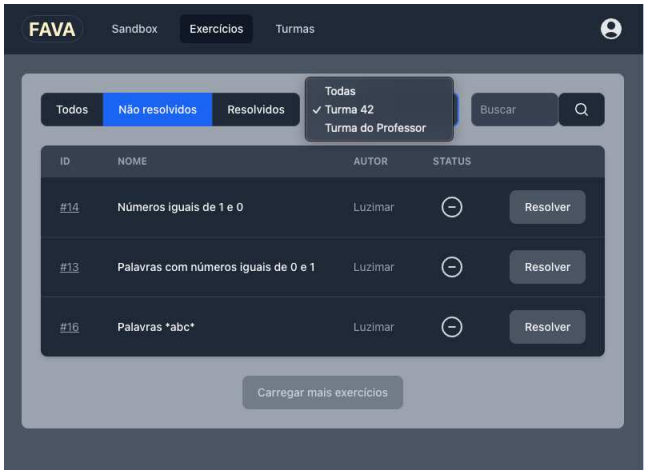

Figura 5 - Listagem de exercícios filtrando apenas os da *Turma 42* e que não foram resolvidos.

Para facilitar a organização e o acesso dos exercícios disponíveis ao usuário, é possibilitada a filtragem a partir dos seguintes critérios: por estado de resolução, agrupando os exercícios em dois grupos, aqueles que já foram e os outros que ainda não foram resolvidos; por turma, permitindo, a partir da seleção da turma desejada, a exibição apenas dos exercícios disponibilizados apenas para essa; por nome.

3.1.7 Resolução de exercícios:

Os usuários conseguem resolver exercícios para eles disponíveis de acordo com seu papel e inscrições em turmas. Para tal, será apresentado o título e a descrição textual do autômato adicionada pelo criador do exercício, contextualizando-o. O respondente deverá criar um autômato finito que siga as instruções dadas pela questão, podendo realizar testes de aceitação de palavras sob o autômato neste processo.

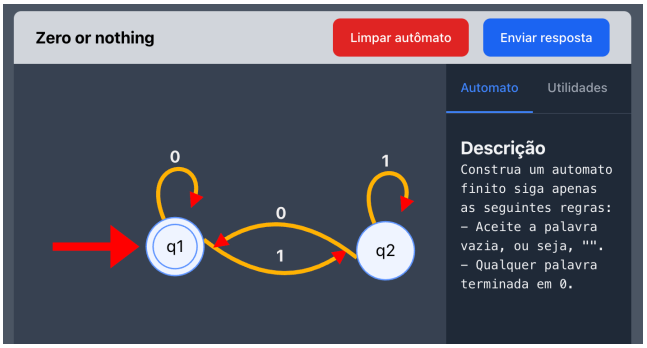

Figura 6 - Ambiente *sandbox* de resolução de exercício por um aluno.

3.1.8 Criação de turma:

Os usuários cadastrados na plataforma na categoria de professor são capazes de criar turmas, a partir da inserção de um título e uma breve descrição. Esta criação resulta em um código *hash* único de 10 caracteres de comprimento para identificação da turma, com função de convite à turma, devendo ser compartilhada com os alunos que desejem se inscreverem nessa.

Com o uso de turmas, é proporcionado um ambiente de aprendizagem mais organizado e controlado, com um acompanhamento mais fino de seus alunos.

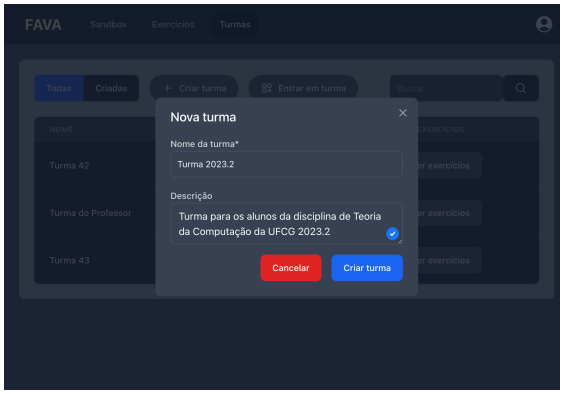

Figura 7 - Modal de criação de turma por professor.

3.1.9 Inscrição em turma:

Os usuários cadastrados no sistema podem se inscrever em turmas criadas anteriormente, disponibilizando o acesso a exercícios privados atribuídos pelo professor a essa e permitindo um acompanhamento mais direto com o professor.

Essa inscrição é feita mediante a utilização do *hash* de identificação da turma no sistema repassada pelo criador dessa.

Se a inscrição foi realizada com sucesso, o sistema apresentará uma mensagem de boas-vindas ao usuário.

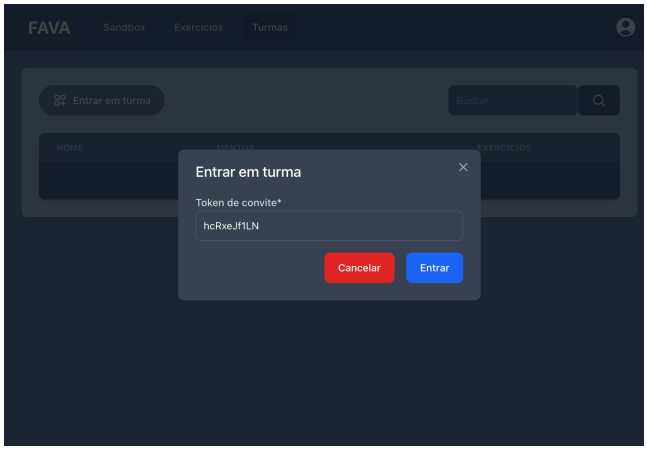

Figura 8 - Modal de inscrição de usuário em turma

3.1.10 Gerenciamento e visualização de turma:

Aos professores tutores de turmas, é fornecida pelo sistema uma visualização em lista dos exercícios privados a suas turmas pertencentes, tal como as submissões de exercícios, realizadas por seus integrantes. Nesta listagem são apresentadas métricas individuais e do grupo geral da quantidade de submissões e da taxa de acerto dos questionários enviados. Assim, permite-se um acompanhamento mais detalhado do progresso dos alunos e *feedbacks* individuais mais contextualizados.

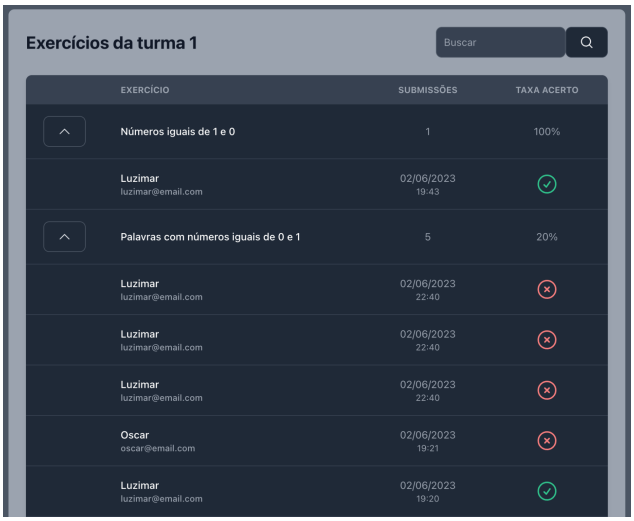

Figura 9 - Visualização de exercícios privados a turma específica, suas submissões e respectivas estatísticas.

3.2 Arquitetura

A solução foi projetada e realizada baseando-se no modelo arquitetural MVC (*Model-View-Controler*) [23] como apresentado na figura 10, segregando a aplicação em três frentes principais: cliente (*frontend/view*), lógica de negócios (*backend/controller*) e banco de dados (*model*). Cada uma destas camadas com suas competências e funcionalidades específicas, e contando com formas de comunicação com sua adjacência seguinte, buscando simplicidade e facilidade na manutenção e evolução do código e da plataforma.

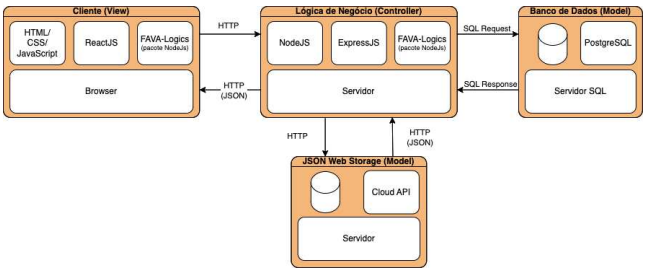

Figura 10 - Diagrama de arquitetura do sistema

3.2.1 Frontend

Parte visual da aplicação, por meio da qual o usuário terá contato direto com as funcionalidades oferecidas pelo sistema, como a criação interativa de autômatos, a resolução de exercícios e o gerenciamento de turmas, por exemplo. Foi modelada e desenvolvida utilizando o *ReactJS* [30], biblioteca focada na construção de interfaces de usuário interativas por meio da união de componentes.

O *ReactJS* foi escolhido pela sua capacidade de ser de rápido, sucinto, mas ainda de robusta construção, e, principalmente, por sua popularidade e adoção [32] em seu meio, possibilitando constante evolução e removendo barreiras para interessados na expansão do sistema.

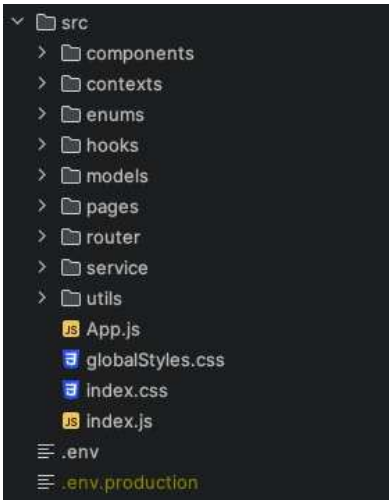

Figura 11 - Estrutura de diretórios do cliente

Em conjunto ao *ReactJS*, foram utilizadas bibliotecas para resolução de requisitos específicos:

- *AntV G6* [2]: Ferramenta de código aberto de construção e visualização interativa de grafos.
- *Flowbite* [11] e *Tailwind* [33]: Bibliotecas, também de código aberto, de componentes visuais que agilizam a construção de interfaces, permitindo que o desenvolvimento seja focado na solução do problema principal, e não nos da tecnologia utilizada.

Para a comunicação com a camada seguinte, são realizadas requisições com protocolo HTTP, enviando e recebendo dados formatados no padrão JSON.

3.2.2 Backend

Camada do sistema que contém suas principais regras de negócio e as utiliza para gerenciá-lo, além de agir como uma ponte entre as camadas vizinhas.

Foi desenvolvida utilizando o *Node.JS* [24] — ambiente de execução da linguagem *JavaScript* independente de um navegador *web* — e o framework *ExpressJS* [9] — ferramenta construída sobre *NodeJS* que fornece os recursos necessários para um servidor *web*.

É responsável por receber, tratar, realizar e responder a requisições vindas da camada visual da aplicação, como as de validação de submissão de questões de alunos, gerência de usuários e permissões de ações realizadas por esses dentro do sistema, por exemplo.

Tem o fluxo de funcionamento dividido nas seguintes: *Router*, *Controller*, *Middleware* e *Service*, conforme a figura 12. Inicialmente, ao receber uma requisição o *router*, determina a partir do *endpoint* chamado a qual *controller* repassar os dados dessa e se devem ser aplicados *middlewares,* funções intermediárias às camadas, como identificação de paginação, autenticação e permissionamento de rotas. O *controller*, por sua vez, tem a função de obter os dados recebidos no corpo da *request* e tratá-los a fim de seguir a interface do *service*. Esse é responsável pela execução das regras de negócio do sistema, tendo acesso ao seu banco de dados, por meio dos repositórios das entidades criadas.

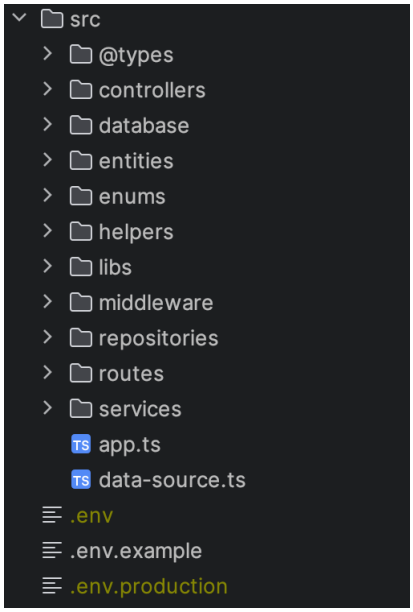

Figura 12 - Estrutura de diretórios do servidor

Para autenticação de usuários, foi utilizado o padrão *JSON Web Token* (JWT) [5], transacionado entre cliente e servidor via cabeçalho *Authorization* das requisições HTTP, com informações básicas do usuário, como nome, *e-mail* e cargo no sistema, indicando qual o usuário e seu estado na aplicação.

A biblioteca *TypeORM* [34] foi utilizada na comunicação desta camada com o banco de dados. Ela fornece um mapeamento do padrão estrutural do banco de dados para um orientado a objetos, utilizado nesta camada da aplicação. Assim,

não é necessária a escritas de consultas SQL diretamente, aumentando a produtividade de desenvolvimento e reduzindo complexidade.

3.2.3 *Banco de dados*

Componente responsável pela persistência dos dados aplicação.

Devido à necessidade de armazenar dados de formatos distintos, foram utilizadas duas soluções de para tal na aplicação:

- SQL: Banco de dados estruturado objeto-relacional SQL, visando armazenamento de estruturas bem definidas geradas e consumidas pelo sistema, como mostrados na figura 13, como questionários, informações de usuários e turmas. Isso se dá por meio do sistema de gerenciamento (*SGBD*) *PostgreSQL* [28], escolhido por sua robustez, escalabilidade e, assim como as soluções das camadas anteriores, larga adoção no mercado [32].
- JSON *Web Storage*: [19] Serviço *web* para armazenamento de dados em formato JSON. Formato utilizado para a representação textual dos autômatos criados visualmente pelos usuários na aplicação. Foi utilizado para evitar o sobreuso da solução SQL, podendo ocasionar em instabilidades e maiores custos de execução do sistema.

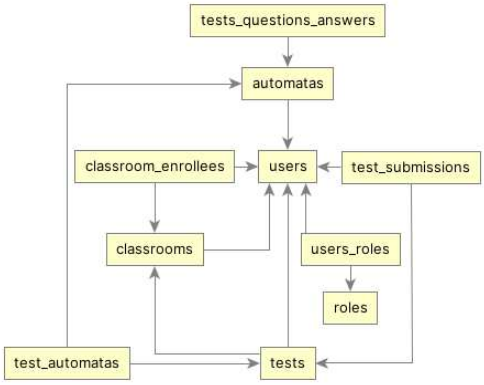

Figura 13 - Diagrama de Entidade-Relacionamento do bando de dados PostegreSQL do sistema

3.2.4 *FAVA*

Além dessas camadas, foi produzido um pacote de código independente para realização de operações sobre autômatos finitos, como teste de aceitação de palavras por um autômato, testes de equivalência entre dois autômatos finitos, por exemplo.

Foi elaborado a fim de unificar toda a lógica referente aos autômatos utilizada pelo sistema, separando, assim, essa e a regra de negócio geral da plataforma, além de evitar a repetição de código e conexões desnecessárias entre as camadas; e, principalmente, para disponibilizar uma biblioteca aberta à comunidade interessada em construir soluções semelhantes e evoluir em conjunto.

Por fim, foi instalado e utilizado em ambos *frontend* e *backend* da aplicação, como ilustrado na figura 10.

3.2.5 *Implantação*

A implantação de todos os componentes do sistema foi realizada a fim de disponibilizar a solução livremente aos interessados. Foram utilizadas plataformas de hospedagem distintas para os diferentes componentes do sistema. As escolhas foram feitas focando nas capacidades dos serviços, na escalabilidade e no nível de integração com o *GitHub*.

O *frontend* foi publicado através do serviço *Vercel* [35], plataforma de hospedagem e implantação de aplicações *web*, com foco voltado para *apps frontend*.

Já o *backend* e o banco de dados *PostgreSQL* foram implantados utilizando os serviços da plataforma *Render* [8], mais focada em serviços internos, de API, banco de dados e outros.

Essas decisões permitiram a disponibilização do sistema de forma automatizada e eficiente, garantindo escalabilidade e maior período de disponibilidade de acesso, no entanto, sem ônus financeiros para tal, já que foram utilizados apenas os planos gratuitos oferecidos pelas plataformas.

4. EXPERIÊNCIA E APRENDIZADO

4.1 Processo de desenvolvimento

Para obtenção e elucidação de requisitos necessários para a solução foram realizadas múltiplas entrevistas, com diversos integrantes dos corpos docente e discente do curso de bacharelado em Ciência da Computação da Universidade Federal de Campina Grande. Dente esses, cerca de 10 alunos que já haviam cursado a disciplina de Teoria da Computação — principal público-alvo da aplicação—, e dois professores que ministram a disciplina na instituição. Estas entrevistas seguiram ambos os modelos semi-estruturado e não estruturado, buscando maior naturalidade na obtenção das necessidades dos usuários para com a plataforma desenvolvida. Em todas as entrevistas foram dados pareceres positivos quanto à utilidade da ferramenta, e cada grupo apresentou funcionalidades que imaginava importantes para a plataforma. Com isso foi gerado um *backlog* e um cronograma de desenvolvimento com períodos quinzenais seguindo o modelo ágil *Scrum* [4].

Logo, foi realizada a busca e revisão bibliográfica sobre o tema de autômatos finitos, para aprofundamento e aprimoramento dos requisitos, possibilitando assim uma melhor esquematização do sistema e definição das tecnologias que seriam utilizadas para seu desenvolvimento. Em conjunto, foi feita uma pesquisa de *design* nas soluções relacionadas e nas soluções de construção de gráficos de fluxo via plataforma de compartilhamento de portifólios *Behance* [6], de onde foram obtidos diversos conceitos e padrões utilizados e seguidos na solução.

Com as tecnologias principais definidas, três repositórios foram criados no *GitHub*, um para cada parte da solução, configurados de acordo com suas necessidades. Em todos foram instaladas e utilizadas bibliotecas para a manutenção de padrões e da qualidade do código, como *Eslint* [10] — analisador estático de código — e *Prettier* [29] — formatador de estilo código. Essas permitiram, a partir da definição de regras de estilo

e normatividade, garantir legibilidade e consistência entre as aplicações. Com isso, deu-se início ao desenvolvimento.

Durante o desenvolvimento, foram realizadas recorrentemente consultas a integrantes dos grupos interessados, para verificação e validação da evolução da aplicação. Possibilitou-se, assim, a aplicação de ajustes ao escopo conforme as necessidades e possibilidades de suas realizações, e o desenvolvimento incremental da aplicação, evitando retrabalhos desnecessários e atendo um maior foco ao escopo prioritário para os usuários.

No final do projeto, foram realizados testes de ponta a ponta da aplicação de maneira manual, em conjunto com um estudante que já havia cursado a disciplina de Teoria da Computação. Neles, foram testadas todas as funcionalidades do sistema, independentemente do tipo de usuário (administrador, professor, estudante), simulando a maior quantidade de ambientes possível para o uso da plataforma. Foram encontrados 3 *bugs* nas funcionalidades de: *debug* de autômato; submissão de exercício; e criação de exercício. Por fim, foram solucionados em uma *sprint* reduzida de uma semana de correção de *bugs*, e posteriormente validados com o usuário em outra sessão de testes.

4.2 Principais Desafios

Com o decorrer do processo de desenvolvimento foram encontrados inúmeros desafios. A seguir serão apresentados dois desses.

A determinação de um algoritmo eficiente para o teste de equivalência de autômatos foi o maior deles. Este teste é recorrentemente tratado em artigos científicos, porém muitos se tornam bastante complexos quando se aprofundam nos ramos da teoria da computação e lógica matemática, tornando-os inviáveis para o desenvolvimento em uma aplicação relativamente simples como essa. Tal desafio foi parcialmente resolvido com mais pesquisas bibliográficas, resultando na identificação do algoritmo de *Hopcroft–Karp* [36]. Porém, como citado na introdução deste artigo, em sua versão original o algoritmo possui apenas suporte a autômatos finitos determinísticos, limitando assim sua utilização direta, e requerendo algoritmos mais custosos, como o de conversão da classe dos não-determinísticos para determinístico para seu uso em casos mais genéricos. Até que foi percebido noutro artigo [1], que apresentava um agrupado de algoritmos com este objetivo, uma variação da solução original do *Hopcroft–Karp*, porém mais performático e com suporte a autômatos não-determinísticos. Supriu-se, assim, o escopo necessário, de modo que foi escolhido para ser utilizado na aplicação.

Outro desafio foi a integração da aplicação com o *Moodle* [22] — plataforma de gerenciamento de cursos utilizada massivamente no meio acadêmico —, de forma a permitir a aplicação de exercícios sobre a construção de autômatos finitos diretamente na plataforma. Ela foi proposta como um dos principais requisitos para o projeto pelo corpo docente entrevistado, já que traria integração direta com o ambiente já utilizado por eles em seus programas pedagógicos. Porém, mesmo após vasta pesquisa por tutorias, repositórios de código público e artigos, não foi encontrado material suficiente para o desenvolvimento dessa integração, uma vez que em sua maioria se tratava de informações desencontradas ou até faltosas. Com este insucesso e para evitar atrasos no cronograma, tal funcionalidade foi retirada do escopo da aplicação, com o entendimento e consentimento dos quais a requisitaram.

5. TRABALHOS FUTUROS

O principal objetivo da versão da plataforma apresentada neste artigo é a prova de conceito de uma solução que poderia integrar alunos e professores, a fim de facilitar seus ofícios para com o tema de autômatos finitos. Há amplo espaço para expansão de domínio e funcionalidades.

Primeiramente, para sua evolução, é necessária a implementação de suítes de testes unitários automatizados que abranjam todos os componentes do sistema, assim como suas integrações. Isso trará mais consistência e confiabilidade ao sistema, pontos fundamentais para uma solução que pode ter grande impacto na compreensão de um tema tão fundamental para a ciência da computação. A implementação destes testes pode ser feita a partir de ferramentas como *Jest* [17], um *framework* de produção de testes unitário, e *Supertest* [20], framework de testes de integração entre componentes.

Após a implementação de tais testes, a fim de garantir mais confiabilidade a aplicação, essa será submetida a uma avaliação em sua completude, em conjunto a um professor e um estudante da disciplina de teoria da computação — essa que possui a teoria dos autômatos em sua ementa —. Essa abordagem permite identificar eventuais falhas e *bugs* para poderem ser corrigidos. Uma vez finalizadas as correções necessárias, agora com o sistema mais robusto e confiável, a fim de validar seu uso, esse deverá ser integrado na metodologia de ensino de uma turma teste real da disciplina de teoria da computação, acompanhando o seu uso pelos alunos, e testando seu sucesso no auxílio ao aprendizado para estes.

Outra pretensão é a expansão do arcabouço de operações realizadas sobre autômatos pelo sistema, adicionando funções como conversão de autômatos não-determinístico para determinístico, concatenação e união de autômatos, aplicação de Fecho de Kleene [14], dentre outras operações. Assim, permite-se aos alunos uma exploração visual e de forma interativa de mais conceitos e algoritmos complexos, aprofundando assim o entendimento deles do tema.

Visando uma maior interação entre aluno e professor na plataforma, um ponto seria a sua integração ao sistema *Moodle*. Essa funcionalidade, como mencionado previamente, foi requisitada por ambos os professores entrevistados, porém por insucesso em sua implementação foi necessária sua remoção do escopo da solução para o estado de prova de conceito.

Ainda sobre integrações, uma com o modelo de *ChatGPT* [25] seria de grande valor, uma vez que esse poderia gerar, a partir de inteligência artificial, questões e questionamentos sobre autômatos de forma automática. Com essa colaboração entre a plataforma e o *GPT*, os estudantes poderiam

tirar dúvidas, praticar e aprimorar suas habilidades de forma mais interativa, envolvente e orgânica.

6. AGRADECIMENTOS

Gostaria de agradecer a todos que estiveram presentes por todo o período que levou a esse momento, a conclusão de mais um ciclo acadêmico.

Em primeiro lugar, agradeço aos meus pais, Jailma e Luizmar, e ao meu irmão, Vitor, que durante minha vida vem me guiando, ensinando-me mais sobre essa, apoiando-me em todas as minhas escolhas e me motivando constantemente, passo a passo, nessa eterna caminhada. Obrigado por todos os ensinamentos, incentivos e por prezarem pela educação, humana, cultural e crítica.

Aos meus amigos, Murilo Holanda, Lucas Aquino, Alexandre Barbosa, Gabriel Dantas, Eduardo Henrique, Gustavo Bispo, Luan Carlos, Gabriel Almeida, Matheus Araújo, Roberto Honório, Maiana Brito, Cleyton Bruno, e principalmente a Rafael Dantas e Edson Weslley que me ajudaram a persistir no curso, com suas palavras e seu suporte durante os períodos mais árduos. E a todos que caminharam ao meu lado, me deram suporte e me suportaram.

Ao professor Kyller Gorgonio, que primeiro me orientou e compartilhou a ideia que se findou no desenvolvimento deste trabalho de conclusão, despertando minha curiosidade e motivação para com o tema. E ao professor Andrey Elísio, por ter me guiado durante o processo de maturação e desenvolvimento da ideia, oferecendo valiosos *insights* importantes para a melhoria da solução. Obrigado por compartilharem seus conhecimentos, e principalmente, por terem sido humanos durante o processo de produção deste trabalho.

7. REFERÊNCIAS

[1] Almeida, M., Moreira, N., & Reis, R. (2009). Testing the equivalence of regular languages. Electronic Proceedings in Theoretical Computer Science, 3, 47–57. https://doi.org/10.4204/eptcs.3.4

[2] AntV G6. (n.d.). AntV G6. Retrieved June 9, 2023, from https://g6.antv.antgroup.com/en/

[3] Armoni, M., Rodger, S., Vardi, M., & Verma, R. (2006). automata theory. ACM SIGCSE Bulletin, 38(1), 197–198. https://doi.org/10.1145/1124706.1121403

[4] Atlassian. (n.d.). Scrum: O que é, como funciona e como começar. Atlassian. Retrieved June 9, 2023, from https://www.atlassian.com/br/agile/scrum

[5] auth0.com. (n.d.). JSON web tokens introduction. Auth0. Retrieved June 9, 2023, from https://jwt.io/introduction

[6] Behance. (n.d.). Retrieved June 9, 2023, from https://www.behance.net/

[7] Chakraborty, P., Saxena, P. C., & Katti, C. P. (2011). Fifty years of automata simulation. ACM Inroads, 2(4), 59–70. https://doi.org/10.1145/2038876.2038893

[8] Cloud application hosting for developers. (n.d.). Cloud Application Hosting for Developers | Render. Retrieved June 9, 2023, from https://render.com/

[9] ExpressJS. (n.d.). Node.Js Web Application Framework. Retrieved June 9, 2023, from https://expressjs.com/

[10] Find and fix problems in your JavaScript code - ESLint. (n.d.). Pluggable JavaScript Linter. Retrieved June 9, 2023, from https://eslint.org/

[11] Flowbite. (n.d.). Flowbite. Retrieved June 9, 2023, from https://flowbite.com/

[12] GitHub: Let's build from here. (n.d.). GitHub. Retrieved June 9, 2023, from http://github.com

[13] Hamada, M. (2013). Turing Machine and Automata Simulators. Procedia Computer Science, 26(18), iii. https://doi.org/10.1016/s1877-0509(13)01289-1

[14] Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). Introduction to automata theory, languages, and computation. Pearson.

[15] JavaScript. (n.d.). MDN. Retrieved June 9, 2023, from https://developer.mozilla.org/pt-BR/docs/Web/JavaScript

[16] JavaScript with syntax for types. (n.d.). Retrieved June 9, 2023, from https://www.typescriptlang.org/

[17] Jest. (n.d.). Retrieved June 9, 2023, from https://jestjs.io/pt-BR

[18] JFLAP. (n.d.). Retrieved June 9, 2023, from https://www.jflap.org/

[19] JSON web storage. (n.d.). Retrieved June 9, 2023, from https://extendsclass.com/json-storage.html

[20] ladjs. (n.d.). Supertest: Super-agent driven library for testing node.js HTTP servers using a fluent API. GitHub. Retrieved June 9, 2023, from https://github.com/ladjs/supertest

[21] McDonald, J. (2002, February 27). Interactive pushdown automata animation. Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education. http://dx.doi.org/10.1145/563340.563489

[22] Moodle - Open-source learning platform. (n.d.). Moodle.Org. Retrieved June 9, 2023, from https://moodle.org/

[23] MVC - MDN Web Docs Glossary: Definitions of Web-related terms. (n.d.). MDN. Retrieved June 9, 2023, from https://developer.mozilla.org/en-US/docs/Glossary/MVC

[24] Node.js. (n.d.). Node.Js. Retrieved June 9, 2023, from https://nodejs.org

[25] O que é GPT AI? - Explicação sobre os transformadores generativos pré-treinados - AWS. (n.d.). Amazon Web Services, Inc. Retrieved June 9, 2023, from https://aws.amazon.com/pt/what-is/gpt/

[26] OpenDSA. (n.d.). Retrieved June 9, 2023, from https://opendsa-server.cs.vt.edu/

[27] OpenFLAP. (n.d.). Retrieved June 9, 2023, from http://lti.cs.vt.edu/LTI_ruby/AV/OpenFLAP/OpenFLAP.html

[28] PostgreSQL Global Development Group. (2023, June 9). PostgreSQL. PostgreSQL. https://www.postgresql.org/

[29] Prettier · opinionated code formatter. (n.d.). Retrieved June 9, 2023, from https://prettier.io/

[30] React. (n.d.). Retrieved June 9, 2023, from https://react.dev/

[31] simulator. (n.d.). Retrieved June 9, 2023, from https://web.cs.ucdavis.edu/~doty/automata/

[32] Stack overflow developer survey 2022. (n.d.). Stack Overflow. Retrieved June 9, 2023, from https://survey.stackoverflow.co/2022/

[33] Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. (n.d.). Tailwind CSS. Retrieved June 9, 2023, from https://tailwindcss.com

[34] TypeORM. (n.d.). Amazing ORM for TypeScript and JavaScript (ES7, ES6, ES5). Supports MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, WebSQL Databases. Works in NodeJS, Browser, Ionic, Cordova and Electron Platforms. Retrieved June 9, 2023, from https://typeorm.io/

[35] Vercel: Develop. Preview. Ship. For the best frontend teams. (n.d.). Retrieved June 9, 2023, from https://vercel.com

[36] Hopcroft, J., & Karp, R. (1971). A linear algorithm for testing equivalence of finite automata. In eCommons. Cornell University. https://ecommons.cornell.edu/handle/1813/5958