UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

HYTHAN MATHEUS CORREIA DE OLIVEIRA

DESENVOLVIMENTO DE UM PROTÓTIPO DE SISTEMA COM ARQUITETURA DE MICROSSERVIÇOS PARA EMPRESA MAXIAMBIENTAL TREINAMENTOS

MONOGRAFIA DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO

GUARAPUAVA
2022

HYTHAN MATHEUS CORREIA DE OLIVEIRA

DESENVOLVIMENTO DE UM PROTÓTIPO DE SISTEMA COM ARQUITETURA DE MICROSSERVIÇOS PARA EMPRESA MAXIAMBIENTAL TREINAMENTOS

Monografia de Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Sistemas para Internet – TSI – da Universidade Tecnológica Federal do Paraná – UTFPR – Campus Guarapuava, como requisito parcial para obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Dr. Diego Marczal
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

GUARAPUAVA
2022

HYTHAN MATHEUS CORREIA DE OLIVEIRA

DESENVOLVIMENTO DE UM PROTÓTIPO DE SISTEMA COM ARQUITETURA DE MICROSSERVIÇOS PARA EMPRESA MAXIAMBIENTAL TREINAMENTOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnológo em Sistemas para Internet do Curso de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná (UTFPR).

Data da aprovação: 14/dezembro/2022

Prof. Diego Marczal
Doutor
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

---

Prof. Renata Luiza Stange
Mestre
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

---

Prof. Roni Fabio Banaszewski
Doutor
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

GUARAPUAVA
2022

AGRADECIMENTOS

Agradeço meu orientador Diego Marczal por todos os ensinamentos, tanto em sala de aula quanto na minha orientação, e por todo o incentivo de não desistir do curso, além das muitas horas de conversa me passando dicas e me falando de vagas de emprego, para que eu pudesse estar evoluindo profissionalmente. Muitas vezes o trabalho de mestre não é fácil, mas quando há dedicação pela profissão os ensinamentos tornam-se mais acessíveis e sou grato por isso.

Agradeço a minha família e amigos por sempre terem me apoiado e não me deixado desistir do curso. E dedico um agradecimento especial a minha esposa, Amanda Hofmann, que sempre esteve ao meu lado, me apoiando e incentivando a continuar e não desistir, acredito que sem ela não estaria aqui hoje. Agradeço de forma especial também minha sogra, Guiomara Hofmann, por ter me ajudado sempre a organizar minhas ideias para que pudesse escrever este trabalho. Enfim, sem vocês nada disso seria possível.

Por fim, agradeço a todos os professores que fizeram parte da minha graduação, me transmitindo conhecimentos e experiências, para que pudesse me tornar um profissional de qualidade.

RESUMO

OLIVEIRA, Hythan. Desenvolvimento de um protótipo de sistema com arquitetura de microsserviços para empresa Maxiambiental treinamentos. 2022. 39 f. Monografia de Trabalho de Conclusão de Curso de Graduação – Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Guarapuava, 2022.

Problemas ambientais culminaram no surgimento de uma área de profissionalização para que leis de preservação sejam cumpridas. Empresas e instituições buscam capacitar esses profissionais, dentre elas está a Maxiambiental. Porém ela conta com um sistema antigo que precisa de uma atumatização em alguns processos, bem como a capacidade de adicionar novas funções, como por exemplo, geração automatica de certificados. Sendo assim, este trabalho buscou desenvolver um protótipo de sistema de gestão de cursos para a empresa Maxiambiental, com objetivo apresentar as vantagens e desvantagens no uso de microsserviços, como, por exemplo, em um sistema monolítico, se existe uma falha, em uma parte do sistema, todo ele fica comprometido, já em microsserviços, se um serviço para de funcionar outro pode continuar funcionando normalmente. Assim, foi desenvolvido um protótipo composto por dois microsserviços, um para gestão de cursos e outro para gestão de certificados e duas aplicações responsáveis pelo front-end, uma para área administrativa e outra para área pública.

Palavras-chave: Arquitetura de Microsserviços. Protótipo. Gestão de Cursos. Educação Ambiental.

ABSTRACT

OLIVEIRA, Hythan. Using microservices to manage short courses: A case study for company Maxiambiental.. 2022. 39 f. Monografia de Trabalho de Conclusão de Curso de Graduação – Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Guarapuava, 2022.

Environmental problems have culminated in the emergence of an area of professionalization so that preservation laws can be complied with. Companies and institutions seek to capacitate these professionals, among them is Maxiambiental. However, it has an old system that needs to be updated in some processes, as well as the ability to add new functions, such as the automatic generation of certificates. Thus, this work sought to develop a prototype of a course management system for the company Maxiambiental, with the objective of presenting the advantages and disadvantages in the use of microservices. For example, in a monolithic system, if there is a failure in one part of the system, the whole system is compromised, but in microservices, if one service stops working another can continue working normally. Thus, a prototype was developed consisting of two microservices, one for course management and the other for certificate management, and two applications responsible for the front-end, one for the administrative area and the other for the public area.

Keywords: Prototype. System. Microservices Architecture. Environmental Education.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AMQP	Advanced Message Queuing Protocol
CPF	Cadastro de Pessoas Físicas
HTTP	Hypertext Transfer Protocol
UX	User Experience

SUMÁRIO

1 INTRODUÇÃO

Desastres ambientais e problemas de poluição, que ameaçam a qualidade de vida, foram os motivos para a Conferência de Estocolmo de 1972, onde foi abordado a necessidade de consciência mundial, por parte de todos os indivíduos, com relação à preservação e conservação do meio ambiente. A partir disso, a expressão Educação Ambiental ganhou destaque no contexto político e pedagógico (RAMOS, 2001). Assim, no Brasil, surgiu uma área, voltada a cuidar das questões ambientais, prevista na Constituição Federal do Brasil, de 1988, art 225.

> "Todos têm direito ao meio ambiente ecologicamente equilibrado, bem de uso comum do povo e essencial à sadia qualidade de vida, impondo-se ao poder público e à coletividade o dever de defendê-lo e preservá-lo para as presentes e futuras gerações."

Desse modo, é de fundamental importância a educação ambiental, seja para instituições ou para civis. Assim, é necessário capacitar profissionais para conhecerem, empregarem e respeitarem as leis ambientais. Para isso, existem diversas categorias de cursos e treinamentos, como, por exemplo, cursos de graduação, pós-graduação e treinamentos de longa e curta duração.

A Maxiambiental[1] é uma empresa especializada em ministrar treinamentos presenciais na área ambiental. A empresa foi fundada em agosto de 2009 e a partir deste período já capacitou mais de 8.000 profissionais da área ambiental, em 150 turmas, com os cursos sendo ministrados em 16 estados e mais de 30 cidades brasileiras. Através de treinamentos de curto período, a empresa proporciona o aperfeiçoamento do conhecimento dos profissionais e estudantes das áreas ambientais, como Engenheiros Agrônomos, Civis, Ambientais, Florestais, Biólogos entre outros. Os cursos são aplicados por profissionais altamente qualificados e que possuem conhecimento teórico e prático, o que facilita no momento de compartilhar experiências com os alunos.

Atualmente a Maxiambiental conta com um sistema para gerenciamento de turmas e alunos que buscam cursar seus treinamentos, porém o mesmo encontra-se desatualizado. Possuindo um sistema que precisa de correções, pois alguns links e páginas não estão funcionando ou possuem conteúdos em branco e também falta um mecanismo de pesquisa na área dos cursos. Além disso, é preciso automatizar alguns processos, como por exemplo, na geração de certificados visto que hoje é algo manual.

Outro ponto que pode ser observado, é a necessidade de adição de novas funcionalidades, como por exemplo, uma área em que o estudante possa estar gerenciando seus dados, bem como por exemplo, a implementação de uma ferramenta de pagamento direto no sistema, o que vai agilizar o lado do usuário na hora de se cadastrar em um curso.

[1] https://maxiambiental.com/

Por estes motivos, a criação de um novo sistema é essencial para que se possa garantir que o mesmo possua funcionalidades que proporcionem mais agilidade e facilidade para o usuário durante a utilização. Desta forma, este trabalho contribuiu para o desenvolvimento de um protótipo de um sistema organizado em uma arquitetura de microsserviços, esperando trazer um sistema que aprensente alta disponibilidade, uma vez que não irá parar de funcionar em caso de falha por alguma parte do mesmo e espera-se que tal funcionamento afete a experiência do usuário final de forma positiva, já que contará com o sistema sempre disponível. Além de que nessa arquitetura, a implementação de novas funcionalidades vai ser facilitada, uma vez que possibilitará o desenvolvimento de um novo microsserviço na tecnologia que o desenvolvedor se sentir mais confortável, bastando apenas realizar as configurações para a conversação com as demais partes do sistema.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um protótipo de um sistema para o gerenciamento de treinamentos para a empresa Maxiambiental utilizando uma arquitetura distribuída em microsserviços.

1.1.2 Objetivos Específicos

- Organizar a arquitetura da aplicação para funcionamento no modelo de Microsserviços;
- Desenvolver o microsserviço para geração automaziada de certificados;
- Desenvolver uma aplicação administrativa para gerenciamento de cursos, turmas e conteúdos;
- Desenvolver uma aplicação para a área pública que possibilite ao usuário cadastrar e gerenciar suas informações.

2 SISTEMA UTILIZADO PELA MAXIAMBIENTAL

Atualmente a empresa Maxiambiental possui um sistema, para a gestão de turmas e alunos. Este capítulo apresenta uma breve descrição da área pública e da área administrativa do sistema.

2.1 ÁREA PÚBLICA

Por se tratar de um sistema que foi desenvolvido em 2009, a área pública conta com um visual desatualizado, dando a impressão de um site antigo, conforme mostrado na Figura 1. Além disso percebe-se que os formulários não contam com nenhuma validação nos campos, para evitar que seja digitado alguma informação incorreta, como por exemplo, uma letra no meio do CPF, e a notificação de um dado inválido só ocorre ao tentar finalizar o envio, conforme exemplificado na Figura 2.

Figura 1 – Sistema atual da empresa Maxiambiental.
Fonte: Maxiambiental

Figura 2 – Mensagem de erro ao cadastrar com campos inválidos.
Fonte: Maxiambiental

Outro ponto, com relação aos formulários é o fato de todos os campos serem exibidos em uma única página, o que da a impressão de ser muito extenso para o usuário, conforme a Figura 3.

Figura 3 – Formulário de inscrição muito extenso.
Fonte: Maxiambiental

O sistema não conta com uma área onde o usuário possa atualizar ou corrigir seus dados, ou ainda, cancelar sua inscrição de algum treinamento. Pela falta desse cadastro de

usuário, a cada novo curso que, por exemplo, o usuário X venha a realizar, precisará cadastrar suas informações novamente, gerando um retrabalho.

O sistema também não possui a função de realizar pagamentos direto na plataforma, sendo necessário que o usuário aguarde um e-mail com o boleto, ou outra forma de contato, para finalizar e confirmar sua inscrição conforme mostrado na Figura 4.

Figura 4 – Mensagem de sucesso de inscrição em um treinamento.
Fonte: Maxiambiental

2.2 ÁREA ADMINISTRATIVA

O espaço do sistema destinado ao administrador para gerenciamento dos conteúdos do site, bem como dos cursos e alunos, também apresenta-se com um visual desatualizado, além de contar com o mesmo problema de validação nos campos dos formulários.

Uma das falhas que pode ser explorada, durante o cadastro de turmas é a possibilidade de cadastrar uma nova turma sem definir um meio de pagamento ou definindo uma data que já tenha passado, o que vai impossibilitar que o usuário final consiga se cadastrar no treinamento, conforme exemplificado na Figura 5.

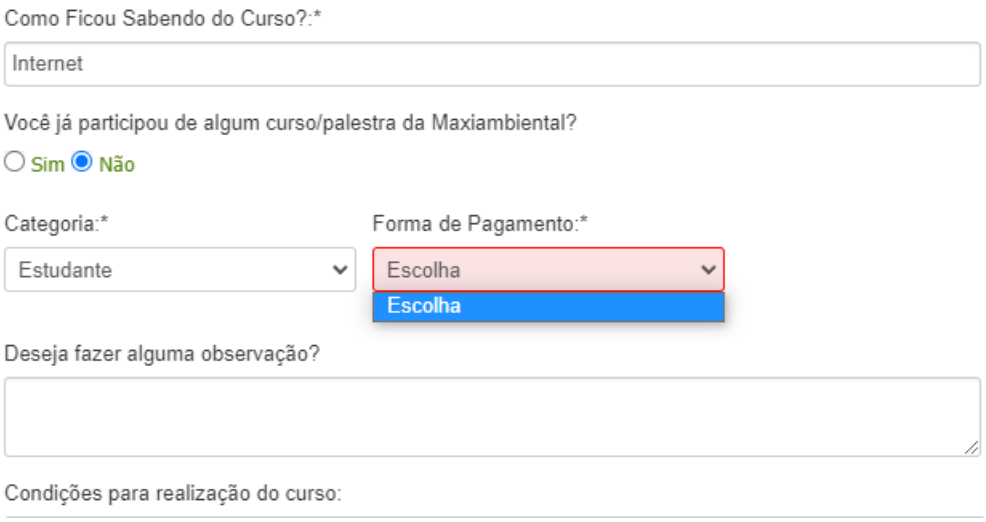

Figura 5 – Falha ao se cadastrar num curso.
Fonte: Maxiambiental

A Figura 6 demonstra que para chegar até o candidato o boleto para pagamento e confirmação da inscrição, é necessário que o administrador cadastre o valor e gere o boleto de forma manual, para só então enviar para o usuário. Este processo permite ainda o cadastro de um valor inválido, o que acarretará numa tela de erro, conforme a Figura 7.

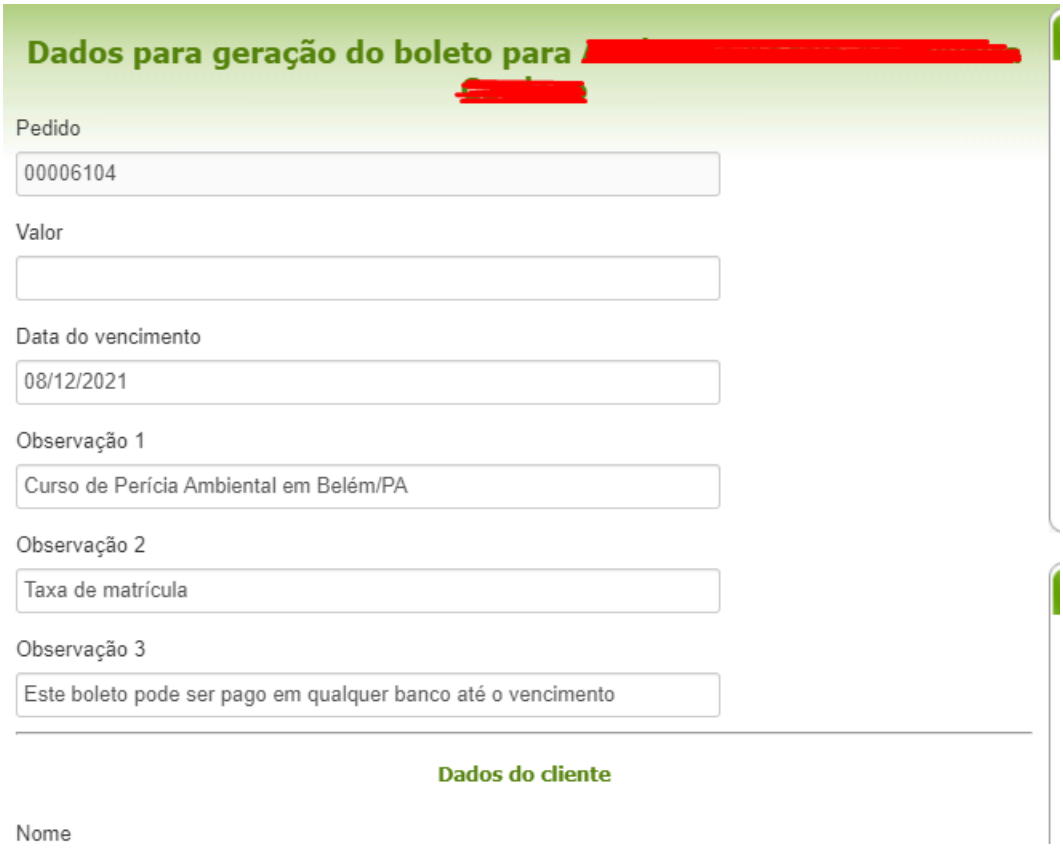

Figura 6 – Cadastro de boleto.
Fonte: Maxiambiental

Figura 7 – Falha ao gerar boleto.

Fonte: Maxiambiental

3 REFERENCIAL TEÓRICO

3.1 UX DESIGN

O aumento da competitividade e a preocupação em atrair clientes, fez com que surgisse uma área focada na experiência do usuário ou *User Experience* (UX), em inglês. Tal área preocupa-se com todo o processo de uso de um determinado produto, e o mesmo se aplica a sites e sistemas, não limitando-se a uma aparência bonita, mas também, na forma como o usuário utilizará o sistema e chegará ao seu objetivo (TEIXEIRA, 2014).

Ainda segundo Teixeira (2014) a simplicidade e usabilidade, algumas das diretrizes do UX Design, são de suma importância para garantir o sucesso de um produto, preocupando-se em colocar o usuário final no centro do processo.

Algumas formas de validar um produto é através de *Wireframes*, que consiste em ser um guia visual que representa um esboço de uma página, contendo a estrutura e organização do conteúdo, conforme mostra a Figura 9. Além de fazer um teste de usabilidade, que consiste em entrevistar o usuário durante a utilização de uma funcionalidade, ou do sistema como um todo, garante uma validação se o fluxo, aparência e funcionalidade estão de acordo com o que ele procura (TEIXEIRA, 2014).

Figura 8 – Exemplo de um *Wireframe*
Fonte: O Autor (2021).

Atualmente a preocupação com a experiência do usuário ao desenvolver um produto é fundamental para garantir o sucesso ou fracasso do mesmo.

3.2 SISTEMAS WEB MONOLÍTICOS VS MICROSSERVIÇOS

Um sistema monolítico possui sua arquitetura implementada em um único processo, ou seja, todas as funcionalidades da aplicação estão acopladas em um único fluxo, conforme mostrado na Figura 10 (SILVA, 2016). Por exemplo, se o processo C parar de funcionar toda aplicação irá parar de funcionar.

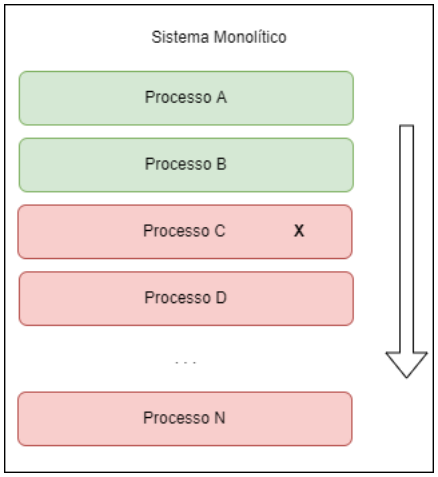

Figura 9 – Exemplo de um sistema monolítico com falha.
Fonte: O Autor (2021).

Ainda em um sistema monolítico, a cada nova funcionalidade desenvolvida, precisa ser testado o sistema como um todo para garantir que alguma outra parte não tenha parado de funcionar, também, a cada nova funcionalidade a complexidade do código aumenta, já que tudo está acoplado (RODRIGUES, 2017). Já em uma aplicação com sua arquitetura baseada em microsserviços, cada funcionalidade é empacotada independente das demais, evitando que se um processo parar de funcionar a aplicação continuará rodando, conforme mostrado na Figura 11 (SILVA, 2016).

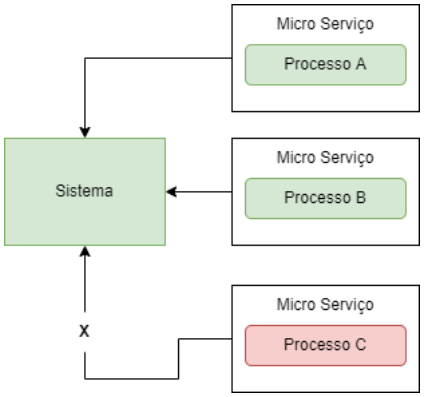

Figura 10 – Exemplo de um sistema baseado em microsserviços
Fonte: O Autor (2021).

Nesse tipo de arquitetura a implementação de novas funcionalidades é facilitada, uma vez que podem ser utilizadas linguagens diferentes para implementação e não há preocupação com o comportamento do resto do sistema. Outro ponto positivo é visto na hora de escalonar a aplicação, já que podem ser escolhidos as partes ou serviços que precisam ser melhorados (RODRIGUES, 2017).

Ainda segundo RODRIGUES (2017) como desvantagem, um sistema em microsserviços está no gerenciamento de dependências de cada microsserviços, uma vez que, conforme o sistema cresce, cresce também a quantidade de microsserviços implementados. Outra preocupação é na comunicação entre eles, garantir que o serviço A comunique-se com o serviço B.

3.3 TECNOLOGIAS DE DESENVOLVIMENTO

Nesta seção serão abordados as tecnologias utilizadas para o desenvolvimento deste trabalho.

3.3.1 Docker

Docker é uma ferramenta que virtualiza contêineres, e estes, possuem seus próprios softwares e arquivos de configuração independentes, podendo se comunicar entre si, e foi criado em 2013 por Solomon Hykes (DOCKER, 2013).

Docker se tornou uma tendência, quase que uma obrigatoriedade entre o meio de desenvolvedores, como mostrado na Figura 12. A procura pela ferramenta aumentou consideravelmente nos últimos anos, de acordo com o resultado do Google Trends[1]. Afinal de contas, o Docker possibilita que o desenvolvimento e a replicação de um sistema, se torne algo mais prático, uma vez que uma aplicação pode possuir diversos contêineres e estes as mais diversas linguagens, tecnologias e configurações, e todos podem conversar entre si para o funcionamento da aplicação como um todo (GOMES, 2017).

[1] Google Trends é uma ferramenta do Google que mostra os termos procurados em um determina período de tempo (GOOGLE, 2021)

Figura 11 – Interesse no assunto 'Docker'.
Fonte: Google Trends.

E como gerenciar tantos contêineres durante o desenvolvimento? Para fazer a orquestração de todos os contêineres necessários para o desenvolvimento de uma aplicação existe o Docker Compose, que é uma ferramenta utilizada para definir e executar diversos contêineres através de um arquivo de configuração (DOCKER, 2013).

3.3.2 Node.js

Node.js é uma plataforma para execução de JavaScript[1] no *back end* criado por Ryan Dahl em 2009 (OPENJS, 2021).

Uma grande vantagem para se utilizar Node.js para desenvolvimento de aplicações é o fato de utilizar JavaScript como linguagem, isso diminui a curva de aprendizagem, uma vez que a linguagem é a mesma do *client-side*. Outra vantagem é uma comunidade extremamente ativa, ou seja, o desenvolvedor facilmente encontra respostas para possíveis dúvidas que possam surgir durante o desenvolvimento de uma aplicação (PEREIRA, 2016).

Conforme mostra o gráfico do Google Trends na Figura 13, a busca por JavaScript, linguagem utilizada no Node.js, superou, a partir da escrita desse trabalho, a busca de outras linguagens de programação como PHP e Ruby, o que reforça a questão de uma comunidade mais ativa.

[1] JavaScript é uma linguagem de programação interpretada e baseada em objetos, utilizada para rodar scripts em páginas Web (MOZILLA, 2021)

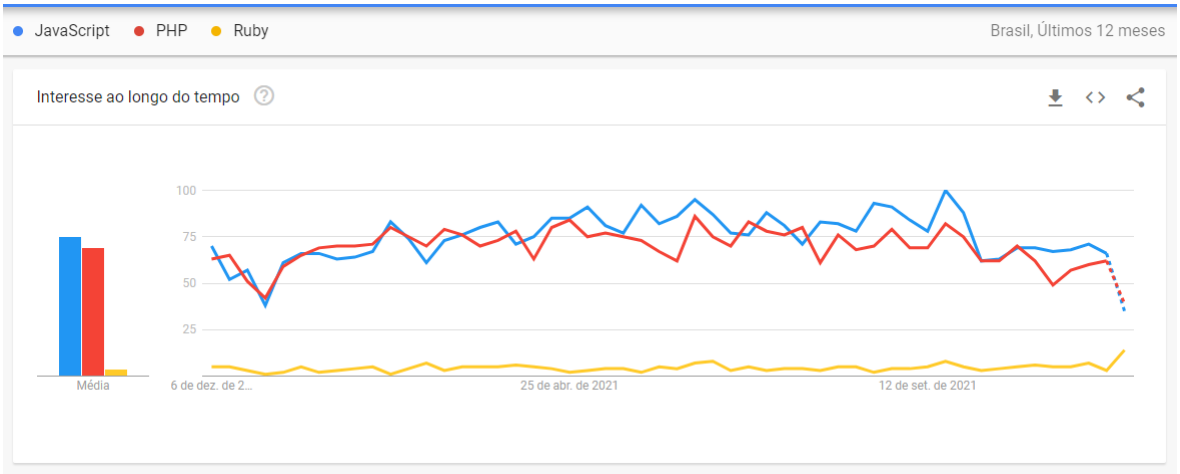

Figura 12 – Interesse no assunto JavaScript em comparação a PHP e Ruby.
Fonte: Google Trends.

3.3.3 Express

Express é um *framework back-end* minimalista e flexível que conta com recursos robustos para o desenvolvimento de aplicações Web JavaScript com o Node.js (OPENJS, 2010). Conta com recursos como: roteamento robusto, foco em alto desempenho, alta cobertura de testes, auxiliares de HTTP (*Hypertext Transfer Protocol*), entre outros (OPENJS, 2010).

E por que utilizar Express como framework? O Express permite um desenvolvimento mais ágil, uma vez que é simples de ser estudado, e trás uma liberdade maior na forma como o desenvolvedor pode organizar seu código (PEREIRA, 2016).

3.3.4 ReactJS

Pensando no desenvolvimento *front-end* de uma aplicação existe o React, uma biblioteca JavaScript criada pelos desenvolvedores do Facebook, que tem como objetivo principal permitir o desenvolvimento de interfaces interativas de maneira facilitada (FACEBOOK, 2013). É uma tecnologia que procura trazer velocidade, simplicidade e escalabilidade e que conta com a possibilidade de criação e reutilização de componentes (LINS, 2019).

3.3.5 VueJS e NuxtJS

Vue é um framework JavaScript que possuí um ecossistema pronto para o desenvolvimento de interfaces e aplicações front-end. É baseado no modelo de componentes, o que promete garantir eficiencia e agilidade no desenvolvimento (YOU, 2022).

O Vue vem crescendo e ganhando espaço na comunidade de desenvolvedores, com cerca de mais de 1.5 milhão de usuários e já esta presente em grandes organizações como NASA, Apple, Google, etc (YOU, 2022).

Nuxt é um framework Vue para desenvolvimento de aplicações front-end. Ele é utilizado para agilizar e simplificar a configuração de um ambiente de desenvolvimento para a criação de aplicações em Vue (CHOPIN; CHOPIN, 2022).

Considerado intuitivo, de fácil aprendizagem e que promete trazer uma ótima experiencia ao desenvolvedor, descomplicando as configurações iniciais, automatizando configurações de roteamento, componentizando as páginas, trazendo uma organização de código intuitiva e de fácil compreensão (CHOPIN; CHOPIN, 2022).

3.3.6 NestJS

NestJS é um framework JavaScript para criação de aplicações Node.js para o back-end. Possibilita a implementação de aplicações escaláveis e eficientes, trazendo uma abstração da estrutura do código, melhor que outros frameworks como Express e Fastify (MYSLIWIEC, 2022).

O NestJS busca resolver um problema que outros frameworks de JavaScript back-end possuem, a arquitetura do projeto. Ele já fornece ao desenvolvedor uma arquitetura e estrutura de código pronta, e configurada, facilitando e agilizando a produtividade na hora de desenvolver (MYSLIWIEC, 2022).

3.4 FERRAMENTAS DE DESENVOLVIMENTO

3.4.1 Kanban

No desenvolvimento de aplicações é de suma importância que o desenvolvedor, ou a equipe, que o estiver fazendo, busque por utilizar tecnologias e ferramentas que otimizem e organizem os processos, a fim de fazer com que prazos de entrega sejam cumpridos. A implementação de uma metodologia ágil é uma forma de auxiliar para que isso ocorra (ROSA, 2014).

Dentre os métodos ágeis existentes, existe o artefato Kanban, que em tradução literal significa "cartão" em japonês, e surgiu dos sistemas de fábricas, onde consiste em gerenciar o fluxo de trabalho, separando em cartões coloridos as tarefas, ou processos, que devem ser feitos e adicionados num quadro, dessa forma todo o processo fica visível (MARIOTTI, 2008).

O processo tradicional de utilização do Kanban é separar as tarefas em cartões, que são adicionados em um quadro, que possui divisões em colunas, conforme mostrado na Figura 8.

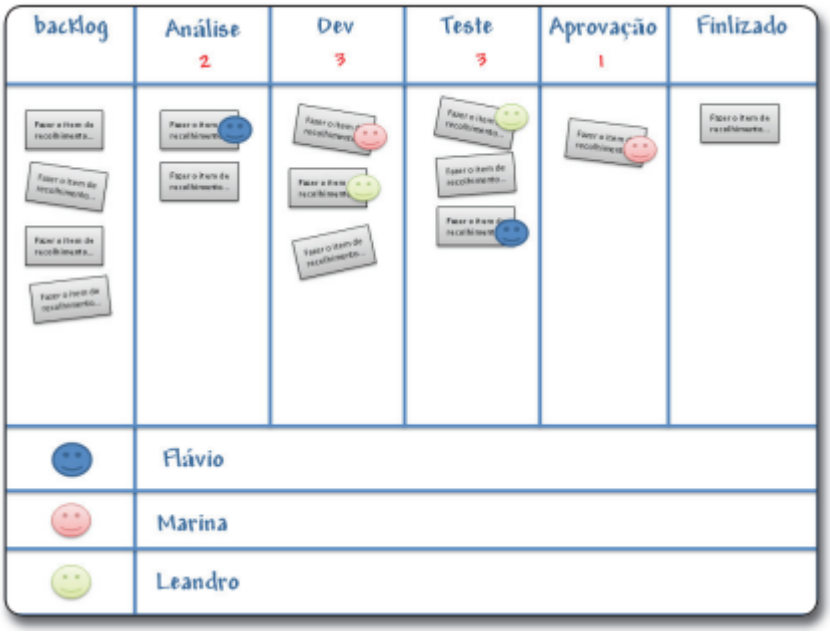

Figura 13 – Ilustração de um Kanban
Fonte: MARIOTTI (2008)

Ainda segundo o artigo de MARIOTTI (2008) o kanban garante uma transparência que torna fácil a identificação de gargalos e tarefas que estão há muito tempo em desenvolvimento ou na fila. Desta forma, toda a equipe fica ciente do processo em si, e pode tomar as devidas providências.

3.4.2 PostgresSQL

PostgresSQL é um Sistema de Gerenciamento de Banco de Dados (SGBD) relacional, ou seja, ele é responsável pelo armazenamento e consulta de dados que forem salvos por alguma aplicação, gratuito e de código aberto (POSTGRESQL, 1996).

O Postgres conta com recursos que vão garantir integridade de todos os dados que forem salvos, sendo ideal para aplicações de quaisquer porte, sejam elas, pequenas, médias ou grandes (MILANI, 2008). Dentre os recursos pode-se destacar: integridade de dados (restrições de exclusão, bloqueios explícitos, bloqueios consultivos, etc), desempenho (indexação, indexação avançada, particionamento de tabela, etc), confiabilidade (log de gravação antecipada, recuperação pontal, etc), entre outros (POSTGRESQL, 1996).

3.4.3 RabbitMQ

RabbitMQ é um dos seridores de mensageria mais utilizados no mundo. Suporta diversos protocolos de mensagens e é leve e de fácil implantação. Muito utilizado em aplicações distribuidas para atender aos requisitos de alta disponibilidade (.INC, 2022).

O RabbitMQ possibilita a implementação de mensagens assíncronas entre as aplicações, e que as aplicações possuam diferentes linguagens, o que proporciona à equipe de desenvolvedores uma experiencia agradável (.INC, 2022).

4 METODOLOGIA DE DESENVOLVIMENTO

A primeira etapa para o início do desenvolvimento do projeto foi a realização de uma reunião com o administrador do sistema, a fim de levantar os principais problemas e dificuldades enfrentados na sua utilização. Todos os problemas e melhorias que foram apontados na reunião foram divididos em cartões e inseridos em um quadro Kanban, através da ferramenta Trello[1], conforme a Figura 14.

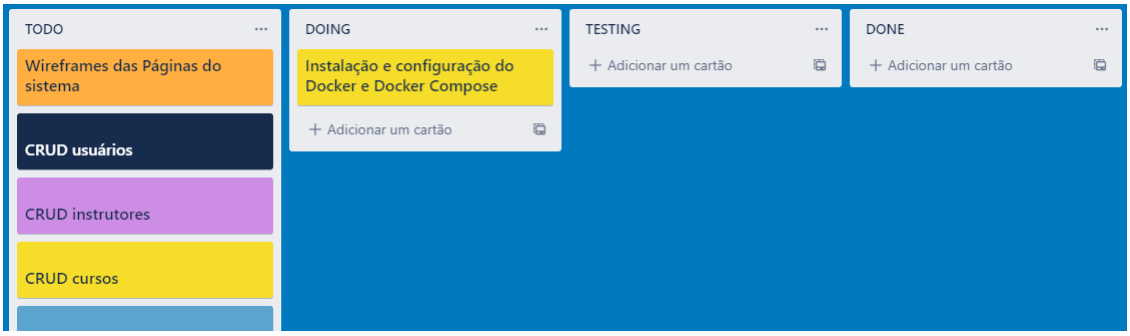

Figura 14 – Exemplo de um quadro Kanban
Fonte: Autor (2022).

Durante o desenvolvimento do projeto todas as tarefas que foram feitas, ficaram na aba *TODO*, em ordem de prioridade e quando foi iniciada a implementação da tarefa, esta foi movida para aba *DOING*. Após finalizada a tarefa passou para a aba *TESTING*, onde o administrador realizou o teste da funcionalidade desenvolvida e então sua aprovação, e caso não tenha sido aprovada, retornou para *DOING* com as correções solicitadas. Após aprovada, a tarefa foi movida para a aba *DONE*.

Antes de iniciado a implementação do sistema foi configurado o ambiente de desenvolvimento, para isso foi utilizado o Docker juntamente com o Docker Compose para a orquestração dos diversos contêineres necessários. Neste ambiente foi configurado o Nodejs, o banco de dados PostgresSQL e configurados os frameworks NestJS, VueJS, NuxtJS, bem como o servidor de mensageria RabbitMQ e demais tecnologias.

Durante o desenvolvimento do projeto, as tecnologias ReactJS e Express, escolhidas inicialmente para serem utilizadas, foram alteradas por NuxtJS e NestJS. Essa mudança fez-se necessária devido ao tempo de aprendizagem, sendo que, estas são tecnologias que já possuem todas as configurações iniciais para o desenvolvimento, bem como, organização de código mais intuitivas. Por exemplo, comparando o Express com o NestJS necessitaria de muitas configurações manuais e instalação de bibliotecas para o início e durante o desenvolvimento, já

[1]Trello é um gerenciador de tarefas visual, que organiza as mesmas em cards, o que possíbilita toda equipe ter conhecimento das demandas e sobre como esta o andamento do projeto(ATLASSIAN, 2022).

o NestJS já contava com toda essa configuração inicial, agilizando o início do desenvolvimento do projeto.

Todos estes pontos foram observados após terem sido realizado estudos e implementações com Express, bem como com React, além de ter sido testado outros servidores de mensageria, como por exemplo o Kafka, para orquestramento e funcionamento da arquitetura. E após todos estes testes, as tecnologias escolhidas foram as descritas anteriormente.

Com todas as configurações feitas e tecnologias configuradas, o processo de desenvolvimento das tarefas se deu por implementação de cada funcionalidade, que foi submetida a testes de código com auxilio do JestJS[2], posteriormente submetida a um ambiente de teste local e avaliada pelo admnistrador do sistema, aprovada a funcionalidade, passava-se para o desenvolvimento da próxima.

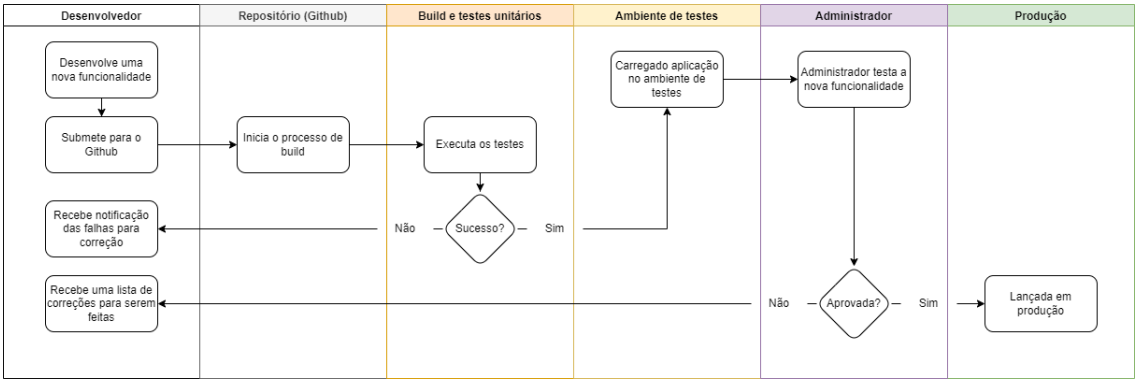

Figura 15 – Fluxograma do processo de desenvolvimento do sistema
Fonte: Autor (2021).

A Fígura 15 exemplifica o processo original para a implementação das funcionalidades. Percebe-se que possuia etapas a mais, como por exemplo, a implementação e automatização de testes no repositório remoto (Github), bem como a criação de um ambiente de testes e um ambiente de produção. Porém, pela mudança de foco para um protótipo, bem como os esforços feitos nas etapas iniciais, tais processos não foram implementados por não impactarem no resultado final do projeto.

[2] Jest é um *framework* de testes em JavaScript que busca trabalhar na simplicidade para implementação dos testes (FACEBOOK, 2022).

5 RESULTADOS

O presente capítulo tem por objetivo explanar a respeito dos resultados obtidos ao densenvolver o protótipo do novo sistema da empresa Maxiambiental.

5.1 ORGANIZAÇÃO DO PROJETO

Durante todo o desenvolvimento do sistema, foi utilizado o método Kaban para organizar as tarefas que precisavam ser implementadas. Atráves da ferramenta Trello, foi possível gerenciar e ter uma visão completa do que precisava ser feito e do que já estava pronto. Isso garantiu que nenhuma parte importante do sistema ficasse esquecida. A Figura 16, demonstra o quadro utilizado para organizar e gerenciar as tarefas que foram realizadas.

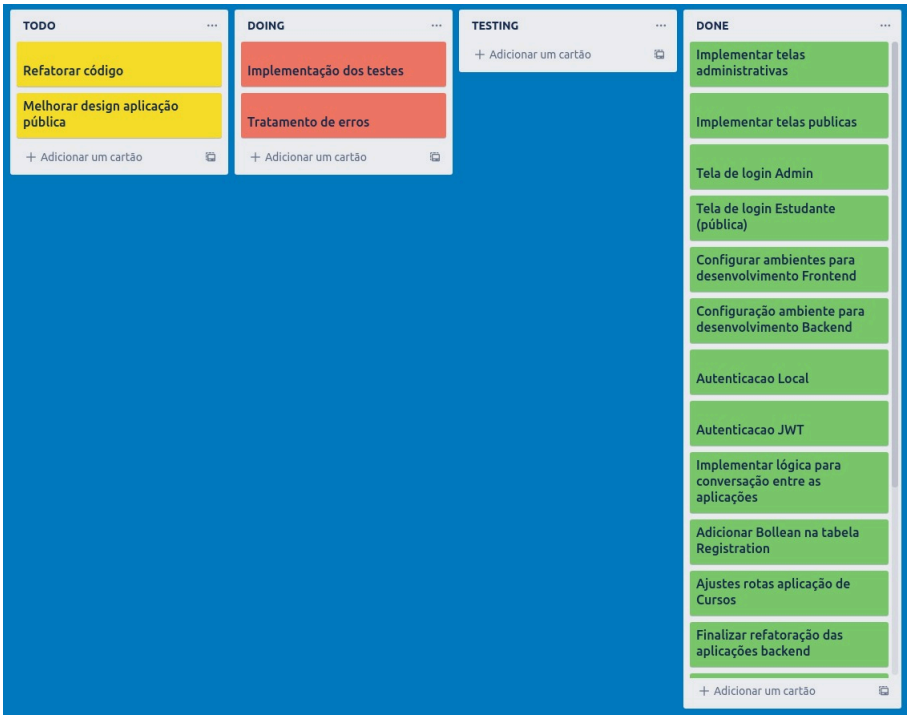

Figura 16 – Quadro Kanban do projeto da Maxiambiental.
Fonte: Autor (2022).

Nota-se que ainda possuem tarefas a serem concluídas, e isso se deve ao fato de que diante de um projeto real há vários setores trabalhando a fim de sanar a demanda que o mesmo exige. Portanto, com o objetivo de entregar as principais funcionalidades, e devido ao tempo dedicado para o estudo e entendimento das tecnologias necessárias para o desenvolvimento deste projeto, algumas tarefas ficaram pendentes, mas não comprometem o funcionamento do protótipo do sistema.

5.2 FUNCIONALIDADES

Durante os estudos iniciais esperava-se desenvolver algumas funcionalidades para o protótipo, como por exemplo, separar a arquitetura em microsserviços de gestão de cursos, certificados, autenticação, pagamento e desenvolver toda a parte de gestão dos dados do sistema pelo administrador na aplicação administrativa. Além disso, esperava-se implementar para o usuário, na aplicação pública, toda a parte de gestão de seus dados, bem como, visualização de inscrições feitas, status a respeito dessas inscrições, e pagamentos realizados. Entretanto, devido ao tempo utilizado para estudos e testes e a complexidade que a arquitetura de microsserviços apresentou, algumas delas não foram implementadas.

As funcionalidades selecionadas para que se conseguisse ter o fluxo principal do protótipo do sistema, bem como, conseguir que a arquitetura funcionasse, foram as de desenvolver os microsseriços de cursos e certificados, simplificar a área administrativa para que se fizesse a gestão dos dados principais e na aplicação pública apenas foi implementado um controle de acesso para o usuário e o fluxo de cadastro em uma turma, bem como a visualização de seus certificados.

Outro ponto que vale destacar, é que devido ao tempo dedicado aos estudos, questões de *design* e aparência do protótipo, não foram priorizadas, sendo como principal foco o funcionamento do mesmo.

5.3 ARQUITETURA

Durante a organização e estudos a respeito do funcionamento de uma aplicação com arquitetura organizada em microsserviços, chegou-se ao seguinte resultado, no *back-end* foi desenvolvido uma API para recebimento das requições, esta comumente chamada de *gateway* (portão de entrada), afim de direcioná-las conforme a necessidade, através do uso do servidor de mensageria RabbitMQ. Além do *gateway*, têm-se as aplicações (microsserviços) de cursos e certificados, cada uma contando com seu banco de dados com as informações relevantes para seu funcionamento, conforme ilustrado na Figura 17.

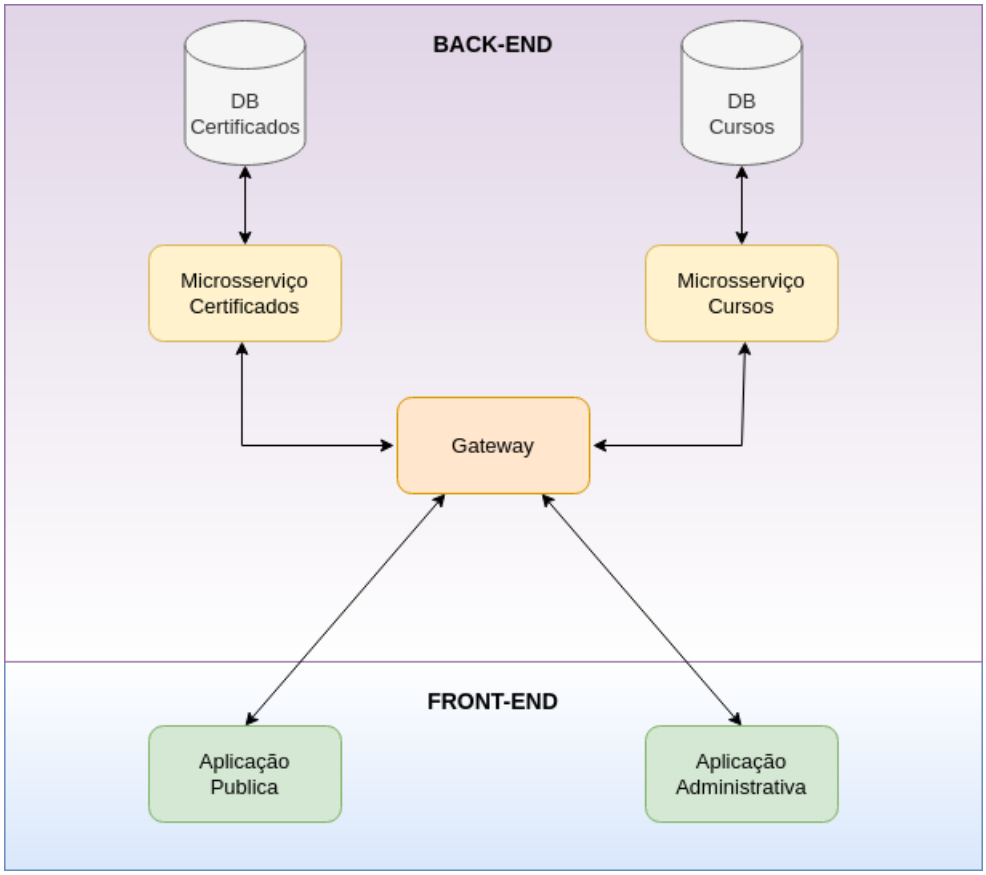

Figura 17 – Arquitetura do novo sistema Maxiambiental.
Fonte: Autor (2022).

A conversação entre as aplicações é garantida pela ferramenta RabbitMQ. Na configuração do projeto, foi implementada usando o seu principal protocolo o AMQP[1], este responsável por fazer a entrega das mensagens entre as aplicações, além de garantir que as mensagens não se percam em caso de um microsserviço estiver indisponível no momento do envio da mensagem, pois mantém a mensagem na fila até que o serviço volte a funcionar.

Essa arquitetura, permite que a aplicação não pare de funcionar caso ocorra algum erro com uma determinada parte dela, como por exemplo, um erro no acesso ao banco de dados da aplicação de cursos, não irá impedir que o usuário consiga visualizar seus certificados, e caso o contrário aconteça, o usuário ainda conseguirá acessar os cursos, turmas e até mesmo fazer inscrições.

Além disso, na parte das aplicações *front-end*, foi implementado duas aplicações distintas, uma para a área pública, site em que qualquer usuário consegue acessar, e outra para a aplicação administrativa, esta direcionada aos administradores do sistema.

[1]AMQP (*Advanced Message Queuing Protocol*) é um protocolo de internet, para envio de mensagens entre diferentes aplicações, que garante segurança e confiabilidade (OASIS, 2022).

5.4 BANCO DE DADOS

A organização e implementação de um banco de dados de forma correta, garante o bom funcionamento e velocidade para a aplicação. Pensando nisso, e considerando oberter as principais funcionalidades para o desenvolvimento do protótipo, chegou-se às seguintes modelagens ilustradas nas Figuras 18 e 19.

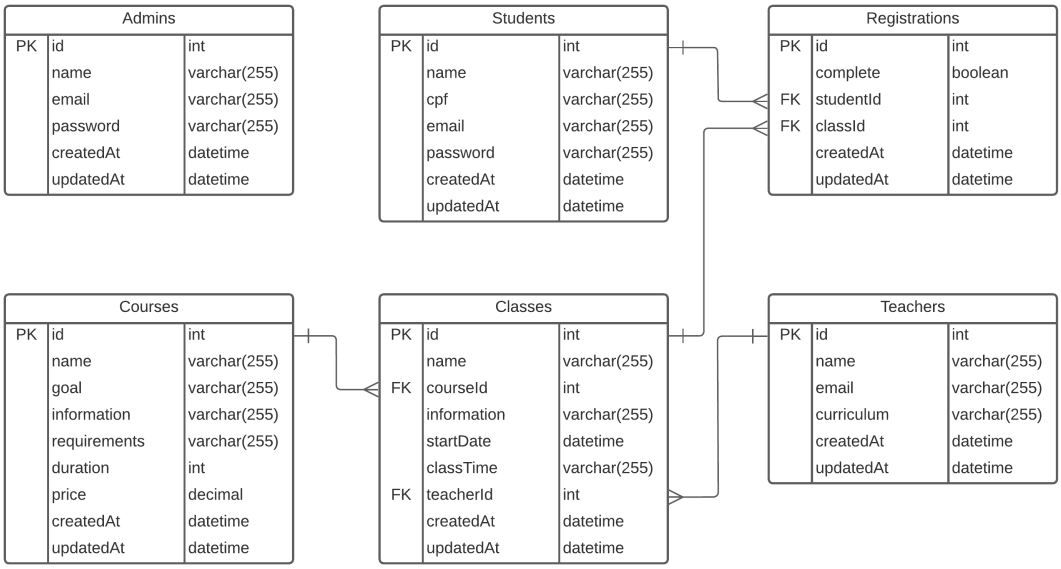

Figura 18 – Banco de dados da aplicação Cursos.
Fonte: Autor (2022).

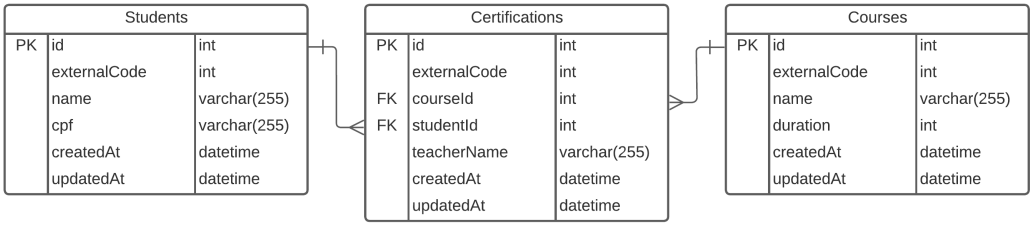

Figura 19 – Banco de dados da aplicação Certificados.
Fonte: Autor (2022).

Percebe-se que em ambos os bancos de dados têm-se tabelas "iguais", e isso ocorre uma vez que a arquitetura desenvolvida foi a de microsserviços, assim faz-se necessário que todas as aplicações tenham conhecimento de certos dados, para que possam garantir a integridade em caso de falha de alguma aplicação.

5.5 DESENVOLVIMENTO

A presente subção tem por objetivo explanar a respeito do desenvolvimento das telas e funcionalidades do sistema, bem como desafios encontrados durante sua execução.

5.5.1 CONFIGURAÇÃO DO AMBIENTE

Para dar início ao desenvolvimento de qualquer aplicação, é necessário a realização de configurações, como por exemplo, instalação de bibliotecas, tecnologias, editores de código, entre outros, e com o sistema da Maxiambiental não foi diferente.

Inicialmente precisou-se instalar e configurar o Docker e Docker-compose, para que fosse possível encapsular o ambiente de densenvolvimento de cada aplicação, dessa forma, foi possível isolar as tecnologias, banco de dados e bibliotecas necessárias para o desenvolvimento de cada um dos microsserviços e aplicações *front-end*.

A Figura 20[1] apresenta a configuração realizada para o desenvolvimento das aplicações *back-end*. Foi feito a separação em containêres de cada um dos microsserviços, *gateway*, RabbitMQ, além dos containêres dos banco de dados de cada microsserviço. Também fez-se necessário a configuração da *network*, pela qual os containeres conversam, além da configuração dos volumes (espaço em disco dedicado para armazenamento dos dados). Dentro dos conteinêres dos microsserviços, foi instalado e configurado o *framework* NestJS, para o desenvolvimento da API.

[1] Link: https://github.com/hythan/app-tcc/blob/master/docker-compose.yml

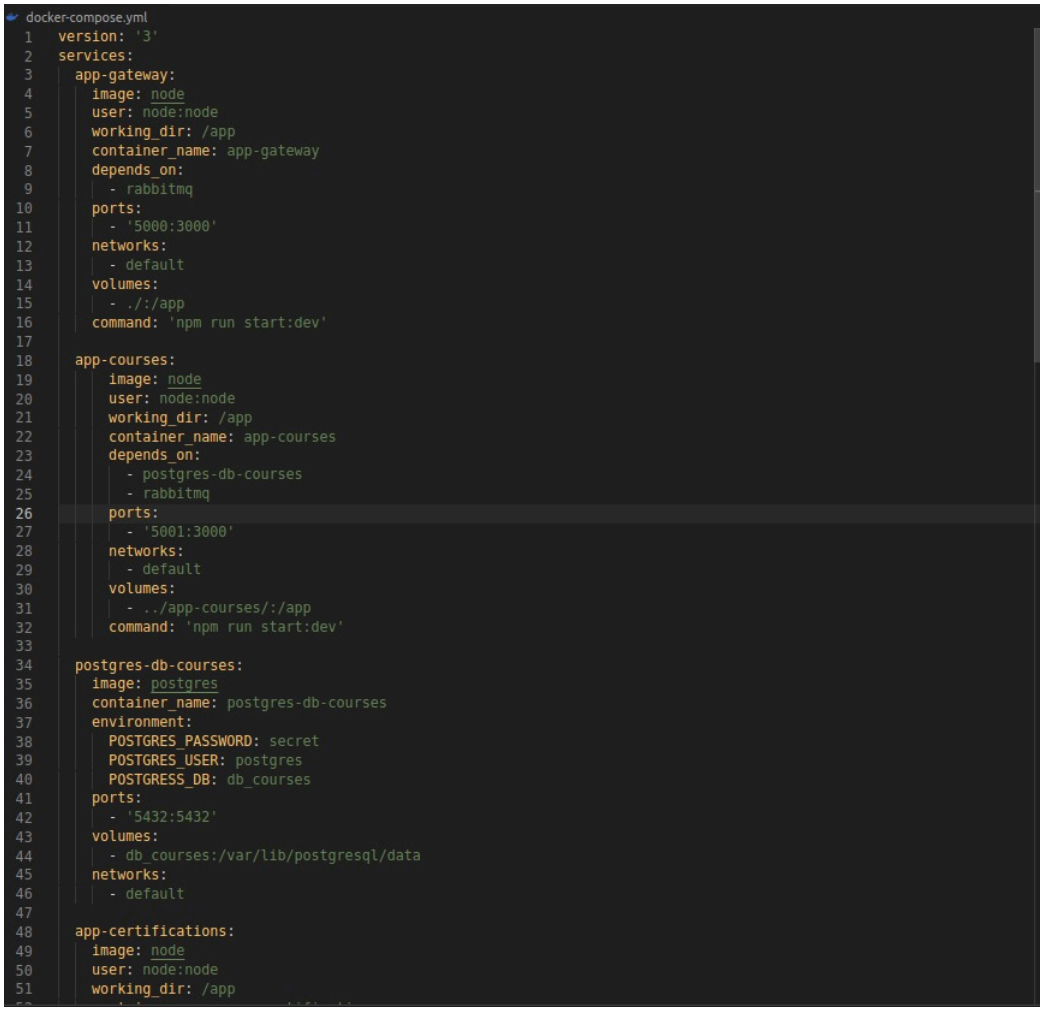

Figura 20 – Docker-compose *back-end*.
Fonte: Autor (2022).

Nas aplicações *front-end*, a configuração do Docker-compose necessitou apenas da instalação e configuração do *framework* NuxtJS, uma vez que essas aplicações são apenas responsáveis por consumir os dados da API. As Figuras 21[1] e 22[2], apresentam a configuração da aplicação pública e administrativa.

[1] Link: https://github.com/hythan/front-public/blob/master/docker-compose.yml
[2] Link: https://github.com/hythan/front-admin/blob/master/docker-compose.yml

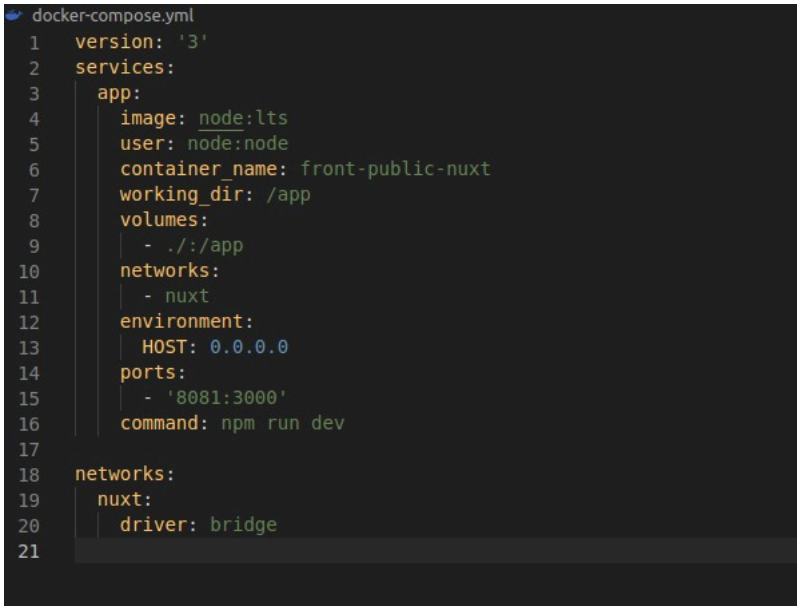

Figura 21 – Docker-compose *front-end*: aplicação pública.
Fonte: Autor (2022).

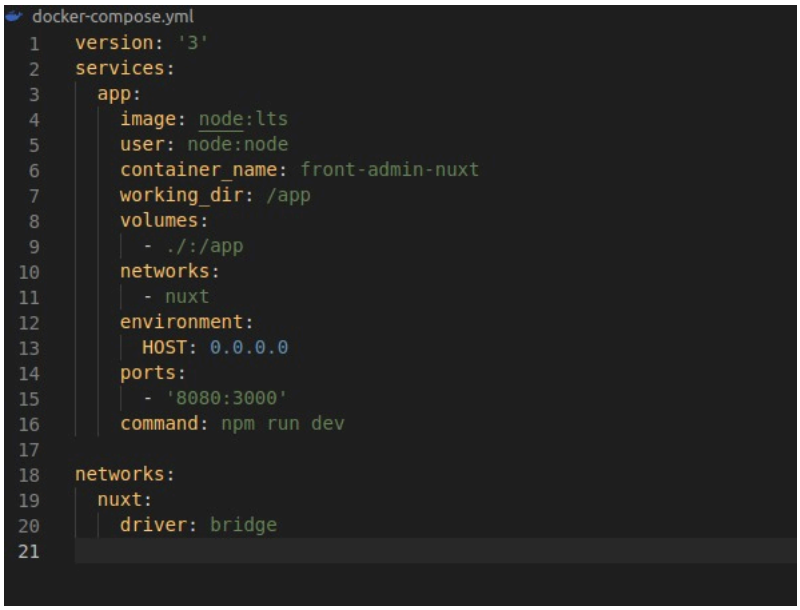

Figura 22 – Docker-compose *front-end*: aplicação administrativa.
Fonte: O Autor (2022).

5.5.2 DESENVOLVIMENTO DAS TELAS E FUNCIONALIDADES

No início da implementação do sistema, a primeira dificuldade enfrentada, foi a de entender e conseguir desenvolver o código para que a aplicação funcionasse de acordo com uma arquitetura de microsserviços. Tal problema, necessitou de um estudo aprofundado, uma vez que essa arquitetura possui uma complexidade maior para ser implementada. Para isso, a utilização do RabbitMQ, para orquestrar e gerenciar os envios e recebimentos de mensagens entre as

aplicações, foi essencial para seu funcionamento. Estas mensagens, subistituem uma requisação tradicional à uma API, nela as requisições são centralizadas no *gateway*, e este, é responsável por disparar as mensagens com as requisições. Já o servidor RabbitMQ, disponibiliza as mensagens numa fila, em que o microsserviço, que "atende" determinada fila e mensagem, recebe a requisição e responde com o dado solicitado. As Figuras 23, 24, 25, 26 e 27, exemplificam o código para a implementação do fluxo de funcionamento dos microsserviços.

```
}

@Get()
async findAll(@Query() where?: any) {
  return await this.classesService.findAll(where);
}
```

Figura 23 – Rota tradicional de *get* no *gateway*.
Fonte: Autor (2022).

```
async findAll(where?: any) {
  return await this.client.send('find-all-classes', { where });
}
```

Figura 24 – *Service* responsável por enviar a mensagem para a fila.
Fonte: Autor (2022).

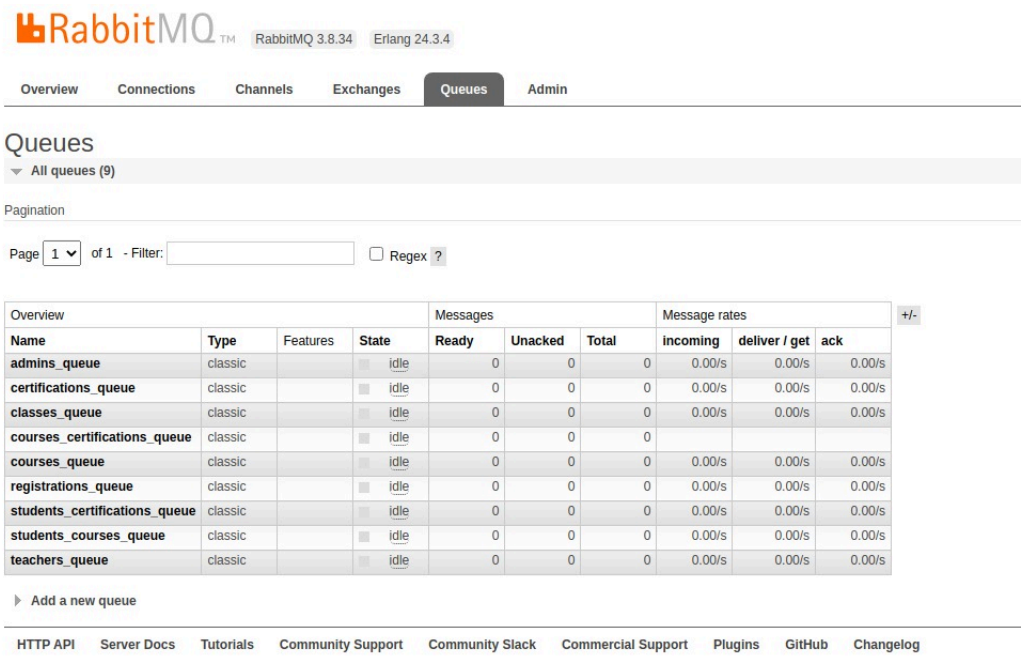

Figura 25 – Painel do RabbitMQ que mostra as filas e mensagens.
Fonte: Autor (2022).

```
app.connectMicroservice({
  transport: Transport.RMQ,
  options: {
    urls: ['amqp://admin:admin@rabbitmq:5672'],
    queue: 'classes_queue',
    queueOptions: {
      durable: false,
    },
  },
});
```

Figura 26 – Configuração no microsserviço para se conectar à uma fila.
Fonte: Autor (2022).

```
@MessagePattern('find-all-classes')
findAll(@Payload() payload: any) {
  return this.classesService.findAll(payload.where);
}
```

Figura 27 – Função do microsserviço responsável por tratar determinada mensagem.
Fonte: O Autor (2022).

Pensando ainda no funcionamento da arquitetura, foi necessário fazer com que certos dados fossem salvos em mais de uma tabela e em bancos de dados diferentes, como por exemplo, no caso de cadastrar um novo curso. Assim, é necessário salvar alguns dados no banco de certificados, para ter acesso aos mesmos, num cenário em que a aplicação de cursos pare de funcionar. A Figura 28 demonstra o código implementado para tal funcionamento.

```
async create(createCourseDto: CreateCourseDto) {
  try {
    const response = await lastValueFrom(
      await this.clientCourse.send('create-course', {
        data: createCourseDto,
      }),
    );

    this.clientCerfications
      .send('create-certifications-course', {
        data: createCourseDto,
        id: response.id,
      })
      .subscribe();

    return 'Sucessfully created course';
  } catch (error) {
    console.log(error);
    return error;
  }
}
```

Figura 28 – Código para salvar o curso em mais um banco de dados.
Fonte: Autor (2022).

Foi necessário enviar duas mensagens diferentes, e para filas distintas, para que ambos os microsserviços recebem-se os dados e realizassem seu tratamento. Para o caso de uma das

aplicações cair, supondo que seja a aplicação de certificados, a mensagem ficará na fila até que a aplicação retorne e executará a criação do curso, como é o caso desse exemplo.

Com êxito no desenvolvimento básico da arquitetura e com ela rodando para receber as requisições, a implementação das aplicações *front-end* foi iniciada. Primeiramente foi desenvolvido a aplicação administrativa, responsável por todo o gerenciamento dos cursos, turmas, professores, estudantes, inscrições e geração de certificados. As Figuras 29, 30, 31 e 32, apresentam o resultado das telas de gerenciamento de administradores, professores, alunos e cursos, respectivamente. Tais telas aprensentam a listagem dos dados cadastrados, e possuem os botões que direcionam para uma tela de visualização e edição, semelhantes à apresentada na Figura 33, bem como um botão responsável pela exclusão do dado.

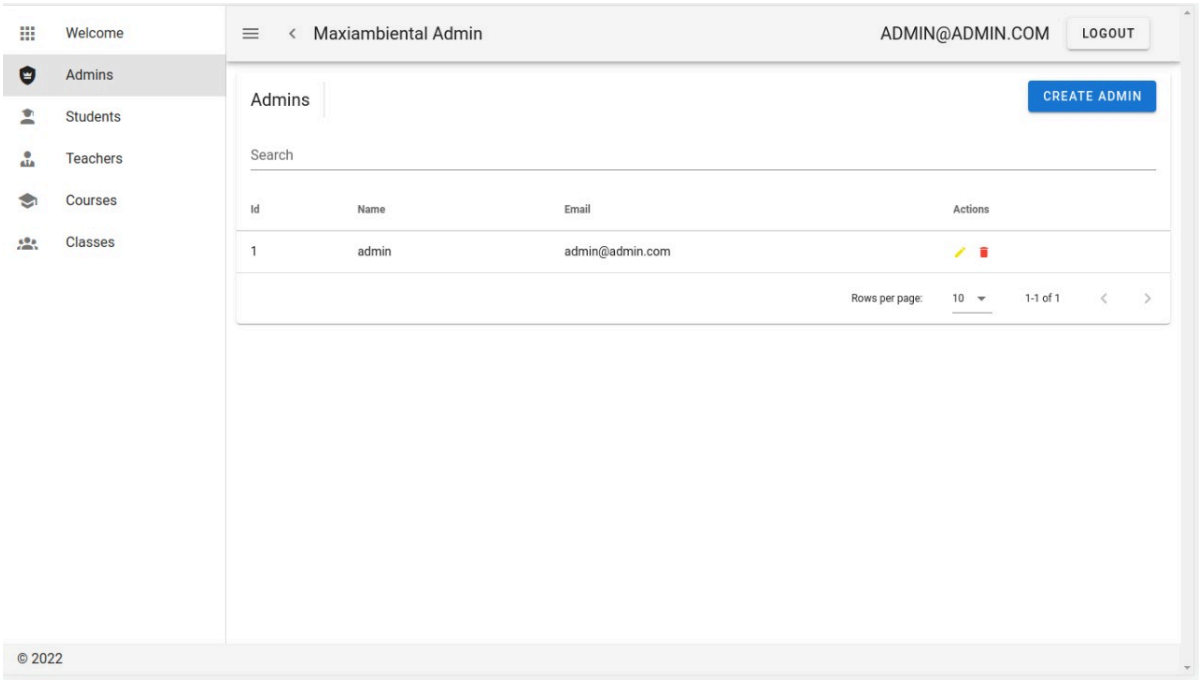

Figura 29 – Tela: listagem dos administradores.
Fonte: Autor (2022).

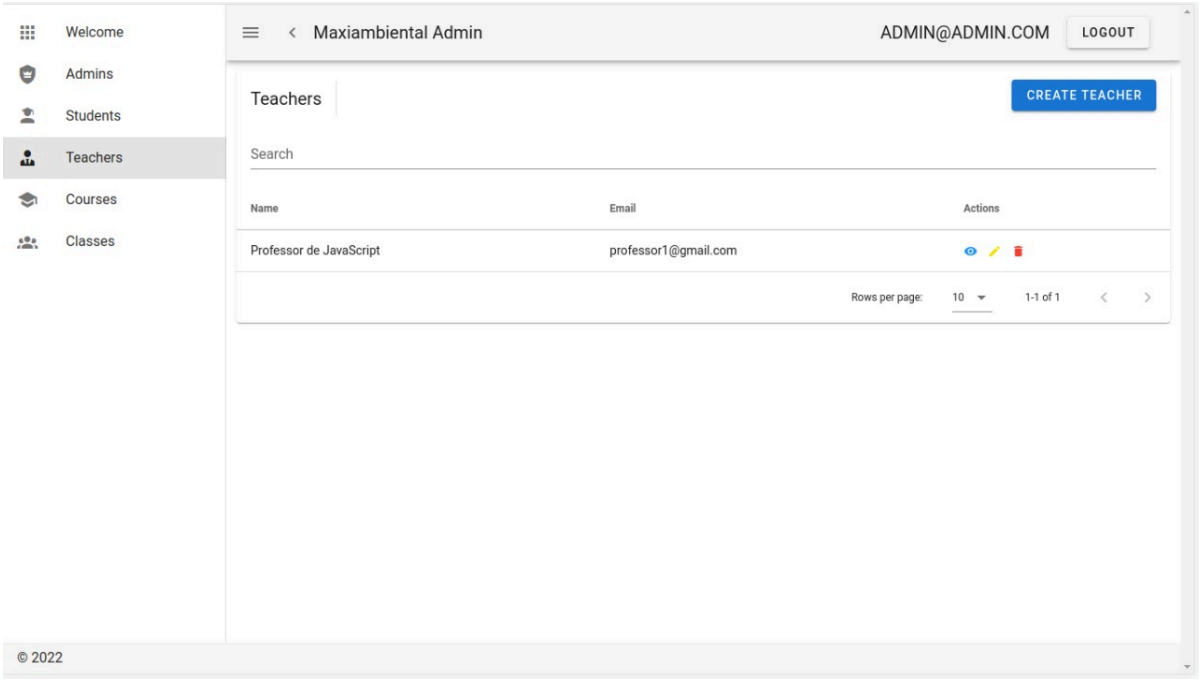

Figura 30 – Tela: listagem dos professores.
Fonte: Autor (2022).

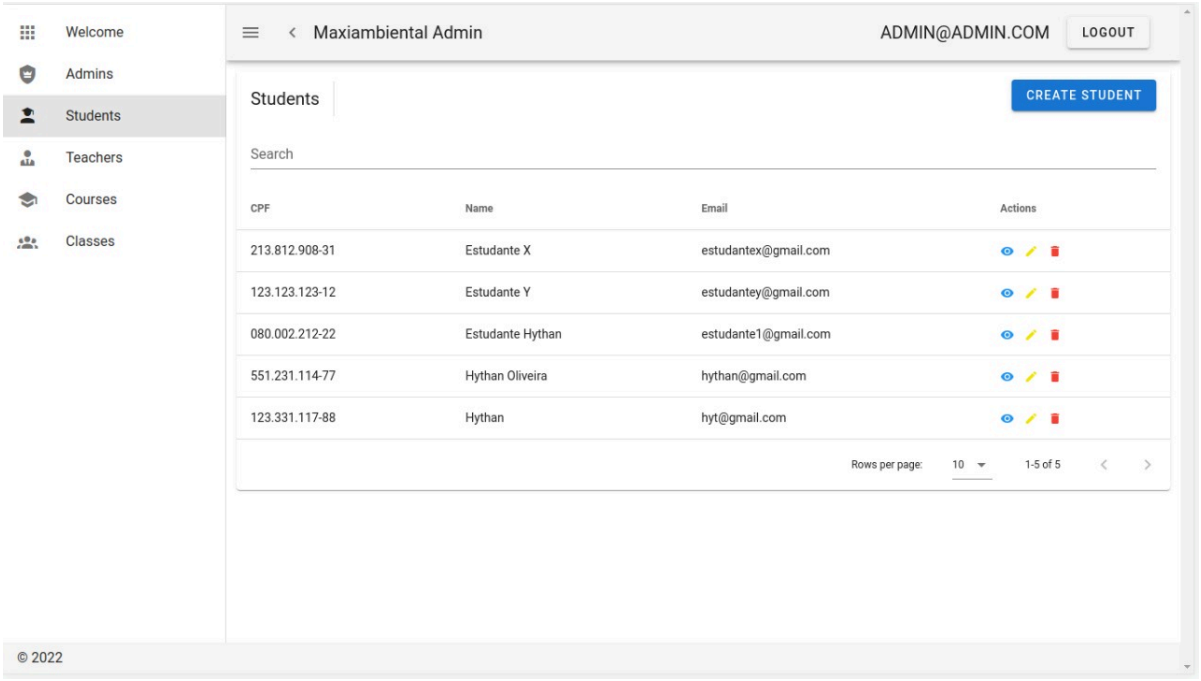

Figura 31 – Tela: listagem dos estudantes.
Fonte: Autor (2022).

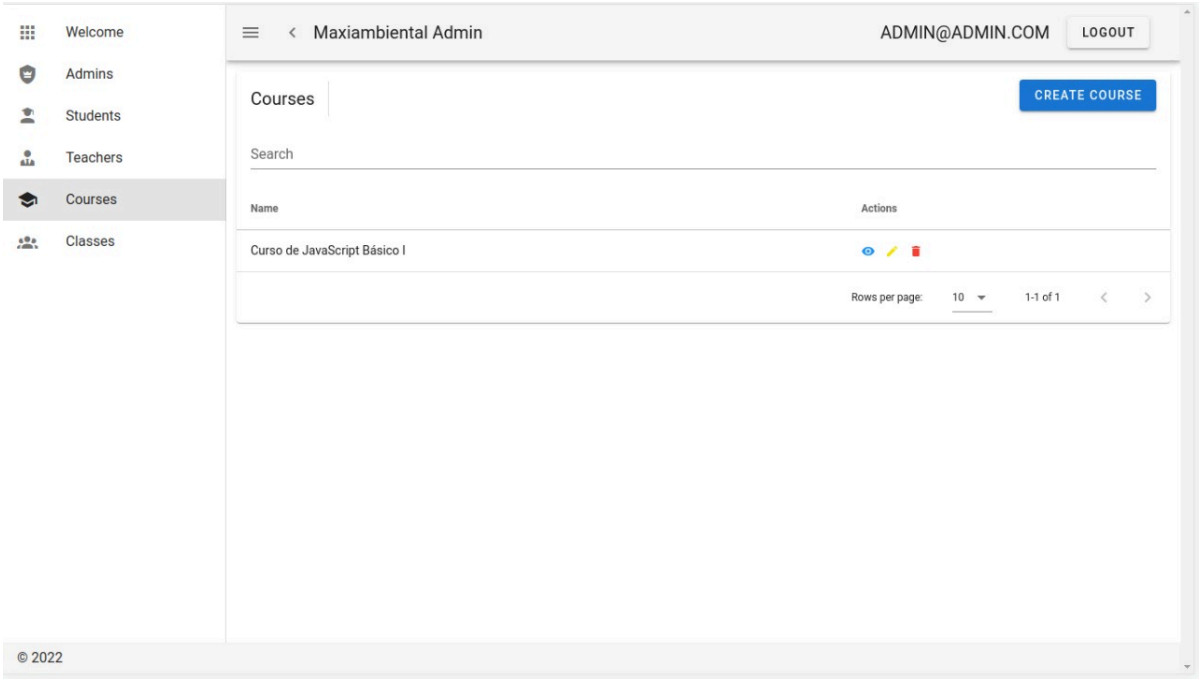

Figura 32 – Tela: listagem dos cursos.
Fonte: Autor (2022).

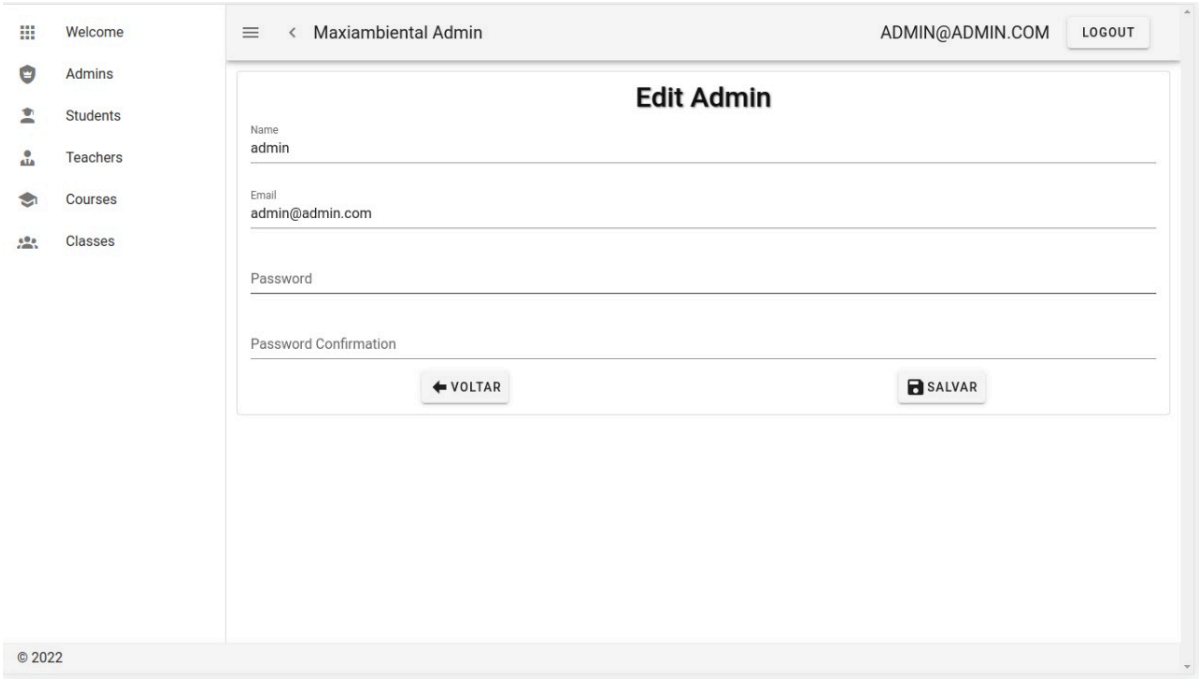

Figura 33 – Tela: editar administrador.
Fonte: Autor (2022).

Ainda na aplicação administrativa, foi criada as páginas e lógicas para gerenciamento das turmas, inscrições e geração de certificados. Dentro das telas de turmas, além do funcionamento tradicional de criação, exclusão, edição e visualização, foram adicionados o

funcionamento do gerenciamento das inscrições de uma determinada turma, apresentado na Figura 34, bem como para a geração de certificados, mostrado na Figura 35.

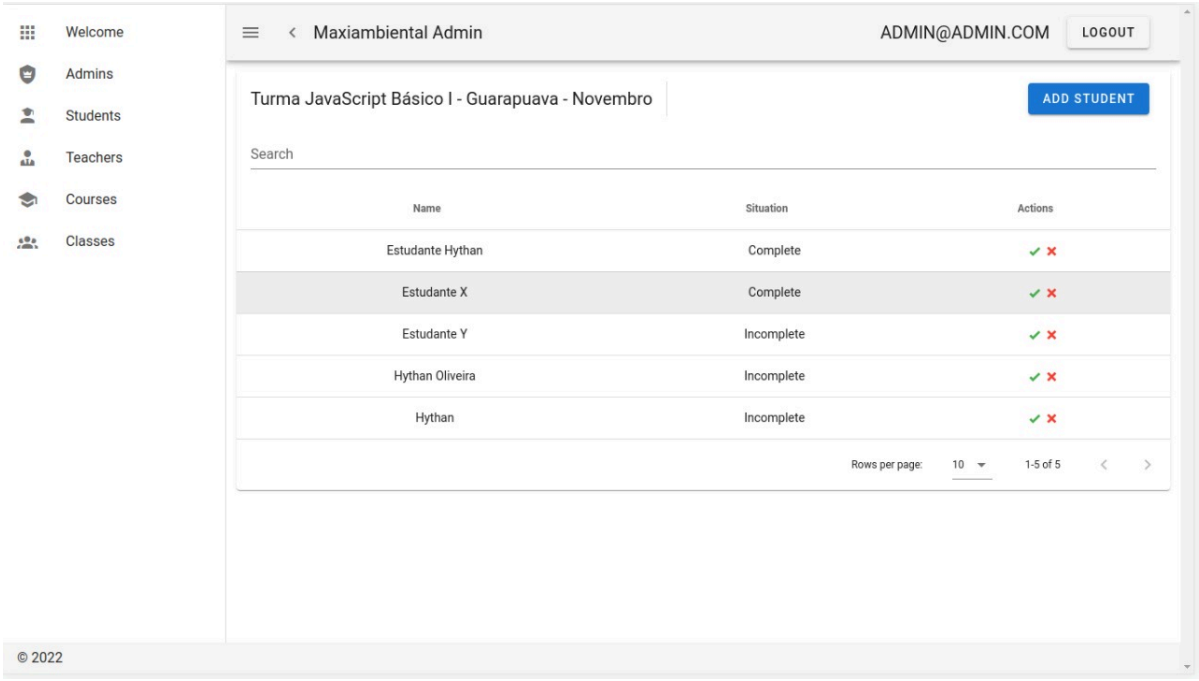

Figura 34 – Tela: alunos inscritos na turma.
Fonte: O Autor (2022).

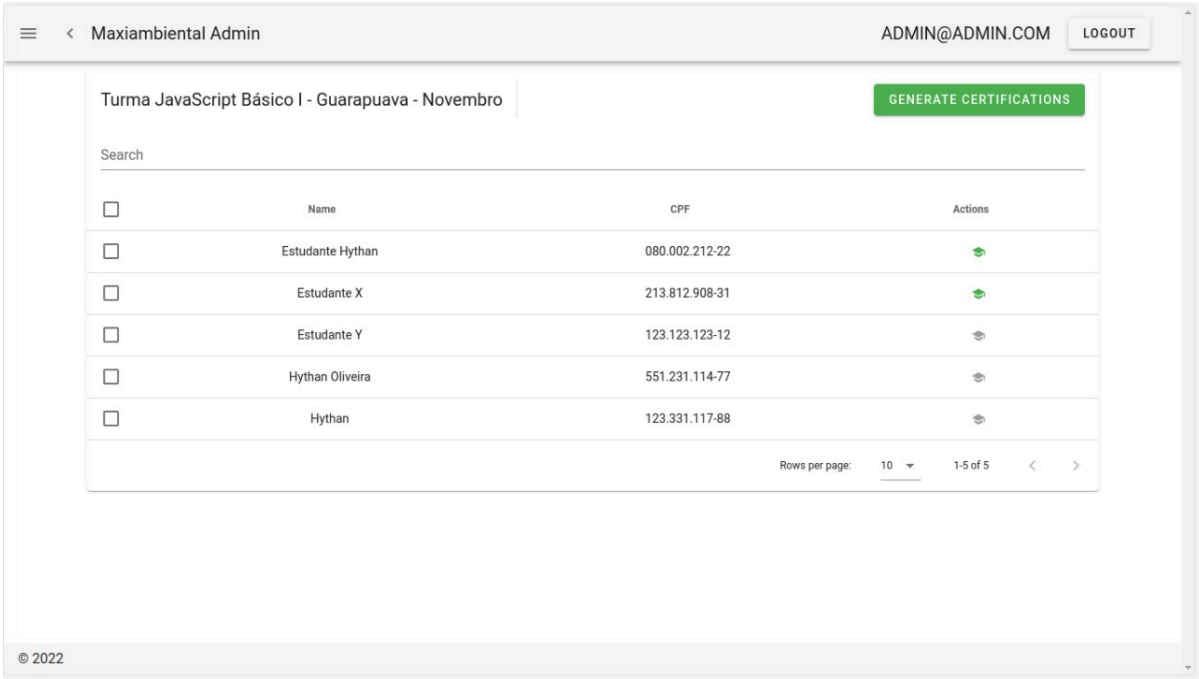

Figura 35 – Tela: alunos com certificados de uma determinada turma.
Fonte: Autor (2022).

Com a aplicação administrativa finalizada, foi iniciado o desenvolvimento da aplicação pública. Assim como a aplicação administrativa, durante o desenvolvimento desta, foi priorizado o funcionamento da arquitetura, deixando um pouco de lado questões de aparência. Pensando nisso, uma página inicial foi implementada, mas destaca-se as páginas de listagem de cursos, turmas abertas e certificados.

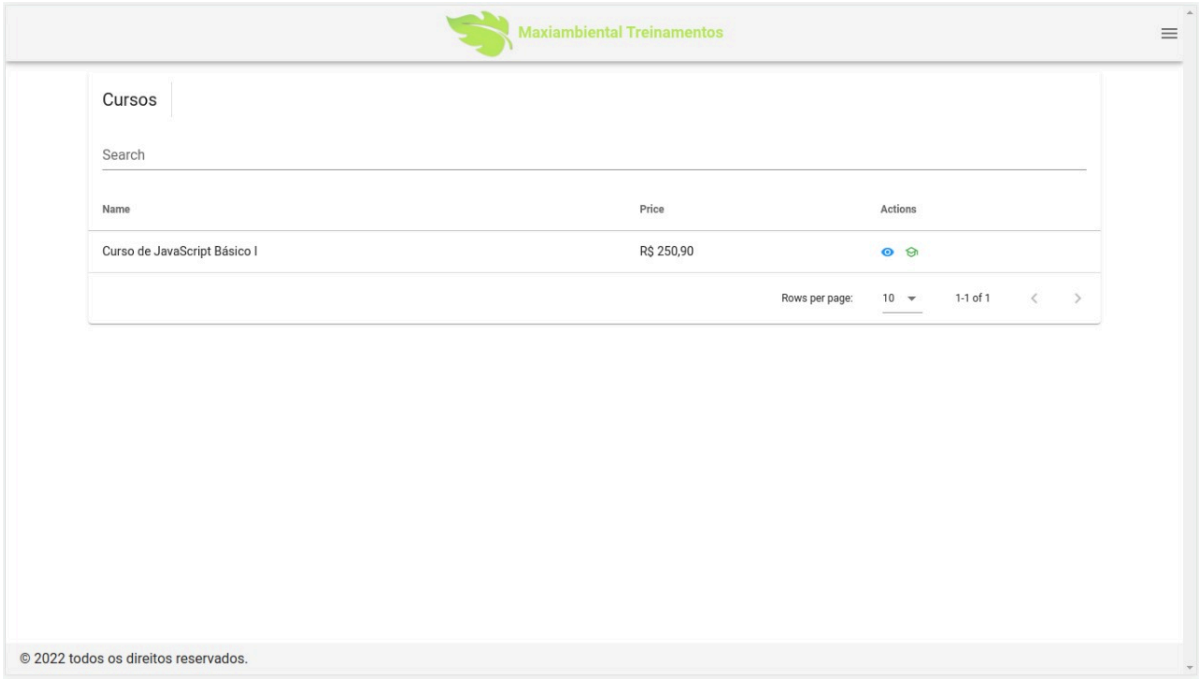

Figura 36 – Tela: listagem de cursos.
Fonte: Autor (2022).

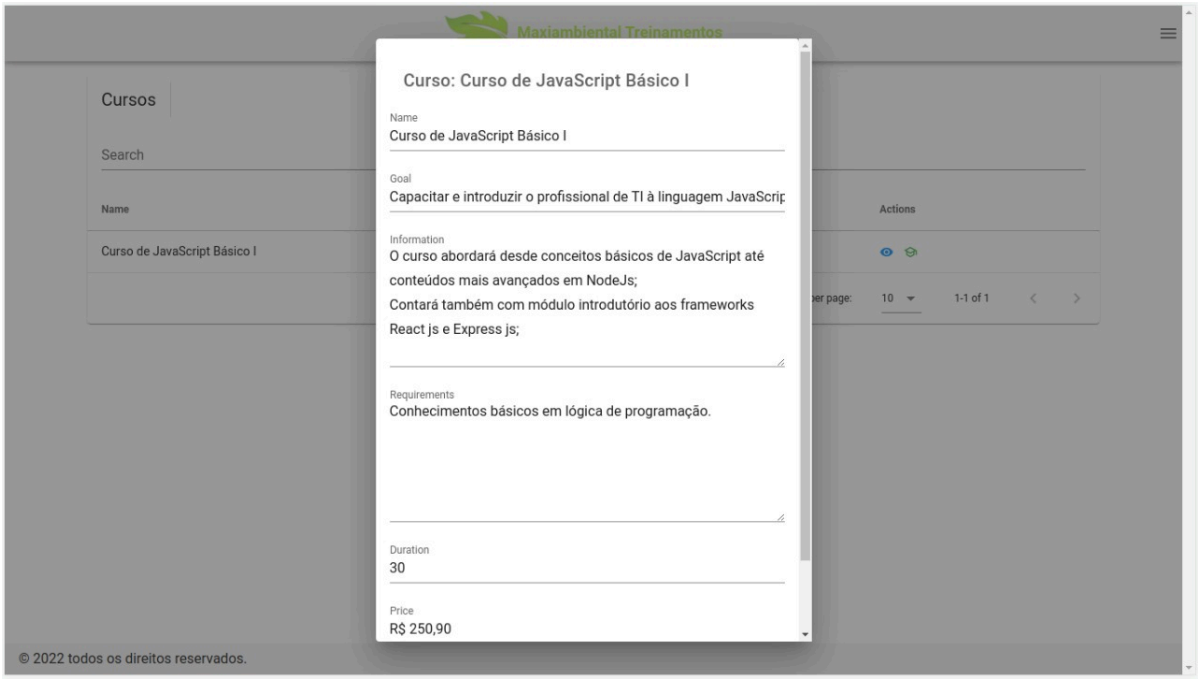

Figura 37 – Modal: informações sobre o curso.
Fonte: Autor (2022).

As Figuras 36 e 37 demonstram as telas inicias para o fluxo da aplicação, apresentando de uma forma objetiva os cursos disponíveis e contando com um botão para visualizar as turmas abertas referente aquele curso.

Nas Figuras 38 e 39, são demonstrados a listagens das turmas abertas, bem como a opção de visualizar as informações de uma determinada turma. Tais telas contam também com um botão responsável por fazer a inscrição do usuário.

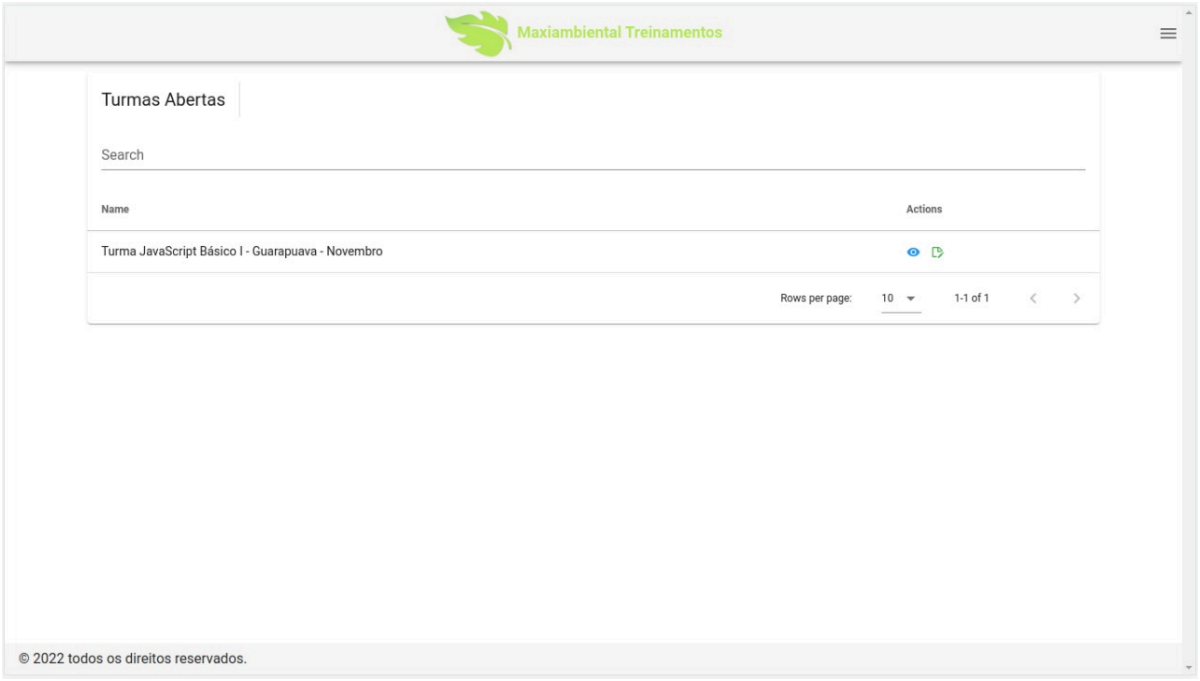

Figura 38 – Tela: listagem de turmas abertas de terminado curso.
Fonte: Autor (2022).

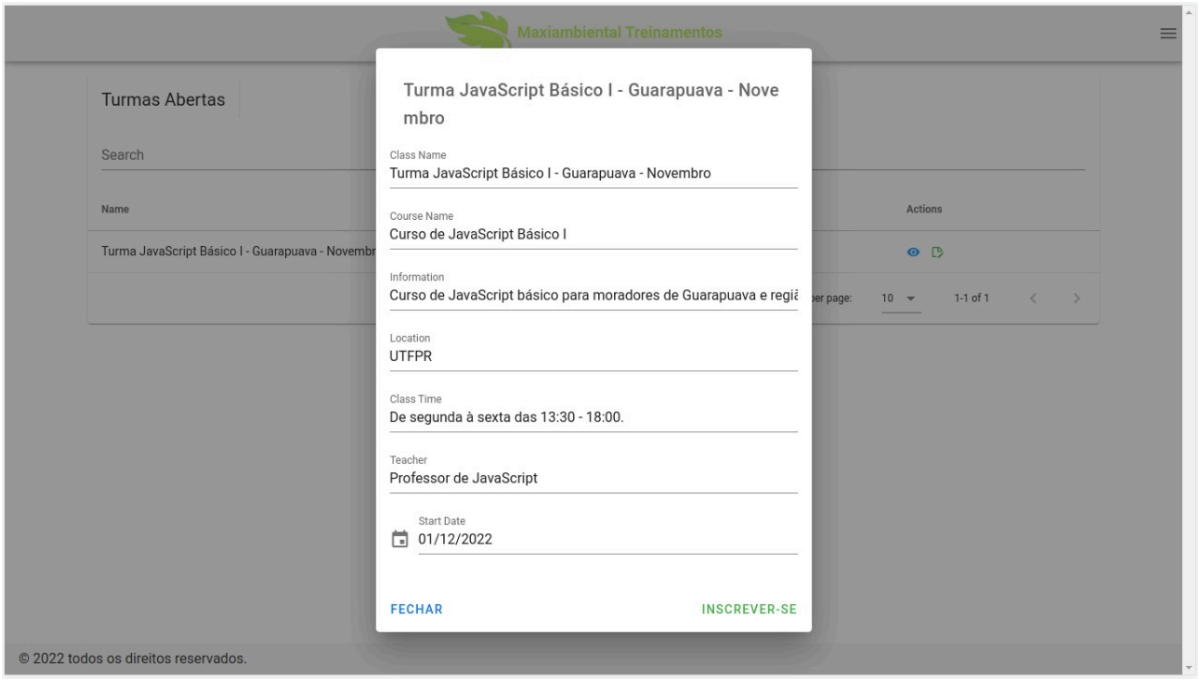

Figura 39 – Modal: informações sobre a turma.
Fonte: Autor (2022).

Além disso, uma página de *login* e cadastro foram implementadas, pensando em agilizar e facilitar a inscrição nos cursos de seu interesse, uma vez que com dados cadastrados, o usuário não precisará repetir o envio de seus dados em cada curso que pretenda fazer.

Por fim, foi implementado a tela responsável por disponibilizar aos usuários o acesso aos seus certificados, necessitando apenas que seja realizada a busca pelo CPF do aluno, conforme mostra a Figura 40, e possibilite que o mesmo realize o *download* do certificado que deseja, mostrado na Figura 41.

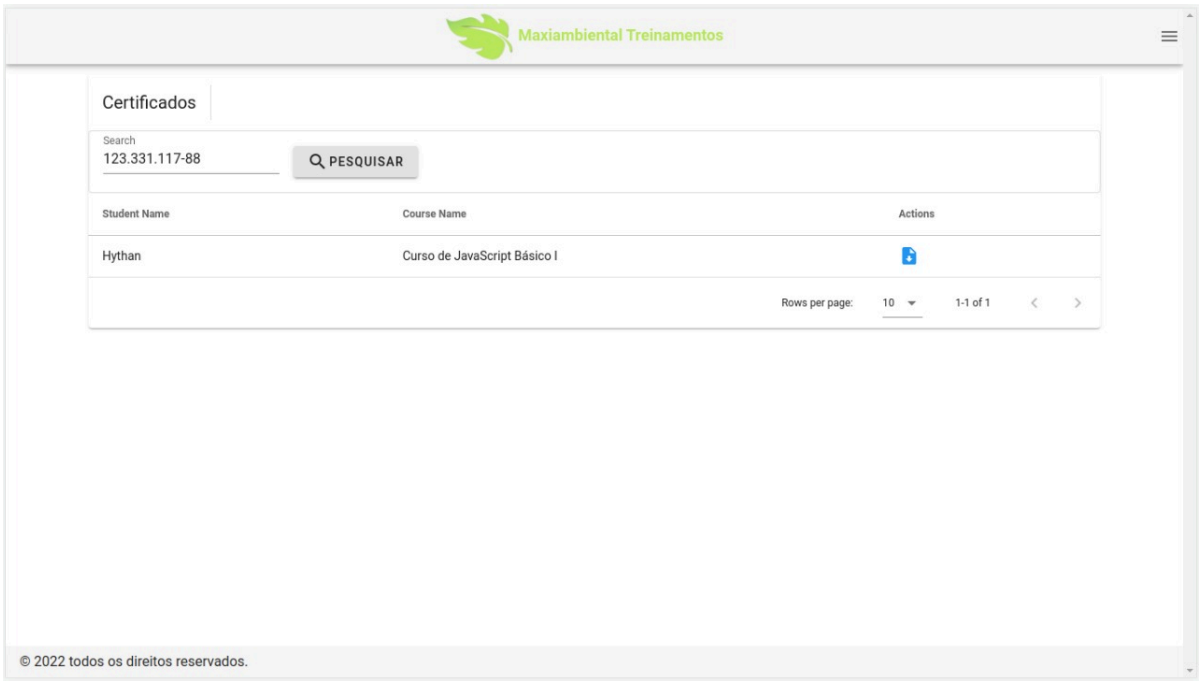

Figura 40 – Tela: listagem de certificados.
Fonte: Autor (2022).

Figura 41 – *Download* certificado.
Fonte: Autor (2022).

5.5.3 TESTES AUTOMATIZADOS

Durante o desenvolvimento de cada funcionalidade foi implementado um teste correspondente para garantir que o código estava apresentando o comportamento esperado. Dessa forma, evita-se que a aplicação não pare de funcionar por algum erro não tratado, uma vez que esta sendo testado os cenários possíveis. Entretanto, não foram feitos os testes de todas as partes do sistema, apenas dos microsserviços (aplicação de cursos e certificados). A Figura 42 apresenta um exemplo de testes feitos nos *services*, classes responsáveis pela lógica de salvar, editar, recuperar e excluir os dados, de uma determinada parte do sistema.

```
import { Test } from '@nestjs/testing';
import { AdminsModule } from 'src/admins/admins.module';
import { AdminsService } from 'src/admins/admins.service';
import { PrismaService } from 'src/prisma/prisma.service';

describe('AdminService Int', () => {
  let prisma: PrismaService;
  let adminService: AdminsService;
  beforeAll(async () => {
    const moduleRef = await Test.createTestingModule({
      imports: [AdminsModule],
    }).compile();
    prisma = moduleRef.get(PrismaService);
    adminService = moduleRef.get(AdminsService);
    await prisma.cleanDatabase();
  });

  describe('admin CRUD', () => {
    it('should create admin', async () => {
      const admin = await adminService.create({
        name: 'John',
        email: 'john@gmail.com',
        password: 'abc123',
      });
      const admins = await adminService.all();
      expect(admin.email).toBe('john@gmail.com');
      expect(admins.length).toBe(1);
    });

    it('should not create duplicated email admin', async () => {
      const response: any = await adminService.create({
        name: 'John 2',
        email: 'john@gmail.com',
        password: 'abc123',
      });

      expect(response.error).toBe('This email already been registred.');
      const admins = await adminService.all();
      expect(admins.length).toBe(1);
    });

    it('should update admin', async () => {
      const admin = await adminService.create({
        name: 'John',
        email: 'john1@gmail.com',
        password: 'abc123',
      });

      const adminUpdated = await adminService.update(admin.id, {
        name: 'Pedro',
      });

      expect(adminUpdated.name).toBe('Pedro');
    });
```

Figura 42 – Código responsável pelos testes do Admin.
Fonte: Autor (2022).

Todos os testes implementados foram utilizando a ferramenta JestJS, pela sua simplicidade e facilidade, além do fato de ser a ferramenta padrão para testes que já esta incluída no

framework NestJS.

Conforme citado anteriormente, os testes garantem que a aplicação não apresente erros inesperados, porém eles também possíbilitam que se melhore a implementação dos códigos. Um exemplo desse cenário foi observado durante o desenvolvimento dos testes para o *service* responsável pelas incrições, onde estava sendo feito a implementação de métodos desnecessários para salvar o dado, uma vez que, o próprio *framework* possuía uma tipagem para os dados, o que evitou todo o processo de tratamento.

Apesar de não ter sido feito os testes de todas as aplicações do sistema, acredita-se que com a cobertura de testes implementadas, se pode garantir um bom funcionamento do protótipo. A Fígura 43 apresenta a execução de todos os testes da aplicação de cursos e certificados, onde observa-se que foram feitos 52 testes no total.

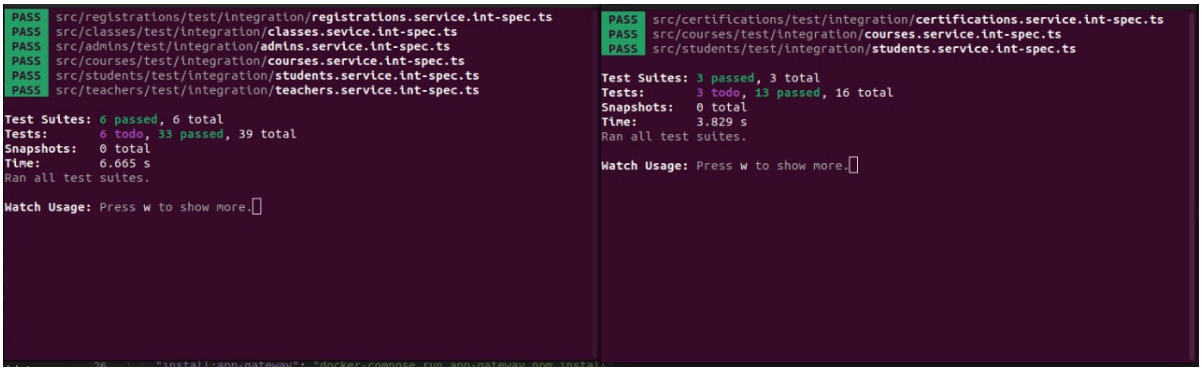

Figura 43 – Todos testes executados com sucesso.
Fonte: Autor (2022).

6 CONSIDERAÇÕES FINAIS

O desenvolvimento de um trabalho exige muita informação, dedicação e aprimoramento, e à medida que o aprofundamento num projeto acontece, é possível perceber a demanda que o mesmo exige.

Para a realização desde projeto procurou-se trabalhar com objetivo de se aperfeiçoar e aprender constantemente, para que se obtivesse êxito na implementação do protótipo, utilizando tecnologias e uma arquitetura na qual não se possuia conhecimento.

Durante o desenvolvimento, dificuldades foram encontradas desde o início. A falta de conhecimento em Docker, por exemplo, necessitou estudos e testes para que se chegasse as configurações dos ambientes utilizados. Além disso, nas tecnologias utilizadas, também não se possuia conhecimento, e necessitou de estudos, testes, escritas e reescritas de códigos para que se chegasse ao resultado obtido.

A arquitetura de microsserviços, algo complexo de ser entendido e implementado, também foi fator que dificultou a implementação de algo mais concreto e completo, porém, devido a essa complexidade, acredita-se que o objetivo principal, de conseguir realizar os estudos necessários para a implementação de um protótipo de sistema utilizando a arquitetura de microsserviços, foi atingido.

Por fim, vale resaltar que a experiência como desenvolvedor de *software* serviu de auxilio para verificar a grandiosidade de um sistema completo. Sendo assim nota-se que o conhecimento adquirido ao longo do percurso e o trabalho em equipe qualificada faz-se necessário e é ferramenta importante para desenvolver qualquer projeto a curto prazo.

Referências

ATLASSIAN. **Sobre o Trello**. 2022. Disponível em: <https://trello.com/about>. Acesso em: 18 de Dezembro de 2022. Citado na página 16.

CHOPIN, A.; CHOPIN, S. **NuxtJS**. 2022. Disponível em: <https://nuxtjs.org/>. Acesso em: 19 de Novembro de 2022. Citado na página 13.

DOCKER, I. **Docker**. 2013. Disponível em: <https://www.docker.com/>. Acesso em: 29 de Outubro de 2021. Citado 2 vezes nas páginas 10 e 11.

FACEBOOK. **JestJS**. 2022. Disponível em: <https://jestjs.io/pt-BR/>. Acesso em: 18 de Dezembro de 2022. Citado na página 17.

FACEBOOK, O. S. **React**. 2013. Disponível em: <https://pt-br.reactjs.org/>. Citado na página 12.

GOMES, R. **Docker para desenvolvedores**. [S.l.]: Leanpub, 2017. Citado na página 10.

GOOGLE. **Google Trends**. 2021. Disponível em: <https://trends.google.com.br/>. Citado na página 10.

.INC, V. **RabbitMQ**. 2022. Disponível em: <https://www.rabbitmq.com/>. Acesso em: 19 de Novembro de 2022. Citado 2 vezes nas páginas 14 e 15.

LINS, G. d. S. Utilizando reactjs para o desenvolvimento de um sistema de: alocação e reserva de salas no campus da ufc em quixadá. 2019. Citado na página 12.

MARIOTTI, F. S. Kanban: o ágil adaptativo: Introduzindo kanban na equipe ágil. Revista: Engenharia de Software Magazine., 2008. Disponível em: <http://www.garcia.pro.br/EngenhariadeSW/artigosMA/A6%20-%2045-6-%20Kanbam.pdf>. Acesso em: 26 de novembro de 2021. Citado 2 vezes nas páginas 13 e 14.

MILANI, A. **PostgreSQL-Guia do Programador**. [S.l.]: Novatec Editora, 2008. Citado na página 14.

MOZILLA, R. d. D. **JavaScript**. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 30 de Novembro de 2021. Citado na página 11.

MYSLIWIEC, K. **NestJS**. 2022. Disponível em: <https://docs.nestjs.com/>. Acesso em: 19 de Novembro de 2022. Citado na página 13.

OASIS. **About AMQP**. 2022. Disponível em: <https://www.amqp.org/about/what>. Acesso em: 18 de Dezembro de 2022. Citado na página 20.

OPENJS. **Node.js**. 2021. Disponível em: <https://nodejs.org/en/>. Acesso em: 29 de Outubro de 2021. Citado na página 11.

OPENJS, F. **Express**. 2010. Disponível em: <https://expressjs.com/>. Citado na página 12.

PEREIRA, C. R. **Construindo APIs REST com Node.js**. [S.l.]: Casa do Código, 2016. Citado 2 vezes nas páginas 11 e 12.

POSTGRESQL, G. d. D. G. **Postgres**. 1996. Disponível em: <https://www.postgresql.org/about/>. Citado na página 14.

RAMOS, E. C. Educação ambiental: origem e perspectivas. **Educar em Revista**, SciELO Brasil, p. 201–218, 2001. Citado na página 1.

RODRIGUES, A. B. **Uma abordagem gradativa de modernização de software monolítico e em camadas para SOA**. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2017. Citado 2 vezes nas páginas 9 e 10.

ROSA, A. A. Scrum e kanban para o gerenciamento de comunicacao em equipes de desenvolvimento de software. 2014. Citado na página 13.

SILVA, A. V. Desenvolvimento de aplicações com base na arquitetura de micro serviços. 2016. Disponível em: <https://cepein.femanet.com.br/BDigital/arqTccs/1411420621.pdf>. Acesso em: 19 de novembro de 2021. Citado na página 9.

TEIXEIRA, F. **Introdução e boas práticas em UX Design**. [S.l.]: Editora Casa do Código, 2014. Citado na página 8.

YOU, E. **VueJS**. 2022. Disponível em: <https://vuejs.org/>. Acesso em: 19 de Novembro de 2022. Citado na página 12.