doi.org/10.51891/rease.v8i12.7986

CRIANDO SISTEMA ESCALÁVEL DE AGENDAMENTOS UTILIZANDO TYPESCRIPT COM NESTJS NO BACKEND E NEXTJS NO FRONTEND

CREATING A SCALABLE SCHEDULING SYSTEM USING TYPESCRIPT WITH NESTJS ON THE BACKEND AND NEXTJS ON THE FRONTEND

Francisco Moreira Calado Souza[1]
Edilson Carlos Silva Lima[2]
Elda Regina de Sena Caridade[3]

RESUMO: Este trabalho busca mostrar o desenvolvimento de um sistema voltado para agendamentos de espaços dentro de uma instituição. Para isso, traz uma pequena amostra do funcionamento de algumas tecnologias utilizadas no desenvolvimento. O sistema objetiva implementar um controle dos espaços disponíveis dentro da instituição para se ter um controle de cada um. As etapas do desenvolvimento do sistema e os passos da implementação foram apresentadas no decorrer do trabalho, que tem como produto um sistema de agendamento de espaços. No desenvolvimento do sistema foi utilizado o padrão de arquitetura de software MVC responsável por contribuir na otimização da velocidade entre as requisições feitas pelo comando dos usuários, frameworks como Nest.JS e NextJS que são frameworks Node responsável pelo desenvolvimento do BackEnd e FrontEnd, também foi utilizado a programação funcional para construção de softwares através de composição de funções puras e a orientação a objetos.

Palavras-chave: MVC. POO Programação Funcional. TypeScript. NestJS. Next. JS.

ABSTRACT: This work seeks to show the development of a system aimed at scheduling spaces within an institution. For this, it brings a small sample of the operation of some technologies used in the development. The system aims to implement a control of the available spaces within the institution to have a control of each one. The system development stages and implementation steps were presented during the work, which has a space scheduling system as a product. In the development of the system, the MVC software architecture pattern was used, responsible for contributing to the optimization of the speed between requests made by the users' command, frameworks such as Nest.JS and NextJS, which are Node frameworks responsible for the development of the BackEnd and FrontEnd, as well functional programming was used to build software through the composition of pure functions and object orientation.

Keywords: MVC. POO. Functional Programming. TypeScript. NestJS. Next. JS.

[1]Engenharia da Computação – Universidade Ceuma (CEUMA) – São Luís– MA – Brasil.
[2]Engenharia da Computação – Universidade Ceuma (CEUMA) – São Luís– MA – Brasil
[3]Engenharia da Computação – Universidade Ceuma (CEUMA) – São Luís– MA – Brasil

1. INTRODUÇÃO

Nas instituições educacionais, os espaços como laboratórios e auditórios, utilizados por educadores são de extrema importância e, bastante valorizados para a educação. São espaços vistos como tendo um papel fundamental para o avanço no desenvolvimento dos alunos. É necessário que se mantenha uma organização quanto ao uso desses espaços para que seja possível manter um controle sobre cada um deles.

Esse trabalho destina-se a descrever, testar e implementar aplicações através de Api desenvolvida com NestJS, um framework Express construído com os princípios do MVC, Orientação a Objetos (POO) e Programação Funcional (PF) em TypeScript. E também da interface gráfica, desenvolvido com Next.Js, um framework React que busca melhorar a experiência dos desenvolvedores através de blocos de funcionalidades essenciais para a criação de aplicações web.

Visando explorar a possibilidade de manter uma organização na utilização desses espaços educacionais, foi pensando no desenvolvimento de um sistema, que permita o agendamento de espaços disponíveis para uso e assim se tenha uma organização dentro da instituição.

Foi utilizado o método de pesquisa descritiva com a finalidade de analisar a forma de utilização dos espaços e os benefícios que poderiam ter com a utilização do sistema proposto. Para isso, a pesquisa foi baseada no cotidiano dos alunos e profissionais da instituição e, foi necessário também uma pesquisa documental e entrevistas com esses alunos e profissionais.

2. Fundamentação Teórica

Nesta seção, os conceitos/técnicas que serão utilizados para realizar este artigo, e os sistemas associados analisados, serão apresentados para gerar ideias para o protótipo desenvolvido.

Deve-se notar que este trabalho abrange diferentes domínios computacionais. Conceito/Tecnologia utilizados foram divididos, conforme demonstrado abaixo. E a tecnologia em comum utilizada tanto para o BackEnd como para o FrontEnd é o TypeScript. Algumas das funcionalidades que o TypeScript introduz ao JavaScript além da tipagem em si, são os Decorators. Decorators no TypeScript são formas de injetar lógica dentro de classes, métodos, accessors, propriedades ou parâmetros. Estes

decorators são considerados um dos pontos mais positivos do Framework NestJS, utilizado no Backend desta aplicação.

2.1. Back-End

O Back-End refere-se a parte de um aplicativo ou código de um programa que permite que ele funcione e que não pode ser acessado por um usuário. "Desenvolvedores BackEnd trabalham em uma camada que é abstraída do usuário" (MARQUEZ-SOTO, 2022). A maioria dos dados e da sintaxe operacional é armazenada e acessada no back-end de um sistema. Normalmente, o código é composto por uma ou mais linguagens de programação. O back-end também é chamado de camada de acesso a dados de software ou hardware e inclui qualquer funcionalidade que precise ser acessada e navegada por meios digitais.

2.2. Front-End

A camada acima do back-end é o front-end e inclui todo o software ou hardware que faz parte de uma interface de usuário. Os usuários humanos ou digitais interagem diretamente com vários aspectos do front-end de um programa, incluindo dados inseridos pelo usuário, botões, programas, sites e outros recursos. A maioria desses recursos é projetada por profissionais de Experiência do Usuário (UX) para serem acessíveis, agradáveis e fáceis de usar. Além disso, para Kinsbruner (2022), um desenvolvedor FrontEnd é responsável por toda criação de elementos UI que sejam compatíveis com todas as plataformas web e mobile.

2.3 POO

Programação orientada a objetos é um dos paradigmas mais populares e mais utilizados em todo o mundo, é principalmente ensinado nas universidades como a maneira de programar com linguagens como Java. POO está diretamente ligada a classes e objetos, é utilizada para estruturar o código em pedaços reutilizáveis (classes), geralmente utilizadas para criar instâncias únicas de objetos. Na visão de CRAIG (2007), "A Programação Orientada a Objetos abriu muitas perspectivas no conceito de software e foi aclamada como parte da solução da então chamada *crise de software*".

2.4 MVC

O modelo no padrão MVC representa as partes do aplicativo que implementam a lógica do domínio de dados. A operação do modelo pode vir da geração de classes que representam objetos em um armazenamento de dados como um banco de dados. 'Views' são os elementos visíveis no aplicativo. Eles são os componentes que normalmente mostram aos usuários os dados do modelo. Os controladores são classes que coletam a solicitação do usuário, trabalham com o modelo e, por fim, selecionam uma exibição para renderizar a interface do usuário apropriada. Uma de suas principais vantagens é mencionada por SARCAR (2022) como "separar a lógica da interface do usuário das regras de negócio e desacoplar componentes principais de forma que podem ser reutilizados de forma eficiente".

2.5 Programação Funcional

Com a Programação Funcional é possível desenvolver software com funções puras, evitando estados compartilhados, dados mutáveis e side-effects. Na PF, o estado flui através das funções puras, algo completamente diferente da POO em que o estado fica armazenado em propriedades e métodos inseridos em um objeto. A principal vantagem disso é que o código passa a ter um comportamento mais previsível, "Isso torna programas funcionais mais previsíveis do que em outras abordagens e pode tornar programas funcionais mais fáceis de funcionar em múltiplos processadores", MUELLER (2019).

2.6 REST API

REST, Representational State Transfer, foi a arquitetura utilizada para definir a API utilizada no FrontEnd. Como dito por MASSE, Mark (2011), "Se pudéssemos dizer que a internet possui um 'sistema operacional', seu estilo de arquitetura é REST". Essa arquitetura consiste basicamente em ter rotas que representam recursos no sistemas e que podem ser utilizadas por métodos como GET, POST, DELETE, PATCH para realizar ações no recurso ao qual a rota se refere.

2.7 TypeScript

TypeScript é uma linguagem de programação fortemente tipada, multi-paradigma, construída como um superset do JavaScript. TypeScript foi escolhida como

a linguagem a ser utilizada no Backend pelo fato de ser mais escalável que o JavaScript puro e também por ser a mais utilizadas em Frameworks FrontEnd, o que aumenta a produtividade de desenvolvedores FullStack, já que podem desenvolver uma aplicação completa utilizando somente uma linguagem de programação. Além disso o TypeScript possui uma quantidade enorme de bibliotecas que solucionam dos mais diversos problemas, e como mencionado por CHERNY, Boris, "Ao se aproveitar de ferramentas, bibliotecas e frameworks já existentes para desenvolver tanto o frontend quanto o backend, podemos avançar rapidamente e com uma base sólida ao construir nossa própria aplicação.".

2.8 NestJS

NestJS é um framework opinionated utilizado para o desenvolvimento de aplicações executadas no lado do servidor, pode ser utilizado para desenvolver Rest API, MicroServices e também GraphQL. O NestJS vem com padrões de código pré-definidos que devem ser seguidos na hora de desenvolver, além disso, conta com uma grande variedade de ferramentas para auxiliar no desenvolvimento de aplicações BackEnd complexas e escaláveis. Porém nem sempre desenvolvedores optam por essa abordagem, muitos optam por utilizar algo com menos boilerplate e que permite ter mais controle da aplicação, como mencionado no livro The TypeScript Workshop (2021), "Alguns desenvolvedores realmente gostam de trabalhar com um framework cheio de recursos e outros enxergam o boilerplate como opressor escolhem desenvolver com algo mais básico como Express".

3. Estudo de caso

O sistema foi desenvolvido com o intuito de realizar agendamentos de espaços físicos dentro da instituição educacional para o uso desses espaços em determinadas datas e horários, tendo como principal objetivo, manter a organização e o controle de disponibilidades desses espaços.

3.1. Desenvolvimento do Back-End

A construção da API foi realizada seguindo padrões disponíveis nas documentações de cada tecnologia e frameworks. Para chegar às funcionalidades finais do sistema, foi utilizado NestJS como principal recurso.

Para manter a escalabilidade e boa manutenção de código, NestJS se utiliza de princípios da POO (figura 1), Programação Funcional (figura 2) e padrão de projeto MVC (figura 3).

```
export class AuthModule implements NestModule {
  configure(consumer: MiddlewareConsumer) {
    consumer.apply(LoginValidationMiddleware).forRoutes('login');
  }
}
```

Figura 1: Herança de Classes no Nest.JS
Fonte: Autoral, 2022.

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.getHttpAdapter().getInstance().disable('x-powered-by');
  app.enableCors({
    origin: '*',
    methods: ['OPTIONS', 'GET', 'POST', 'PATCH', 'DELETE'],
    preflightContinue: false,
    optionsSuccessStatus: 204,
  });
  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
      whitelist: true,
      forbidNonWhitelisted: true,
    }),
  );

  await app.listen(process.env.PORT || 3000);
}
bootstrap();
```

Figura 2: Programação Funcional no Nest.JS
Fonte: Autoral, 2022.

```
Calado, 6 months ago | 1 author (Calado)
@Module({
  imports: [UsersModule, SpacesModule, RequestsModule, AuthModule],
  controllers: [AppController],
  providers: [AppService, JwtAuthProvider, RolesProvider],
})
export class AppModule {}
```

Figura 3: MVC no Nest.JS
Fonte: Autoral, 2022.

O Nest.JS apresenta uma grande facilidade de integrar e dar suporte a bibliotecas e recursos populares já existentes e isso é um divisor de águas no desenvolvimento de REST APIs. Um exemplo é o suporte oficial à biblioteca do Prisma. Prisma vai muito

além de um ORM responsável pela interação com banco de dados, garantindo type-safety em todas suas operações.

Nesta aplicação o Prisma foi utilizado para modelar o banco de dados, criar e aplicar migrations automaticamente, visualizar o banco de dados de maneira mais user-friendly, mas a principal funcionalidade que torna Prisma uma biblioteca única é sem dúvidas o sistema de query build, em que se pode realizar consultas complexas no banco de dados sem precisar saber ou escrever SQL, utilizando seu formato de construção de query utilizando objetos com tipos gerados baseados no schema. Na figura 4 pode-se observar como a biblioteca do Prisma integra facilmente com o NestJS ao implementar a interface OnModuleInit, que sinaliza ao Prisma, no momento da inicialização do módulo, que pode iniciar a conexão com o banco de dados.

```typescript
import { INestApplication, Injectable, OnModuleInit } from '@nestjs/common';
import { PrismaClient } from '@prisma/client';

@Injectable()
export class PrismaService extends PrismaClient implements OnModuleInit {
  async onModuleInit() {
    await this.$connect();
  }

  async enableShutdownHooks(app: INestApplication) {
    this.$on('beforeExit', async () => {
      await app.close();
    });
  }
}
```

Figura 4: Conexão do Prisma com o Nest.Js
Fonte: Autoral, 2022.

Além disso, NestJS fornece suporte oficial a bibliotecas de autenticação como PassportJS, através de guards. Guards consistem em classes de responsabilidade única que determinam se a requisição vinda do Client deve ou não ser repassada para o Router dependendo de fatores definidos pelo código.

Figura 5: Diagrama do fluxo dos guards na aplicação.
Fonte: NestJS, 2022.

Para fazer o login inicial com username e senha, é necessário utilizar um guard de autenticação de estratégia local fornecido pela biblioteca Passport, um middleware de autenticação para NodeJS, e @nestjs/passport, que contém utilitários para facilitar a integração com NestJS. Após as credenciais serem validadas, o passport injeta um objeto *user* dentro do objeto da requisição, para facilitar a identificação do usuário em diferentes rotas (Figura 6).

```
import {
  ExecutionContext,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class LocalAuthGuard extends AuthGuard('local') {
  canActivate(context: ExecutionContext) {
    return super.canActivate(context);
  }

  handleRequest(err, user) {
    if (err || !user) {
      throw new UnauthorizedException(err?.message);
    }

    return user;
  }
```

Figura 6: Implementação do Guard de a.
Fonte: Autoral, 2022.

Depois de validado, o usuário recebe um JWT, JSON Web Token, que contém suas informações e pode ser utilizado como uma espécie de passaporte para rotas que são protegidas pelo guard do JWT. Um JWT consistem em basicamente um objeto

codificado em Base64 dividido em 3 partes, o Header que contém o tipo de algoritmo utilizado para assinar o token, o Payload que contém os dados referentes ao usuário, e uma assinatura que é o que garante a autenticidade de um JWT.

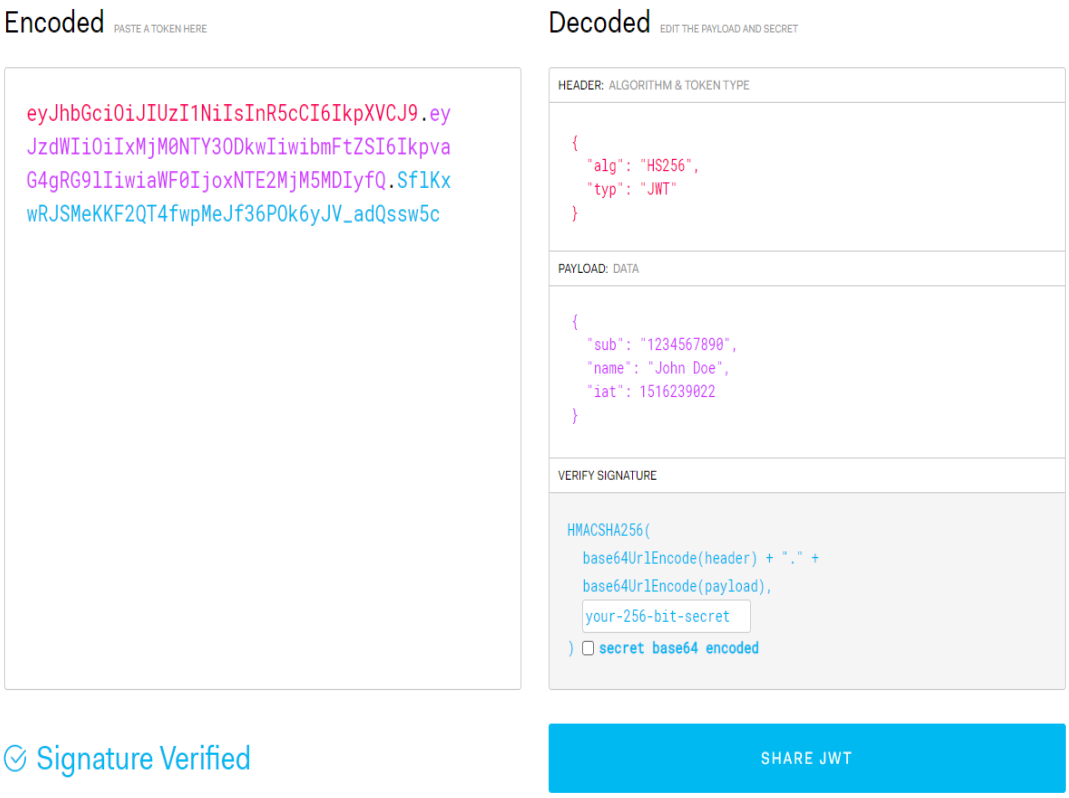

Figura 7: Ilustração da estrutura de um JWT.
Fonte: JWT.IO, 2022.

3.2. Desenvolvimento do Front-End

O FrontEnd também foi criado seguindo padrões e documentações disponíveis pelas tecnologias necessárias. Para desenvolver o FrontEnd utilizou-se o Framework Next.Js, que facilita o desenvolvimento de aplicações React e dá escalabilidade e performance para aplicações de maneira automática devido a sua forma de utilizar server-side rendering, uma forma de renderizar HTML no lado do servidor.

Next.Js foi importante principalmente para a autenticação, segurança e configuração de rotas na aplicação. Sua função getServerSideProps foi bastante utilizada na renderização de páginas de dashboard que necessitavam de permissões especiais, já que antes de algo ser devolvido para o cliente poderia se verificar se ele poderia ou não estar acessando aquela rota.

```
export const getServerSideProps: GetServerSideProps<RequestsDashboard> = async (
  ctx
) => {
  const token = getToken(ctx)

  const reqs = await getManyRequests(`${token}`)
  if (!reqs) return sendToHomePage()

  return {
    props: { reqs },
  }
}
```

Figura 8: Implementação do Guard de a.
Fonte: Autoral, 2022.

Além disso, foram utilizados middlewares para proteger de maneira mais abrangente as rotas. Middlewares no Next.Js executam código antes da requisição ser concluída, então é possível fazer qualquer tipo de modificação e manipulação da maneira que você deseja lidar com os dados que recebe.

```
import { NextRequest, NextResponse } from 'next/server'
import { apiURL } from '../../services';

export async function middleware(req: NextRequest) {
  const token = req.cookies['token'];

  const headers = new Headers();
  headers.append('Content-Type', 'application/json');
  headers.append('Accept', 'application/json');
  headers.append('Authorization', `Bearer ${token}`);
  const options = {
    method: 'GET',
    headers,
  }

  const user = await fetch(`${apiURL}/me`, options).then(response => {
    return response.json()
  })
  if(user.role !== 'ADMIN') {
    return NextResponse.redirect(new URL('/', req.url));
  }

  return NextResponse.next()
}

```

Figura 9: Implementação do Guard de a.
Fonte: Autoral, 2022.

Uma das partes mais importantes, se não a mais importante no ponto de vista do usuário, é a estilização, a forma que o website aparece para o usuário. Como desejamos construir aplicações escaláveis, com boa manutenibilidade e performáticas, também se optou pela utilização do Framework CSS TailwindCSS, junto ao post processor PostCSS. TailwindCSS consiste basicamente em um conjunto de 'utility' classes que facilitam a estilização e dão controle total e 'pixel-perfect' do que é exibido na tela para o usuário.

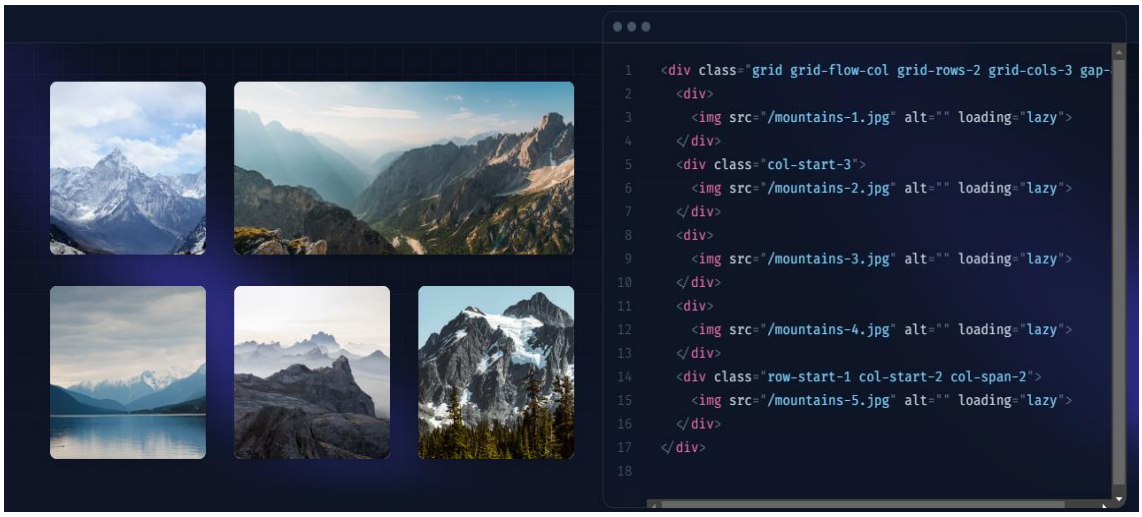

Figura 10: Estilização com TailwindCSS.
Fonte: TailwindLabs, 2022.

O motivo de ter sido escolhido vai além da DX, Developer Experience, TailwindCSS também se utiliza de técnicas que analisam o CSS escrito e automaticamente remove estilizações desnecessárias. Isso aliado com sua alta capacidade de reutilização o torna a melhor opção para desenvolver aplicações de alta performance e de grande escala sem se preocupar com o tamanho do bundle do CSS, já que uma classe é exportada somente uma vez para o cliente e reutilizada em todos os componentes necessários.

4. RESULTADOS E DISCUSSÃO

Depois de uma longa análise realizada com testes e entrevistas feitas com orientadores da instituição educacional, chegou-se à conclusão de que manter a organização dos espaços físicos disponíveis para uso dentro da instituição é algo indispensável e de extrema importância para se manter um controle sobre eles.

Através da metodologia de pesquisa descritiva utilizada para desenvolver a aplicação, alguns dados puderam ser coletados para conhecimento da instituição, dentre eles o público que mais costuma agendar horários em certos espaços, assim podendo também medir o desempenho de atividades práticas na instituição.

Em princípio, através do roteiro de perguntas da entrevista em questão, investigamos se antes já foi ao menos pensado entre eles, algo sobre a necessidade da organização dos espaços físicos dentro da instituição. Em sua maioria, as respostas foram que até um certo período não se pensava sobre tal assunto. Percebemos que a minoria que apresentou uma resposta contrária, já havia pensado sobre e, entende a importância de se manter essa organização, de modo que enfatizou que, é necessário valorizar os espaços físicos disponíveis da instituição e trabalhar para que seja possível que todos usem da forma correta e possa facilitar esse uso sem prejudicar os alunos que se beneficiam desses espaços.

Seria, pois, de extrema importância que, esse sistema seja implementado, para que todos possam utilizar de forma organizada os espaços realizando os agendamentos caso seja possível de acordo com a disponibilidade de cada um.

Com relação às opiniões dos orientadores e profissionais entrevistados sobre a importância da organização dos espaços, todos reconhecem essa relevância, considerando que influencia na forma como será feito a utilização deles no dia-a-dia.

Posteriormente, quando questionamos sobre como eram organizados os espaços e que critérios eram levados em conta para o uso deles, responderam que não se tinha uma organização e que eram usados caso um outro orientador não estivesse usando. Não tinha um controle e, que eles estavam livres para o primeiro que tivesse interesse em usá-lo. Esse aspecto se mostra de extrema relevância, considerando que o espaço auxilia no desenvolvimento do aluno, onde eles colocam em prática o que é visto na teoria e também tem contato com outras competências necessárias para cada um. Os orientadores enfatizaram a importância da organização do espaço como estratégia para se ter controle e direcionamento de cada espaço.

Por fim, como objetivamos no início da pesquisa, foi realizada uma observação do espaço em questão, para constatar que características possui e, como esses espaços são bem pensados para a prática educacional. A instituição possui salas de aulas, laboratórios, e auditórios para permitir aos alunos o contato com outras formas de

desenvolvimento. A instituição valoriza o desenvolvimento do aluno e promove ambientes adequados para explorar e desenvolver a aprendizagem de cada um.

CONCLUSÃO

Neste trabalho de conclusão de curso, um sistema para gerenciar os espaços disponibilizados por uma instituição de ensino. Gerenciamento dos espaços na instituição que é de extrema importância para se manter uma organização e controle. Diante de muitas pesquisas e investigações realizadas, percebemos que esses espaços não devem ser vistos apenas como um lugar ao qual os alunos vão estudar, mas um lugar onde eles devem aprender a desenvolver habilidades e adquirir conhecimentos sobre assuntos vistos em sala de aula e assuntos que serão importantes para o futuro de cada um.

Este projeto foi criado a partir de ideias e reuniões com profissionais da instituição beneficiada. O processo de criação do projeto teve algumas etapas como: pré-reunião, reunião e pós-reunião. Para automatizar o processo de projeto, foi feita uma modelagem (análise) para que fosse desenvolvido o sistema para gerenciamento dos espaços. Além da modelagem completa do sistema, foi desenvolvido um protótipo inicial para observação de todos os detalhes necessários. Durante o desenvolvimento da aplicação foi-se descobrindo problemas para desenvolver o módulo de autorização, uma vez que as funções de aprovar solicitações de agendamento poderiam ser delegadas para outros usuários, porém com acesso limitado. Este problema foi resolvido com a criação de níveis de autorização na tabela do usuário banco de dados.

Ressalta-se que o sistema já está em uso dentro da instituição e até o momento apresentando ótimos resultados.

A partir do que já foi desenvolvido, pretende-se tornar o sistema agnóstico à instituição, para que possa assim ser utilizado em todas as unidades. Além disso, há a expectativa de que possa ser utilizado também como fonte de divulgação de eventos que ocorrerão em todas as instituições, tornando-se assim o sistema aberto ao público e não somente aos funcionários da instituição.

REFERÊNCIAS

AUTHo. JWT.IO. Disponível em: https://jwt.io/. Acesso em: 6 nov. 2022.

CHERNY. Boris. Programming TypeScript: Making Your JavaScript Applications Scale. O'Reilly Media, 2019.

CRAIG. Iain. Object-Oriented Programming Languages: Interpretation (Undergraduate Topics in Computer Science). Springer, 2007.

HUDGENS, Jordan., STEFANOVSKI, Wekoslav., GRYNHAUS, Ben., MORGAN, Matt., HUNTE, Rayon. The TypeScript Workshop: A Practical Guide to Confident, Effective TypeScript Programming. Packt Publishing, 2021.

JARED HANSON. Passport.js. Disponível em: https://www.passportjs.org/. Acesso em: 6 nov. 2022.

KINSBRUNER, ERAN, e BAHMUTOV, GLEB. A A Frontend Web Developer's Guide to Testing: Explore Leading Web Test Automation Frameworks and Their Future Driven by Low-code and AI. Packt Publishing, 2022.

MARQUEZ-SOTO. Pedro. Backend Developer in 30 Days: Acquire Skills on API Designing, Data Management, Application Testing, Deployment, Security and Performance Optimization (English Edition). BPB Publications, 2022.

MASSE. Mark. REST API Design Rulebook. O'Reilly Media, 2011.

MUELLER. John Paul. Functional Programming For Dummies. Wiley, 2019.

PRISMA. DATABASE TOOLS FOR MODERN APPLICATION DEVELOPMENT. Disponível em: https://www.prisma.io/. Acesso em: 6 nov. 2022.

SARCAR. Java Design Patterns: A Hands-On Experience with Real-World Examples. Apress, 2022.

TAILWINDLABS. Utility-First Fundamentals - Tailwind CSS. Disponível em: https://tailwindcss.com/docs/utility-first#maintainability-concerns. Acesso em: 6 nov. 2022.

TAILWINDLABS. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. Disponível em: https://tailwindcss.com/.

TRILON. Documentation | NestJS - A progressive Node.js framework. Disponível em: https://docs.nestjs.com/recipes/prisma#prisma. Acesso em: 6 nov. 2022.

TRILON. Documentation | NestJS - A progressive Node.js framework. Disponível em: https://docs.nestjs.com/guards#guards. Acesso em: 6 nov. 2022.

TRILON. Documentation | NestJS - A progressive Node.js framework. Disponível em: https://docs.nestjs.com/security/authentication#implementing-passport-strategies.Acesso em: 6 nov. 2022.

TRILON. Documentation | NestJS - A progressive Node.js framework. Disponível em: https://docs.nestjs.com/security/authentication#implementing-passport-jwt. Acesso em: 6 nov. 2022.

VERCEL. Advanced Features: Middleware | Next.js. Disponível em: https://nextjs.org/docs/advanced-features/middleware. Acesso em: 6 nov. 2022.

VERCEL. Data Fetching: getServerSideProps | Next.js. Disponível em: https://nextjs.org/docs/basic-features/data-fetching/get-server-side-props. Acesso em: 6 nov. 2022.

VERCEL. Learn | Next.js. Disponível em: https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs. Acesso em: 6 nov. 2022.

VERCEL. getServerSideProps | Next.js. Disponível em: https://nextjs.org/docs/basic-features/data-fetching/get-server-side-props. Acesso em: 6 nov. 2022.