Caio Vinícius Pinto Zirretta

Nolan - Software de Gestão de Salas de Cinema

Goiânia
2023

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): Caio Vinícius Pinto Zirretta

Título do trabalho: Nolan - Software de Gestão de Salas de Cinema

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Marcelo Stehling De Castro**, **Professor do Magistério Superior**, em 17/08/2023, às 18:30, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Caio Vinicius Pinto Zirretta**, **Discente**, em 17/08/2023, às 18:35, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **3973579** e o código CRC **EC149B14**.

Referência: Processo nº 23070.024052/2023-48 SEI nº 3973579

Caio Vinícius Pinto Zirretta

Nolan - Software de Gestão de Salas de Cinema

Trabalho de conclusão de curso apresentado na Escola de Engenharia Elétrica, Mecânica e de Computação como requisito para a conclusão do curso de Engenharia de Computação e obtenção do título de Engenheiro de Com putação.

Universidade Federal de Goiás – UFG

Escola de Engenharia Elétrica, Mecânica e de Computação (EMC)

Orientador: Prof. Dr. Marcelo Stehling de Castro

Goiânia

2023

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Zirretta, Caio Vinícius Pinto
Nolan [manuscrito] : Software de Gestão de Salas de Cinema / Caio Vinícius Pinto Zirretta. - 2023.
xvi, 16 f.: il.

Orientador: Prof. Dr. Marcelo Stehling de Castro.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia da Computação, Goiânia, 2023.
Bibliografia.
Inclui siglas, fotografias, abreviaturas, tabelas.

1. Sistema web. 2. Angular. 3. TypeScript. 4. ERP. 5. Prisma. I. Castro, Marcelo Stehling de, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Aos oito dias do mês de agosto do ano de dois mil e vinte e três iniciou-se a sessão pública de defesa do Projeto Final de Curso (PFCC) intitulado “Nolan - Software de Gestão de Salas de Cinema”, de autoria de Caio Vinícius Pinto Zirretta, do curso de Engenharia de Computação, da Escola de Engenharia Elétrica, Mecânica e de Computação da UFG. Os trabalhos foram instalados pelo Prof. Dr. Marcelo Stehling de Castro - EMC/UFG com a participação dos demais membros da Banca Examinadora: Engenheiro Especialista Gustavo Dias de Oliveira e Prof. Dr. Rodrigo Pinto Lemos EMC/UFG. Após a apresentação, a banca examinadora realizou a arguição do(a) estudante. Posteriormente, de forma reservada, a Banca Examinadora atribuiu a nota final de dez, tendo sido o PFC considerado aprovado.

Proclamados os resultados, os trabalhos foram encerrados e, para constar, lavrou-se a presente ata que segue assinada pelos Membros da Banca Examinadora.

Documento assinado eletronicamente por **Rodrigo Pinto Lemos**, **Professor do Magistério Superior**, em 23/08/2023, às 15:56, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Marcelo Stehling De Castro**, **Professor do Magistério Superior**, em 23/08/2023, às 16:21, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Gustavo Dias De Oliveira**, **Técnico de Tecnologia da Informação**, em 23/08/2023, às 16:55, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **3988055** e o código CRC **E663DCC2**.

Referência: Processo nº 23070.024052/2023-48 SEI nº 3988055

Nolan – *Software* de gestão de Salas de Cinema

Caio Vinícius P. Zirretta[1] — Marcelo Stehling de Castro[2]
Universidade Federal de Goiás (UFG) - Escola de Engenharia Elétrica, Mecânica e de Computação (EMC) - Goiânia, Goiás, Brasil 74601-010, e-mails: caiovpz@discente.ufg.br[1], mcastro@ufg.br[2].

Resumo* — O crescimento da utilização da tecnologia nos últimos anos tem impulsionado uma corrida para que os pequenos e grandes negócios busquem soluções mais modernas e eficientes. Estas soluções abrangem desde a organização de uma empresa até a realização de operações vitais, sendo um elemento chave para aumentar a produtividade, conquistar mais clientes, descobrir novos mercados, e impulsionar um produto ou serviço. Estes fatores contribuem com elemento primordial que uma empresa sempre deseja: aumentar seus lucros. Soluções web, comparadas às convencionais, possuem uma fácil adaptabilidade, uma vez que podem ser acessadas da maioria dos dispositivos eletrônicos utilizados hoje em dia e não exigem um alto poder computacional. O acesso aos navegadores é o único pré-requisito para utilizá-las, eliminando a necessidade da instalação de ferramentas adicionais que podem prejudicar o desempenho de um computador ou celular. Tendo isto em vista, o objetivo deste projeto é criar uma solução web que possa auxiliar na solução de problemas de um cinema, como a organização de sessões em salas de cinema, cadastro e alteração de filmes, registro de bilheteria e proporcionar ao cliente uma experiência de consumo agradável e direta, podendo ser utilizado tanto em computadores quanto em totens de autoatendimento.

***Palavras-chaves* — Sistema web, Angular, TypeScript, ERP, Prisma, gerenciamento de cinemas.**

Abstract* — The growth of technology utilization in the latest years has been promoting a race towards small e big business to search for the most modern and efficient solutions. Those solutions range from a company organization to even perform vital operations, as it is a key element to increase productivity, earn more clients, discover new markets, and propel a product or a service. Those factors contribute to the primordial element that a business always desire: to increase profit. Web solutions, compared to conventional ones, have an easy adaptability as they can be accessed from most of the electronic devices used nowadays and do not demand a high computer power. A browser access is the only required to utilize them, eliminating the necessity for any additional tools that could harm a mobile or PC performance. The objective of this project is to create a web solution that could assist on solving common cinemas problems, such as sessions' organization in their respective movie rooms, the register and modification of movies registers, registry of tickets and provide the customers a direct and enjoyable consumption experience.

***Keywords* — Web system, Angular, TypeScript, ERP, Prisma, cinema management.**

I. Introdução

Na sequência será apresentada a motivação para escolha do tema e os objetivos do trabalho, bem como a forma como ele foi estruturado.

A. Motivação

A motivação deste projeto é a oportunidade de praticar a construção de um projeto de *software* completo. Por vezes, o estudo limita-se ao aprendizado teórico, porém, quando a oportunidade de atuar no mercado profissional de software se torna uma necessidade, a prática vira indispensável [1].

Em uma área tão competitiva quando a do desenvolvimento de software, graças à expansão dos serviços online devido à pandemia do Coronavírus [2], praticar a criação de um projeto, desde o planejamento até os testes, é uma excelente oportunidade para aumentar o próprio conhecimento e descobrir novas técnicas.

B. Objetivos

O objetivo deste projeto é praticar o desenvolvimento de um projeto de software completo e funcional utilizando metodologias ágeis.

Para tanto, alguns objetivos específicos foram definidos:

1) Construção de um projeto de software completo e funcional, utilizando ferramentas modernas de desenvolvimento web;
2) Com a funcionalidade que permita fazer a gestão de salas de cinema;
3) Com a possibilidade de fornecer uma opção de negócio para um pequeno estabelecimento;
4) Funcione nos principais navegadores de internet utilizados atualmente;
5) E possa ser facilmente implantado e utilizado.

C. Estrutura do Trabalho

Este trabalho está organizado em sete seções que abordam as diferentes etapas para a elaboração do projeto, começando pela idealização e indo até o resultado final.

A seção I destaca a ideia por trás do projeto, apresentando os objetivos do trabalho e a forma como ele foi estruturado.

A seção II descreve o ambiente de desenvolvimento e quais ferramentas foram utilizadas nas diferentes etapas do projeto.

A seção III aborda os métodos de desenvolvimento, separando e descrevendo o produto em três frentes diferentes que se interconectam.

A seção IV descreve como o projeto foi idealizado e planejado para atender à ideia inicial e utilizar adequadamente a metodologia ágil proposta.

A seção V mostra o resultado do software construído, dividido em três módulos que operam em conjunto.

Na seção VI são apresentados os testes realizados para garantir a qualidade e usabilidade do produto construído, como também as considerações finais a respeito do tema.

As conclusões são descritas na seção VII, relatando se foi possível atender às expetativas iniciais e planejadas.

II. Softwares para Desenvolvimento

Para o desenvolvimento do software, foram utilizados os softwares mostrados na Fig. 1 e que serão detalhados a seguir.

Fig. 1. Softwares usados no desenvolvimento.
Fonte: elaborado pelo autor.

A. *WebStorm*

Esta IDE é um dos produtos mais antigos da JetBrains s.r.o. É um ambiente de desenvolvimento completo com funcionalidades voltadas para aplicações JavaScript e seu ecossistema [3].

Ela conta com um ambiente de desenvolvimento completo e repleto de funcionalidades que auxiliaram na elaboração deste projeto, como um robusto sistema de refatoração de código, indexação precisa, ferramentas de conexão com banco de dados, capacidade de nativamente realizar depuração de código e integração com Git e GitHub.

Por utilizar JavaScript em todas as camadas e módulos, a escolha desta IDE foi fundamental para trazer facilidade e velocidade ao trabalho, uma vez que não foram necessárias outras ferramentas para o desenvolvimento em si, apenas para os testes. É possível a instalação de plug-ins que a complementam com novas funcionalidades, mesmo que apenas estéticas, como a mudança das cores, dos botões e das fontes do WebStorm.

Por mais que seja uma ferramenta capaz de fornecer funcionalidades práticas e úteis, seu acesso ainda depende de uma assinatura mensal ou anual, com a possibilidade de um teste gratuito por 30, o que pode afastar a maioria dos usuários, tendo em vista outros editores que são gratuitos. Para estudantes, a Jetbrains fornece uma licença gratuita para todos os seus produtos durante o tempo que o estudante estiver em atividade.

B. *Postman*

Postman é uma plataforma de API feita para o desenvolvimento de APIs. Seu uso gratuito requer apenas um registro no site de sua desenvolvedora para que seja acessado. Com esta conta é possível sincronizar configurações e coleções entre vários dispositivos desktop [4].

Com uma estrutura dividida em espaços de trabalho, no Postman é possível criar coleções, dentro desses espaços, no qual é possível organizar, por pastas, requisições que utilizam o protocolo HTTP.

A partir de uma simulação de servidor HTTP, o Postman é capaz de enviar as requisições completamente personalizáveis para qualquer destino, desde que este seja uma URL. É possível alterar o cabeçalho, o corpo, a forma de autenticação, o método HTTP da requisição, dentre outras.

Para facilitar essa personalização, a plataforma oferece a criação de variáveis que armazenam valores que possivelmente irão se repetir entre as requisições, como um usuário ou uma porta de conexão. É possível também programar scripts para serem executados antes de cada requisição, sendo possível automatizar um processo de autenticação que antes seria feito manualmente.

No contexto deste projeto, o Postman foi utilizado para testar o servidor, com requisições organizadas a partir do domínio da aplicação, para que fosse possível testar diferentes funcionalidades individualmente, antes que fosse construída uma interface para ela.

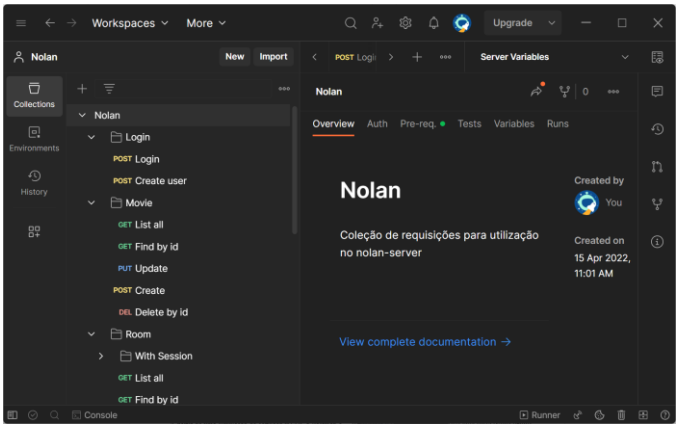

Fig. 2. Coleção de requisições utilizadas no projeto.
Fonte: elaborado pelo autor.

A Fig. 2Fig. 1 mostra parte da coleção utilizada para os testes no **Nolan**, organizadas pelas entidades da aplicação.

Ao enviar uma requisição, é possível visualizar a resposta que o servidor gerou para que seja feita a análise dos dados e concluir se foram tratados corretamente pela aplicação.

C. *DBeaver*

Esta ferramenta construída em Java é um cliente SQL que permite a comunicação com o banco de dados diretamente. É um software multiplataforma de código aberto disponível no GitHub que necessita apenas do JDK para funcionar [5].

Em sua interface é possível a conexão com múltiplos bancos de dados a partir de seus hosts e portas, informando o usuário adequado para cada um.

Após a conexão, é possível rodar scripts SQL diretamente no banco de dados e visualizar os resultados de forma gráfica. Por salvar os scripts internamente e não ser necessária uma instalação, o DBeaver se mostrou uma opção leve e prática para realizar as operações básicas de banco de dados.

D. *Git*

Bastante conhecido no mundo do desenvolvimento [6], este aplicativo multiplataforma é um sistema de controle de versionamento de arquivos, comumente aplicado em projetos de software [7].

Contendo uma estrutura rebuscada de comandos de terminal, para utilizá-lo é necessária apenas a instalação e inicialização na pasta do projeto, assim será criada uma pasta oculta dentro do projeto que irá monitorá-lo. O Git funciona de forma integrada aos terminais de sistemas operacionais como Windows, Linux e MacOS.

Através dele, por meio de códigos de linha de comando, é possível realizar um versionamento detalhado do código em que é possível visualizar todas as alterações até um determinado momento em relação a um momento anterior e até revertê-las para o estado anterior individualmente, se necessário. Ele pode ser configurado para que ignore determinados arquivos, pastas,

ou tipos de arquivos, se estes não forem relevantes para o projeto ou comprometerem sua segurança.

Além das funcionalidades citadas, o Git possui integração com repositórios em nuvem, tornando fácil não só o desenvolvimento colaborativo como para multiplataforma.

Uma prática comum no mercado de software, onde vários desenvolvedores podem atuar no mesmo projeto, é criar ramificações do código a partir de uma ramificação original para incluir ou corrigir funcionalidades e depois reintegrá-la na ramificação original.

Esse modelo de trabalho permite que alterações individuais sejam testadas antes de serem reintegradas em um sistema já funcional e estável.

E. GitHub

Apesar do nome fazer parecer que é um derivado do Git, esta plataforma online não foi desenvolvida pelos mesmos criadores do Git. O GitHub foi desenvolvido em 2008 como um repositório online de projetos de software gerenciados por Git. Foi adquirido pela Microsoft em 2018 [8].

Dentro dessa plataforma é possível armazenar códigos públicos e privados que podem ser clonados para qualquer máquina. A clonag em de um código se refere ao processo de copiá-lo por completo, incluindo seus arquivos de versionamento de projeto, para que seja possível criar mais ramificações e alterá-lo de acordo com o que já foi desenvolvido anteriormente.

Através de comandos do Git, é possível enviar e controlar esses repositórios em nuvem diretamente de qualquer computador ou celular através de uma chave SSH e de uma conta registrada no site.

Dentre as funcionalidades oferecidas pelo GitHub que vão além de um simples repositório estão o *copilot*, uma ferramenta de inteligência artificial que auxilia na construção do código em tempo real, o *web-based code editor*, um editor visualmente semelhante ao VS Code, porém completamente utilizável no navegador para poder visualizar o código fonte de um projeto de maneira mais prática e rápida através da sua indexação.

O GitHub oferece também, de forma independente, o GitHub Desktop, uma interface gráfica que substitui os comandos via terminal do Git.

Com este aplicativo, a integração entre o Git e o GitHub se torna ainda mais fácil e rápida, uma vez que é possível visualizar explicitamente tanto o versionamento quanto as ramificações do código e não ser necessário decorar sequer um comando ou registrar manualmente a chave SSH que conecta as duas pontas.

F. Google Chrome

Sendo uma aplicação web, este projeto necessita de uma plataforma adequada para a execução do lado do cliente. Sendo assim, foi escolhido o navegador Google Chrome para executá-lo em ambiente de desenvolvimento.

Esta escolha se limita a um viés pessoal, uma vez que este tipo de aplicação deve ser acessível a maior quantidade de navegadores disponíveis, respeitando a limitação de cada um.

Não só apenas para executar o projeto, o Google Chrome contém um robusto mecanismo de testes chamado de Chrome DevTools [9].

Essa ferramenta possui diversas abas, dentre elas, destaca-se a de “Elementos”, que permite analisar o código fonte de um website para que o desenvolvedor possa testar funções de UI/UX sem precisar alterar o código e compila-lo novamente.

Também existem as abas de “Console”, no qual os erros e avisos da aplicação podem ser exibidos e a aba de “Rede”, onde, por meio de gráficos e tabelas, todas as conexões da aplicação com serviços ou sistemas externos são exibidos, assim como características relevantes, como o tempo de resposta e um link da conexão que leva diretamente ao código que a executou.

Utilizou-se neste projeto, com menor frequência, as abas “Armazenamento”, onde são mostrados os armazenamentos locais e os cookies do website e a “Fontes”, onde o código fonte compilado é exibido.

G. Node.js

Inicialmente a linguagem JavaScript foi desenvolvida para ser executada apenas em navegadores [10], porém, essa realidade mudou em 2009 quando Ryan Dahl lançou o Node.js [11].

Feito a partir do interpretador de códigos do Google Chrome, o V8, o Node.js é um software de código aberto multiplataforma que permite a execução de códigos JavaScript diretamente de um terminal, como ocorre em outras linguagens como C, ou Java.

Seu funcionamento utiliza a modalidade *single-threaded*, no qual apenas uma *thread* executa todas as operações em uma hierarquia chamada de *event loop*, como representado na Fig. 3.

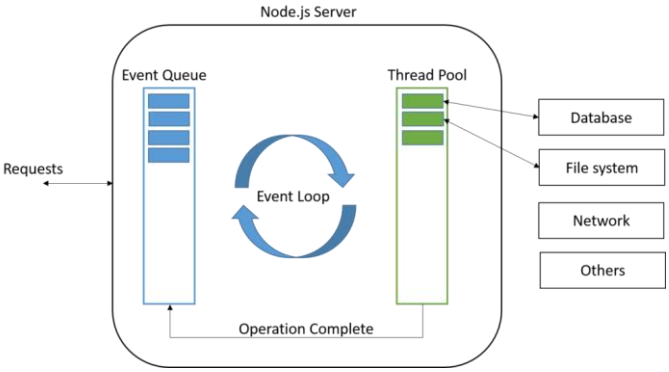

Fig. 3. *Event loop* do NodeJS.
Fonte: elaborado pelo autor.

O Node é comumente utilizado com o Node Package Manager, um gerenciador de pacotes online que registra módulos de código e os deixa disponíveis para download em qualquer projeto através de linhas de comando.

Com essas duas ferramentas acopladas, criar um servidor ou acessar uma árvore de arquivos se tornou algo possível no ambiente do JavaScript, tornando a linguagem mais versátil.

No contexto desse projeto, o Node foi utilizado tanto no lado do servidor, na criação de uma API, quanto no lado do cliente, pois, ao utilizar um framework para desenvolvimento, como o Angular, é necessário que seu código seja previamente compilado para que funcione nos navegadores, pois estes são capazes apenas de interpretar HTML, CSS e JavaScript puros.

III. MÉTODOS

A. *Back-end*

O lado da aplicação chamado de *"server side"*, também conhecido como *"back-end"*, é onde ocorre o fluxo de tratamento de dados de uma aplicação [12].

Neste projeto, a estrutura adotada foi de uma API (*Aplication Programming Interface*) Restful, uma aplicação que possui um conjunto de funções e serviços abertos para utilização [13]. A API foi denominada "Nolan Server" por funcionar de forma similar a um servidor HTTP.

Esta estrutura se baseia no modelo MVC (*Model-View-Controller*). Como o nome sugere, este modelo descreve três camadas distintas que trabalham em conjunto para que uma aplicação funcione [14].

A camada *View* descreve como as informações serão dispostas ao usuário, ou cliente, por exemplo, como será a resposta de uma requisição.

A camada *Model* descreve como os dados serão modelos pela aplicação, por exemplo, como um usuário, mapeado em um banco de dados por uma tabela, será representado em nível de código.

Por fim, a camada *Controller* é aonde as informações chegam na requisição, sendo encaminhadas ao *Model* para que este faça seu trabalho e depois ao *View* para ser devolvida ao *Client*¸ servindo como uma ponte entre os dois.

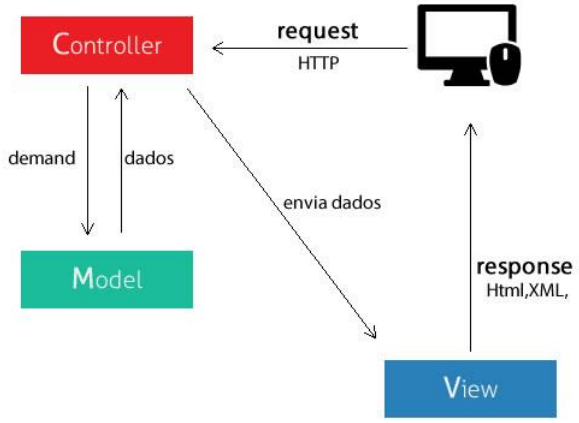

Fig. 4. Exemplificação de um modelo MVC.
Fonte: elaborado pelo autor.

A Fig. 4 ilustra o comportamento da API com a utilização do modelo MVC, com a diferença que neste projeto a *View* não retorna um HTML ou XML e sim um JSON.

Nesse projeto, o *View* e o *Controller* atuam em conjunto através da arquitetura de *Resource*, uma classe responsável por receber a requisição HTTP, validá-la, enviá-la para as demais camadas e retornar a resposta de volta para o cliente. Foi utilizada a biblioteca Zod para a validação dos dados recebidos.

Essa biblioteca oferece uma maneira simples de criar *schemas* que representam objetos e validá-los. É possível fazer uma validação com a chamada de uma única função *parse* em cima de uma *schema*, por exemplo [15].

O *Model*, sendo a camada mais complexa, foi dividido em três subcamadas: o *Domain*, no qual são armazenadas as estruturas que modelam as entidades utilizadas no projeto e suas *schemas* do Zod, o *Service*, onde as regras de negócios referentes aos dados são executadas, como uma regularização de formato de data, e o *Repository*, uma classe que se conecta diretamente com o banco de dados para realizar as operações necessárias e validações que necessitem dessa conexão, como a verificação da pré-existência de um usuário antes de cadastrá-lo. A hierarquia simplificada pode ser visualizada na Fig. 5.

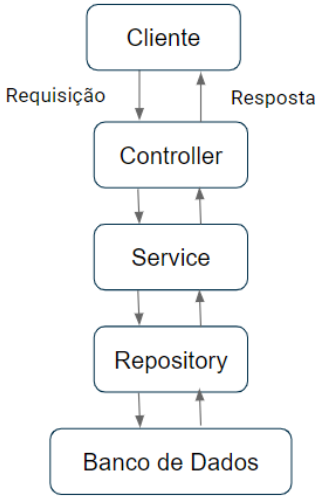

Fig. 5. Padrão de projeto MVC utilizado no projeto.
Fonte: elaborado pelo autor.

Para esta API, foi utilizado o TypeScript, um *superset* de JavaScript com suporte a novas funções, como a tipagem forte, abstrações por interfaces e anotações [16]. Com o TypeScript a criação e validação de modelos e entidades se torna um processo mais ágil e eficiente graças à tipagem e a integração com esta linguagem por parte da IDE escolhida.

Nativamente os navegadores não são capazes de interpretar um código em TypeScript, porém, por ser uma API executável diretamente do sistema operacional, esta aplicação necessita apenas ser executada para que escute possíveis comunicações com ela.

Para isto o Node.js se tornou a ferramenta ideal para sua execução, uma vez que este é capaz de interpretar o *superset* sem a necessidade de quaisquer configurações.

A fim de facilitar a construção de um servidor, foi utilizada a framework Express.js. Ela fornece, através de camadas de abstrações, opções de construções de servidores e roteamento apenas com chamadas simples de funções [17].

Por meio do *framework*, é possível criar um servidor simples em poucas linhas, conforme demonstrado na Fig. 7. Neste exemplo, o servidor é apenas inicializado sem ter rotas configuradas.

```
const app = express();

const port: number = 3000;

app.listen(port, () => {
    console.log("server listening on port " + port);
});
```

Fig. 6. Exemplificação de um servidor construído com Express.js.
Fonte: elaborado pelo autor.

As requisições são enviadas a *endpoints* em formato de URI para a API através de métodos HTTP como GET, POST, PUT e PATCH. A porta de recebimento pode ser costumizada de

acordo com a necessidade através das variáveis de ambiente do projeto, aumentando a sua segurança.

O Express.js permite que sejam feitas diversas configurações de comunicação, como o bloqueio personalizado de requisições pelo CORS (*Cross-Origin Resource Sharing*).

Por padrão, o servidor pode ser acessado por qualquer usuário caso este tenha acesso ao seu nome de host, porta e *endpoint*, o que pode representar um enorme risco de segurança. O CORS atua justamente limitando a origem da requisição para os endereços desejados [18], como um *middleware*, para que apenas os outros módulos do projeto tenham acesso à API.

A configuração do CORS utilizada no servidor utiliza de variáveis de ambiente de projeto, um arquivo com a extensão ".env" localizado na raiz do projeto que contém linhas representando uma variável e um valor do tipo "VARIAVEL='VALOR'".

Este arquivo, ignorado pelo Git por questões de segurança, deve ser escrito manualmente em cada implantação do projeto, uma vez que as informações de conexão de banco e URLs irão variar a cada ambiente.

A proposta do de desenvolvimento do *back-end* se baseia no modelo DDD, *Domain-Driven Design*, no qual o projeto é construído com base nos domínios, ou entidades, que representam as tabelas no banco de dados [19].

Na prática, cada camada foi construída com referência a um domínio. Cada um deles possui seu respectivo Controlador, ou Recurso, Serviço e Repositório, para que suas operações sejam bem definidas e organizadas e as regras de negócios não se misturem acidentalmente.

Para se conectar com um banco de dados, foi utilizada uma ORM, uma técnica de mapeamento relacional de objetos que transforma tabelas em SQL em objetos utilizáveis em código [20].

O Prisma cumpre essa função de forma simples. Esta ORM possui suporte nativo a TypeScript e aos mais populares bancos de dados, como MySQL, Postgre e Oracle [21].

Após a instalação em um projeto, a configuração do Prisma é simples, necessitando apenas da string de conexão com o banco e a criação de um arquivo chamado "schema.prisma".

Este arquivo possui uma sintaxe simples para modelar os objetos de banco, no qual o *Model* será o nome da tabela, os atributos serão as colunas e os tipos serão os tipos de dados das colunas. Um exemplo de *schema* de Usuário pode ser visto na Fig. 7.

```
model User {
  id        String    @id @default(uuid())
  user      String
  password  String
  createdAt DateTime?

  @@map("users")
}
```

Fig. 7. Exemplificação de um modelo no Prisma.
Fonte: elaborado pelo autor.

Dentro do *schema* é permitido definir a chave primária e secundária, assim como definir valores padrões mais complexos, como um uuid. Valores opcionais também são permitidos com a adição de uma interrogação após a tipagem.

Por padrão, o nome da tabela será igual ao nome do Model, a não ser que seja explicitado outro através do comando "@@map("nome_da_tabela")".

Com o *schema* finalizado, basta usar os comandos prontos da ORM para executar a migração, o processo que irá transformar cada Model do arquivo em uma tabela diferente [22].

O arquivo pode ser alterado a qualquer momento, sendo necessário executar outra migração após essa. Todas elas ficam salvas dentro da pasta "*migrations*", onde o SQL bruto gerado pode ser visto.

O Prisma dispõe outra biblioteca chamada de Prisma Client, que dispõe de objetos e tipos gerados pela própria ferramenta, durante a migração, que representam os modelos criados. A Fig. 8 mostra o tipo gerado a partir da *migration* da Fig. 7.

```
export type User = {
  id: string
  user: string
  password: string
  createdAt: Date | null
}
```

Fig. 8. Tipo gerado automaticamente pelo Prisma.
Fonte: elaborado pelo autor.

Esses objetos podem ser acessados em qualquer lugar da aplicação a partir de um objeto global "prisma" e contam um leque de funções que simulam as operações de bancos para que não seja necessário escrever um comando SQL sequer, como demonstra a Fig. 9.

```
await prisma.user.create({
    data: {
        user: user.username,
        password: hashPassword,
        createdAt: user.createdAt,
    }
});
```

Fig. 9. Inserção de um registro utilizando o Prisma.
Fonte: elaborado pelo autor.

Todas as informações e operações são passadas ao ORM via código, por exemplo, o INSERT é equivalente à função *create* e o SELECT é desmembrado em várias funções, como *findMany*, *findFirst* ou *findUnique*, cujos nomes são bastante intuitivos.

B. Front-end

O lado da aplicação chamada de "*client side*", também conhecido como "*front-end*", é onde o usuário irá fazer toda sua interação e poderá visualizar os dados disponíveis a ele pelo *server side* [23].

Diferente do *back-end*, em que apenas dados são transformados, no *front-end* é necessário dispô-los de uma forma acessível e, caso seja possível, elegante.

Essas duas características são normalmente referenciadas como UX e UI [24], dois campos de estudo complexo que se

integram à tecnologia nesta camada, afinal, por mais que um software seja funcional e otimizado, se o usuário não souber como manuseá-lo, ele não terá utilidade.

A maior barreira para o desenvolvimento web focado em responsividade e usabilidade são os próprios navegadores. Estes, conforme citado anteriormente, são capazes de ler apenas HTML, CSS e JavaScript.

Essas três tecnologias andam lado-a-lado na construção de um website, uma analogia interessante é compará-las a um carro, conforme a Fig. 10. O HTML, *HyperText Markup Language*, funciona como o esqueleto da página, demarcando componentes da página e posicionando o texto de acordo com a estrutura utilizada a partir de *tags* [25].

O CSS, *Cascading Style Sheets*, é uma ferramenta de estilização para o HTML. Por meio de referências como classes ou ids, é possível alterar cores de texto, bordas, mudar a fonte ou até mesmo adicionar animações completas [26].

Não só estilização por beleza, o CSS é responsável por grande parte da responsividade e portabilidade de um website, sendo capaz de aplicar alterações visuais distintas entre diferentes tamanhos de tela do navegador, permitindo que exista uma interface completamente diferente entre um ambiente desktop e outro mobile.

O projeto, pensado para ser uma aplicação exclusiva para desktop, visa a interação com o usuário através de uma tela maior, sensível ao toque ou com a utilização de mouse e teclado, não sendo portável para dispositivos móveis.

O JavaScript, por fim, é onde a programação acontece de fato. Ele é capaz de acessar uma página web através do chamado DOM, *Document Oriented Module*, um objeto global e nativo responsável por modelar toda a página [27].

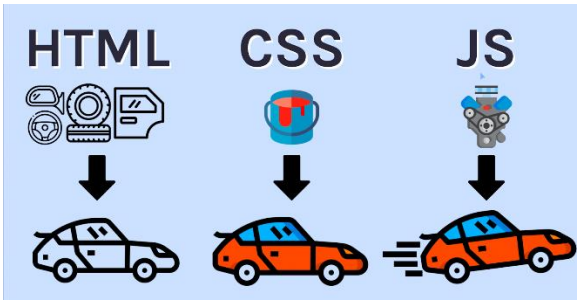

Fig. 10. Analogia do relacionamento entre HTML, CSS e JavaScript.
Fonte: elaborado pelo autor.

Por mais que o DOM forneça funções úteis para acessar componentes e modificá-los dinamicamente, esta operação é bastante trabalhosa quando feita diretamente devido à alta complexidade.

Para resolver esse problema, ao longo dos anos diversos *frameworks* de JavaScript foram criadas, operando de formas diferentes e com finalidades das mais diversas, quase funcionando como linguagens distintas.

Dentre elas, a escolhida para este projeto foi o Angular em sua versão 16. Desenvolvido pela Google em 2009, ele já passou por uma reestruturação completa em 2016, se dividindo entre AngularJS e Angular 2, comumente referenciado sem o número [28].

Esse framework funciona com uma CLI simples e de fácil uso, no qual é possível criar projetos, gerar novos componentes ou inicializar a aplicação como um servidor de desenvolvimento *standalone*.

Nativamente o Angular possui suporte a TypeScript, frameworks de CSS como SASS ou SCSS e bibliotecas de componentes como o Material UI, sendo esta última a única que precisa de comandos adicionais para instalação.

Dentre as funcionalidades mais relevantes para o projeto a que mais se destaca é a componentização. Os elementos da página podem ser gerados individualmente e modificados de acordo com a necessidade.

É interessante dividir os elementos em componentes do menor tamanho possível, uma vez que essa prática ajuda na manutenção e leitura do código, apesar de não refletir visualmente para o usuário.

Por padrão, componentes no Angular são gerados pela CLI em uma pasta própria e divididos em quatro arquivos: o *template* HTML, o CSS, a classe em TypeScript e um arquivo de testes.

Apesar da quantidade de arquivos, somente o de classe é necessário para o funcionamento, porém, a fim de seguir o Clean Code, o padrão original do framework foi mantido.

Elementos do componente podem ser acessado por meio de diretivas no chamado *databinding*, no qual ocorre um ciclo contínuo de mudanças no View e no Model que fazem com que se atualizem constantemente e permitam uma interação dinâmica com o usuário.

O *databinding* permite transitar dados, ou variáveis, entre o JavaScript e o HTML diretamente, ou entre componentes pais e filhos. Existe tanto o *one-way* quanto o *two-way*, se for necessário que o dado seja apenas enviado ou recebido também, respectivamente. A Fig. 11ilustra esse processo de uma forma direcional e com exemplos.

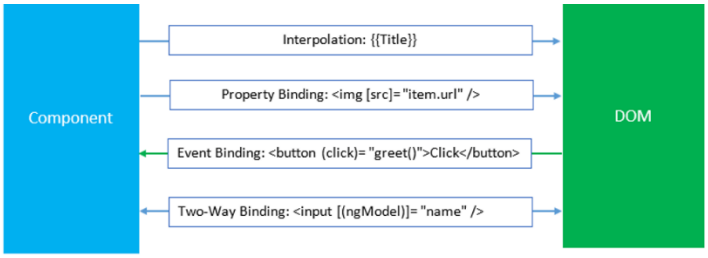

Fig. 11. Formas de *databinding* do Angular.
Fonte: elaborado pelo autor.

Como foi citado anteriormente, a API do *back-end* é responsável por trazer os dados diretamente do banco. Essa comunicação é feita a partir de um *client* HTTP nativo do Angular cujo uso é fácil e direto, basta informá-lo o método, url e, opcionalmente, cabeçalho e corpo para enviar uma requisição.

Por padrão a aplicação envia no cabeçalho apenas uma sinalização de tipo de conteúdo e o token de autenticação necessário para autorizar o acesso aos *endpoints*.

As classes chamadas de *services* são o padrão para organizar os métodos, modelados nesse projeto de acordo com o DDD novamente, para que alterações no *back-end* sejam facilmente reproduzíveis no front-end.

```
private token: string = this.cookieService.get('token');
private url: string = 'http://localhost:3333/movie';
private headers: Record<string, HttpHeaders> = {
  headers: new HttpHeaders({
    'Content-type': 'application/json',
    'Authorization': `Bearer ${this.token}`
  })
};

constructor(private httpClient: HttpClient, private cookieService: CookieService) {
}

list(): Observable<Movie[]> {
  return this.httpClient.get<Movie[]>(this.url, this.headers);
}
```

Fig. 12. Construção de método HTTP com URL e cabeçalho.
Fonte: elaborado pelo autor.

A Fig. 12 exemplifica a criação de um serviço que encapsula o cliente HTTP nativo do Angular e é capaz de fazer uma requisição para um *endpoint* específico enviando os itens necess

Esses dados são recebidos e encapsulados em *Observables*, objetos da biblioteca rxjs, nativa no Angular, para que sejam tratados de forma assíncrona pela aplicação [29].

Na estrutura desses objetos é possível tratar não só os dados, de fato, como os erros recebidos pelo *back-end*, sendo possível disponibilizar mensagens que indiquem ao usuário o problema que está acontecendo no momento.

Com todos os componentes e classes utilitárias prontas, a aplicação está pronta para ser executada. Porém, como os navegadores não são capazes de interpretar não só o JavaScript como a estrutura do Angular.

É necessário utilizar o *build* do framework para que seja gerada uma simples página HTML estática, com um arquivo de CSS global e os respectivos scripts gerador a partir de todos os arquivos TypeScript.

Todas as animações, serviços, componentes e classes utilitárias são condensadas nesses arquivos de forma incompreensível para um ser humano ler, porém funcionais para uma máquina.

O *build* funciona como uma compilação de fato, não só uma tradução direta, pois muitos recursos do Angular sequer existem no HTML convencional, como suas diretivas, funções que manipulam os comportamentos dos componentes diretamente no *template* [30].

C. *Banco de dados*

O Sistema de Gerenciamento de Banco de Dados (**SGBD**) escolhido para este projeto foi o PostgreSQL, dentre diversas opções de bancos de dados, pois se destaca por algumas características que serão apresentadas a seguir.

Sua modelagem de dados permite tipos de dados mais complexos, como JSONs puros, XMLs, imagens, vídeos e outros, sem a necessidade de conversão para bytes, ou CLOBs, previamente [31].

O banco permite acessos simultâneos, com exceções de alterações do mesmo registro. Isto permite que diferentes módulos ou usuário interajam com as tabelas em tempo real. Em um ambiente em que vários clientes acessam a mesma aplicação, esta funcionalidade é vital.

Ele possui suporte a backup e recuperação de dados tanto off-line quanto online, permitindo que haja uma segurança adicional no caso de algum problema com o servidor em que esteja funcionando.

Todas essas características contribuem para uma boa escalabilidade. O escopo deste projeto é focado em um gerenciamento individual de um negócio, porém, com ajustes no banco, seria possível torná-lo adaptado a um sistema de filiais.

O Modelo Entidade Relacionamento (**MER**), procura ser o mais simples possível para evitar uma complexidade desnecessária neste projeto, que não visa ser comercializado inicialmente.

As principais entidades são: Sala, Filme e Sessão, que representam o que seus nomes indicam. Os filmes possuem Ids únicos e informações básicas. As Salas dispõem de números e uma lista de sessões, que contam com uma referência a um filme, dia, horário e uma lista de lugares disponíveis.

IV. MODELAGEM DO SISTEMA

Nessa seção é apresentada a metodologia utilizada para o desenvolvimento do projeto, com uma discussão sobre a análise de requisitos, uma explicação sobre o estudo de caso e o banco de dados utilizado.

A. *Metodologia de Projeto*

Assim como um projeto de qualquer natureza, o desenvolvimento de um *software* pode necessitar de uma metodologia específica para atingir um resultado esperado [32].

Tendo isso em mente, é importante escolher uma metodologia de projeto adequada para atender os requisitos definidos, ou ir além deles.

Definir padrões de implantação, desenvolvimento e o uso de ferramentas para executar um projeto não é uma tarefa fácil, a escolha errada pode atrasar a entrega do produto.

Apesar deste projeto não necessitar de uma ferramenta de gestão complexa, foi optado por utilizar uma metodologia conhecida como *ágil*.

Diferente do desenvolvimento tradicional, no qual itens como escopo e cronograma são definidos com antecedência e se mantêm constantes até o final, o ágil trabalha de forma diferente e dinâmica [33].

A metodologia ágil é adaptativa ao andamento do projeto. Isso significa que existe maior liberdade para redefinir requisitos, repensar estratégias e remodelar o produto.

O cronograma é flexível para incluir eventuais etapas que necessitem de maior tempo, tornando o modelo mais tolerante a falhas ou à inexperiência, esta razão contribui para que seja adotado em startups e pequenas organizações.

Para o **Nolan** foi escolhida a metodologia *Scrum*, um framework de gestão que permite autogerenciamento e adaptabilidade.

Baseada em cinco valores fundamentais, o Scrum se baseia completamente no ágil e trabalha lado-a-lado a fim de atingir o objetivo definido [34].

Compromisso com os objetivos, coragem de discutir as decisões, foco para apresentar resultados, abertura a novas ideias e respeito aos membros da equipe são os valores, ou pilares desta prática.

Com essas caraterísticas imbuídas na equipe, o ciclo de aprendizado contínuo tende a evoluir os envolvidos no ambiente de desenvolvimento de software.

No contexto do **Nolan**, a equipe é composta por apenas um desenvolvedor, condensando os papéis de *Scrum Master*, o responsável pelo andamento do projeto e por conduzir as reuniões, ou cerimônias, ágeis [35], *Product Owner*, o responsável por definir e idealizar o produto [36] e o desenvolvedor em si, responsável pela execução dos requisitos.

As entregas foram definidas semanalmente, no qual o desenvolvimento foi focado em concluir objetivos específicos, como a implantação de uma rota no servidor, ou a criação de uma interface para uma funcionalidade.

A partir da entrega anterior, um novo objetivo é traçado e o ciclo se repete. Este ciclo chama-se *Sprint* e pode ser definido livremente de acordo com a equipe, podendo durar uma semana, um mês ou quarenta e cinco dias. A Fig. 13 ilustra esse processo.

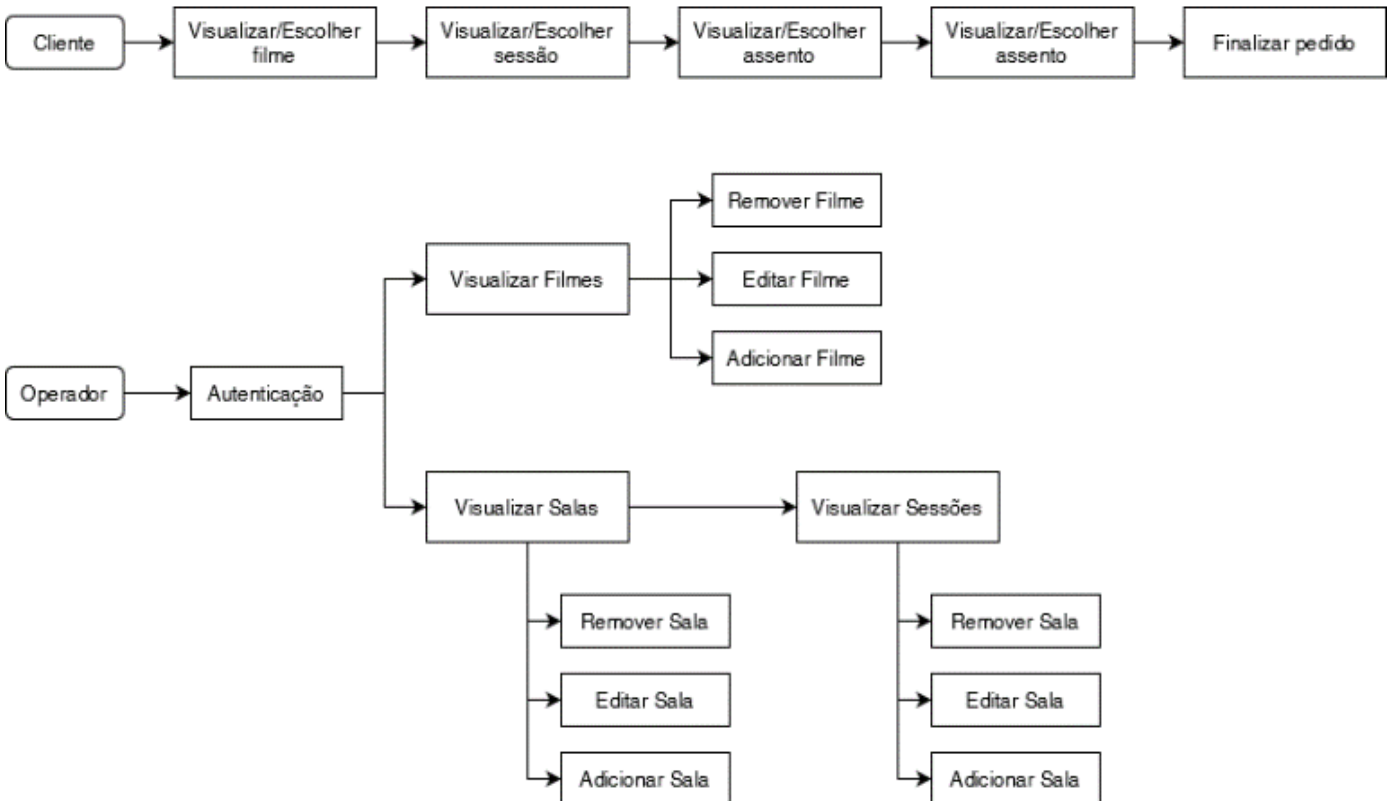

Fig. 13. Diagrama de casos de uso da aplicação.
Fonte: elaborado pelo autor.

B. Análise de Requisitos

A engenharia de requisitos é uma parte crucial no gerenciamento de projetos que se responsabiliza por coletar os dados iniciais para a criação de um produto de software [37].

É importante ter uma ideia do que o produto vai ser e quais públicos ele vai atender, qual problema ele irá resolver ou se é realmente viável, porém, a análise de requisitos irá demonstrar como essa ideia será concretizada.

Esta etapa de planejamento é fundamentalmente ligada ao usuário e como este irá interagir com o produto. São definidos como ele irá utilizá-lo, o que espera, suas necessidades e dificuldades.

Dados terão de ser levantados, quantitativa e detalhadamente, para fundamentar os pontos levantados. É visado reconhecer o problema a qual o produto tenta resolver, avaliá-lo juntamente com a solução proposta e modelar o projeto.

Os requisitos podem ser divididos em três tipos: de projeto, relativos à execução e planejamento. De produto, relativos ao que este necessita tecnicamente, como desempenho e segurança, e funcionais, um detalhamento dos serviços e funções do software.

A revisão é feita por ambas as partes, a equipe de desenvolvimento e o cliente. o Scrum, como neste projeto, as revisões são feitas continuamente ao fim de cada Sprint, para que o produto para ser melhorado e adaptado às necessidades do cliente de forma dinâmica, com transparência e colaboração em ambas as partes.

O **Nolan** tem dois tipos de usuários definidos: Clientes e Operadores. Para administrar um cinema, os funcionários da empresa devem ser capazes de registrar e alterar não só os filmes em cartaz, como as próprias salas e suas respectivas sessões.

Por tratarem informações administrativas, o módulo de funcionários necessita de autenticação de um usuário registrado para ser acessado. As informações sensíveis são salvas em banco com um registro da data de alteração e criação.

Esse módulo é o único capaz de alterar, inserir ou deletar dados salvos, pois o módulo de cliente exerce uma função mais direta.

A partir de *views* definidas, os clientes conseguem visualizar as informações dispostas e organizadas pelos operadores, sendo capazes de acessar os filmes disponíveis, visualizar as sessões disponíveis organizadas por dia e comprar um assento para estas.

A simplicidade acompanha um desafio: A interface deve ser amigável e simples, de modo que fique fácil e direta a compra de um ingresso, um requisito que é tanto funcional e de produto, por exemplo.

A UI/UX é um desafio também para o módulo de Administradores, uma vez que as informações vitais devem estar organizadas e dispostas de uma forma que seja fácil para o usuário saber o que está fazendo e não acabar removendo ou inserindo algo impróprio.

C. *Casos de Uso*

Ilustrado através de um diagrama, na Fig. 14, os casos de uso se referem a como os usuários irão interagir com o sistema. Através de *views* construídas em forma de interfaces gráficas, as informações serão distribuídas a cada tipo de usuário conforme sua necessidade.

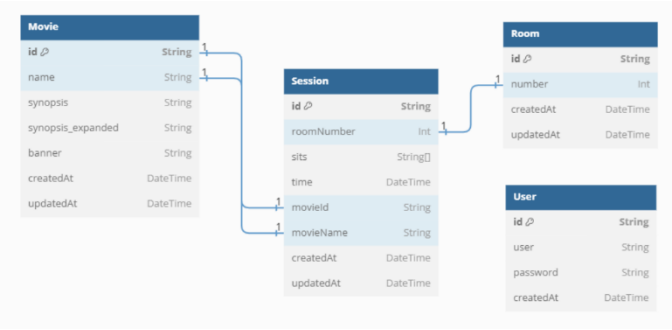

Fig. 14. Diagrama entidade relacionamento.
Fonte: elaborado pelo autor.

D. *Banco de Dados*

A estrutura do banco de dados é um requisito de extrema importância, uma vez que este define diretamente uma entidade e com quais características elas podem ser representadas, através de tabelas e colunas, respectivamente [38].

Para facilitar a criação da estrutura, foi inventado o Modelo Entidade Relacionamento, que é capaz de representar as entidades com seus atributos e relacionamentos de uma forma descritiva [39]. O MER utilizado pode ser visualizado na TABELA 1

TABELA I
DESCRIÇÃO TABELAS DO MER.

Tabela	Descrição
movie	Armazena os filmes e seus respectivos nomes, banners e sinopses. Os banners são imagens codificadas em base64.
room	Armazena as salas e seus respectivos números.
session	Armazena as sessões. Possui vínculo com as tabelas de filme e de sala através de seus respectivos ids, em relações 1:N.
user	Armazena os dados de usuário, como seu nome e senha.

Fonte: elaborado pelo autor.

Graficamente, o MER é representado por um diagrama chamado Diagrama ER ou DER, onde é possível visualizar, através de um padrão de formas, mais facilmente a estrutura do banco, porém, ao contrário do MER, não possui uma descrição. Por esse motivo, os dois modelos comumente são desenvolvidos lado-a-lado. O DER pode ser visualizado na Fig. 14.

V. APRESENTAÇÃO DA APLICAÇÃO DESENVOLVIDA

O projeto, além do gerenciamento de salas de cinema, também foi pensado para fornecer uma opção de venda de ingresso para clientes, já que poderia ser utilizado como um pacote inteiro capaz de operar um negócio em todas as frentes.

Ele visa fornecer uma experiência completa de gestão de salas de cinema e filmes, como também a oportunidade de o usuário reservar seu assento em uma sessão por meio de um *self-checkout* disponibilizado em um totem, ou, pela bilheteria, como tradicionalmente ocorre.

As interfaces foram construídas seguindo o modelo artístico minimalista, no qual pouca informação é mostrada de forma direta [40].

Esta arquitetura dispõe apenas as informações necessárias, explorando a utilização de conceitos como espaço negativo, com o foco em um visual limpo e simples, eliminando distrações para focar no que é importante.

O interesse por saber mais, representado pelo clique em um componente, recompensa o usuário com a informação completa.

Os módulos descritos a seguir podem ser executados singularmente através de *scripts* fornecidos em seus códigos-fontes.

A implantação desse projeto pode ser feita tanto em um servidor centralizado, para uma companhia, ou em nuvem, para que não haja a necessidade de gasto com equipamentos físicos no local.

Apesar da utilização do Postgre para o **Nolan**, ele pode ser adaptado a qualquer tipo de banco suportado pelo Prisma, sendo necessária a configuração prévia deste para a instalação e depois a execução das *migrations*.

Independentemente do modo que for implantado, o **Nolan** irá funcionar normalmente através dos navegadores compatíveis e poderá ser acessado pelo IP e porta a qual estiver hospedado.

A. *Administradores*

Este usuário alvo será o administrador do sistema. Através de um sistema de autenticação simples utilizando JWT (**JSON Web Token**), os usuários podem fazer *login* neste módulo.

O JWT é um token expirável de tempo pré-determinado e variável, que armazena informações referentes à sessão atual de um usuário em determinado sistema [41].

Na tela inicial é possível fazer o *login* ou a criação de usuário, em uma interface simples e direta. As senhas são criptografadas utilizando a biblioteca *bcrypt*, que implementa a técnica homônima no JavaScript [42].

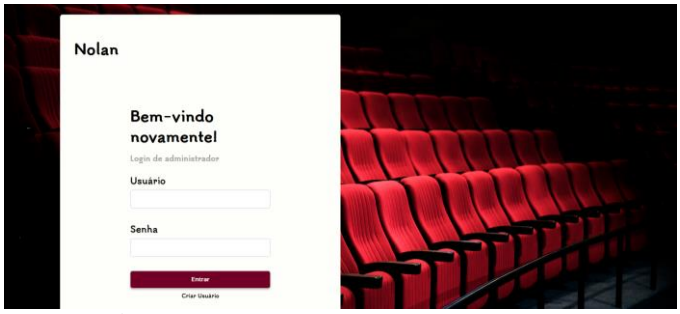

Fig. 15. Interface inicial.
Fonte: elaborado pelo autor.

Ao fazer um *login* bem-sucedido, o *token* é armazenado em formato de *Cookie* no navegador, podendo ser acessado a qualquer momento por meio de um serviço criado dentro aplicação.

Uma única página é mostrada com todas as funcionalidades, conforme a Fig. 15, divididas por abas que fazem referência aos filmes e às salas do cinema. O corpo da página possui altura fixa, apenas seu conteúdo é alterado conforme a interação com o usuário.

O plano de fundo, um cenário de cinema, visa refletir, junto com a paleta de cores, a ambientação de um cinema, com um vermelho mais intenso e tons de cinza e preto para destacar os elementos necessários.

Há uma terceira aba chamada de Parâmetros, salva para uma implementação futura de valores genéricos que poderão ser utilizados globalmente pelos dois módulos, como o preço de um ingresso ou a quantidade máxima de sessões por dia.

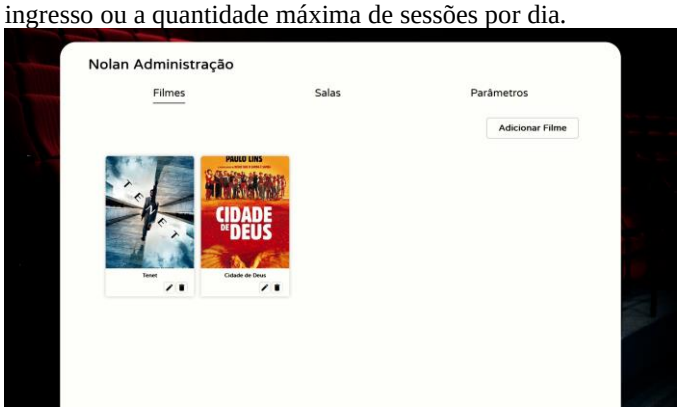

Fig. 16. Interface de gerenciamento de filmes.
Fonte: elaborado pelo autor.

Os filmes, na primeira aba, são exibidos em um componente estilo *card*, como mostrado na Fig. 16, com apenas seu nome e ícones que representam a ação de editar e excluir, um padrão que será bastante seguido no projeto. Destaca-se também o botão de adicionar um novo filme na parte superior.

O clicar nos botões de adicionar, editar ou deletar, um pop-up é exibido com um formulário para incluir ou alterar as informações necessárias, com opção de upload de arquivos para o banner do filme. Ele pode ser visualizado na Fig. 17.

Assim como todos os *pop-ups* que serão descritos nesta seção, esta conta com validações do campo, pois foi construído como um formulário HTML.

Não é possível concluir a ação enquanto todos os campos estiverem preenchidos e, especificamente no domínio dos filmes, o *banner* enviado pode ter no máximo 2MB, para evitar que registros muito grandes sejam armazenados em banco.

No caso da exclusão, o pop-up serve como uma confirmação para saber se o usuário tem certeza de que deseja fazer essa ação.

Após preenchidos os dados necessários, uma requisição é enviada deste modulo ao *back-end* através de um *endpoint* relativo à ação.

Os dados são alterados dinamicamente. Isto significa que não há a necessidade de atualizar a tela para visualizar novas informações ou edições feitas.

Esse mecanismo é feito através do *EventEmitter* do Angular, capaz de propagar sinais entre componentes e enviar informações sobre alterações de estados, como também a alteração em si, para manter todos eles em constante atualização e integração [43].

Os eventos podem ser diversos, como o pressionar de uma tecla ou a movimentação do mouse. Nesse caso o evento é emitido ao apertar o botão para confirmar a criação do item. O componente pai está sempre "ouvindo" por mudanças nos componentes filhos.

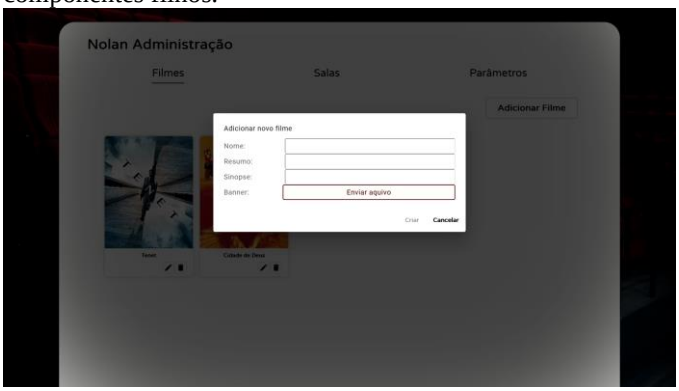

Fig. 17. Pop-up para criação de filmes.
Fonte: elaborado pelo autor.

A segunda aba agrupa todas as sessões em suas respectivas salas. Novamente há o botão para adicionar um novo item, neste caso, uma sala.

Não é possível alterar uma sala já criada, pois esta é apenas um número único que serve, teoricamente, como uma forma de agrupar sessões.

Através de um componente acordeão, as sessões disponíveis, se houverem, são exibidas. Nesta interface não há filtros de pesquisa a não ser a data de criação, para que seja mais fácil de encontrar os dados.

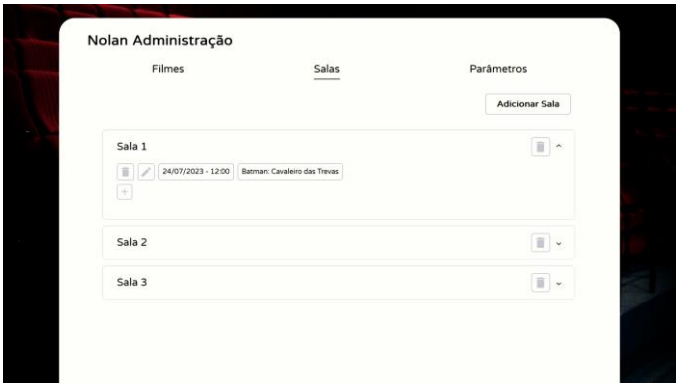

Fig. 18. Interface de gerenciamento de sessões.
Fonte: elaborado pelo autor.

Os containers com as sessões contam com botões de edição e remoção, com o mesmo desenho que os dos filmes, más exibidos em uma linha, ao invés um card, como na Fig. 19.

Fig. 19. Interface inicial do módulo de consumidor.
Fonte: elaborado pelo autor.

Essa escolha de design se deu por causa do volume de dados, uma vez que uma sala pode conter um número muito alto de sessões programadas para a quantidade de dias que os administradores desejarem.

Não há quantidade máxima para o agendamento de sessões por sala, a tela irá exibir todas elas ordenadas pela data e horário.

Os *pop-ups* de sessão seguem o mesmo padrão de design e funcionamento dos filmes e salas, no qual o usuário informa os dados necessários de cada entidade e confirma a operação.

A terceira aba, de parâmetros, não teve sua implementação concluída, estando sem conteúdo visível para o usuário. A ideia para sua interface consiste em uma simples tabela com três colunas, de chave, valor e descrição, e botões de ação como na parte de sessões, apelando para a simplicidade.

Em futuras atualizações, planeja-se implementar um sistema de usuários mais complexo, para distinguir um usuário administrador de um usuário operador. Desta forma pode-se limitar usuário a apenas visualizar determinadas informações ou a alterações apenas seções permitidas. Por exemplo, apenas um gestor poderia alterar a precificação dos ingressos.

Justamente por não haver esta distinção de usuário e de não haver, por enquanto, o registro de alterações por usuário, não foi implementada uma função de *logout*. A sessão é automaticamente encerrada quando o *token* expira, em um tempo padrão de 6 horas.

Sendo construído pensando para ser utilizado em desktops, o módulo de administrador poderá ser expandido para o mobile, caso a necessidade seja levantada, uma vez que em situações de longas filas, é interessante adiantar a venda de ingressos na própria fila.

Por ser um sistema completamente Web, basta que um navegador consiga acessá-lo e que haja portabilidade visual para o dispositivo.

B. Cliente

O usuário alvo deste módulo é o próprio consumidor do cinema. Ao contrário do administrador, para acessar este módulo não é necessária autenticação ou a obrigatoriedade da identificação de usuário.

O padrão de cores e corpo foi mantido, com o mesmo plano de fundo de cinema e paleta de cores com tons de vermelho e preto.

O invés de um sistema de abas, foi optado por um componente do tipo *stepper*. Desta forma, a interface será capaz de guiar o usuário através de um sistema que determina o que ele deve fazer de cada vez, em uma ordem específica que irá se repetir toda vez que for fazer um processo de pedido.

O *stepper* também ajuda a organizar o código, uma vez que é possível visualizar as etapas específicas dentro do *template* (o arquivo HTML) do componente e escrever a lógica de programação em função desse ordenamento.

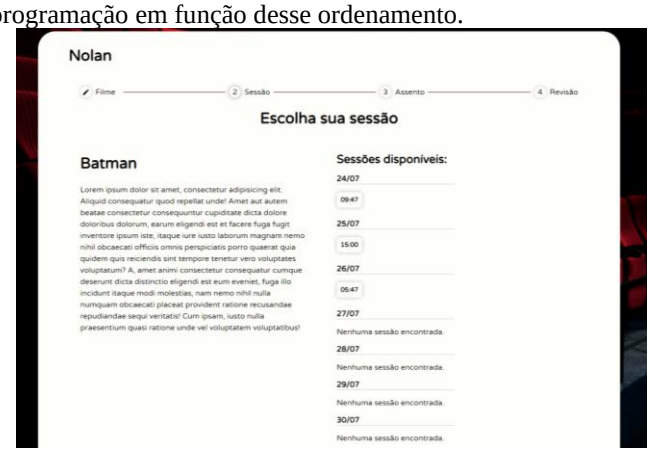

Fig. 20. Seleção de sessões para o filme escolhido.
Fonte: elaborado pelo autor.

Logo na tela inicial, ou primeiro passo, o consumidor irá se deparar com as opções de filme disponíveis, como ilustrado na Fig. 20. Ao clicar em qualquer um deles, o usuário será levado ao segundo passo.

Escolhido o filme, a tela seguinte exibe informações mais detalhadas sobre este, como sua sinopse completa. Para atualizações futuras, planeja-se adicionar mais elementos descritivos para os filmes, como duração, gênero, elenco, dentre outros.

À direita, ocupando a outra metade horizontal da tela, estão as sessões disponíveis, organizadas por dia, como mostra a Fig. 20. São exibidos os próximos sete dias a partir do dia atual, com seus respectivos horários para o filme escolhido. Ao clicar em uma das sessões, o segundo passo é concluído, dando lugar ao terceiro. Nele é exibido um *grid* 10x10 de cadeiras, simulando uma sala de cinema.

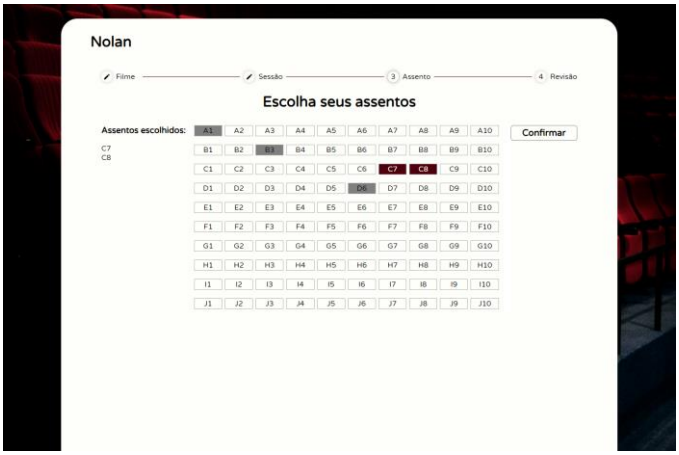

Fig. 21. Seleção de assentos para a sessão escolhida.
Fonte: elaborado pelo autor.

Essas cadeiras, organizadas em linhas ordenadas alfabeticamente de A até J e colunas enumeradas de 1 a 10, representam uma sala de cinema, onde a fileira A está mais próxima da tela e a J está mais distante. O mapa de cadeiras pode ser visualizado na Fig. 21.

Os locais ocupados previamente são marcados com um plano de fundo de cor cinza, enquanto os disponíveis estão na cor branca, indicando transparência.

Ao selecionar um assento, sua identificação é adicionada a uma lista à esquerda da tela, enquanto seu desenho é preenchido com a cor vermelha.

Ao clicar em no botão para avançar, o último passo é exibido, o de conclusão de pedido, ilustrado na Fig 22. Como ainda não foi implementada a opção de precificação, apenas as informações mais básicas para o funcionamento do projeto são exibidas.

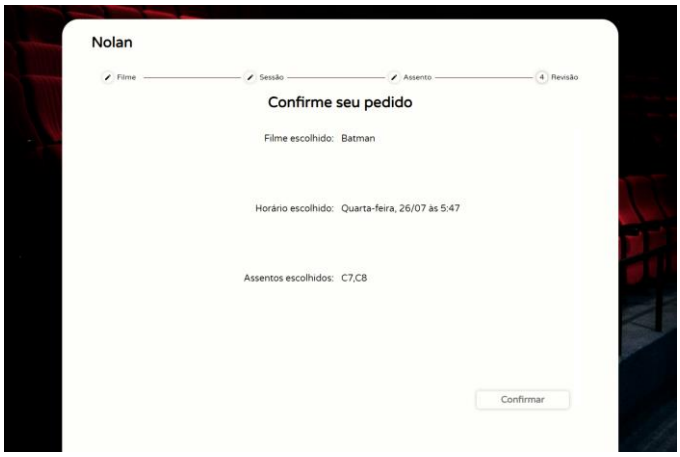

Fig. 22. Revisão e conclusão de pedido.
Fonte: elaborado pelo autor.

Futuramente, planeja-se colocar dois passos entre o último e o penúltimo. O primeiro para que o usuário possa selecionar o tipo do seu ingresso (inteira ou meia), de acordo com o assento escolhido e visualizar não só os preços individuais, como o total.

O segundo passo será exclusivo para o pagamento, com uma API externa integrada ao sistema, capaz de suportar certas formas de pagamento, como crédito, débito e PIX.

C. *Servidor*

Conforme foi descrito anteriormente, as funções do *back-end* estão relacionadas ao tratamento e armazenamento de dados, não sendo necessariamente uma aplicação a qual os usuários irão interagir diretamente.

Os domínios da API foram separados de acordo com as entidades descritas no MER, com *endpoints* nomeados de acordo com seus nomes.

Em cada *endpoint* desses existe um padrão de rotas baseado no modelo CRUD, abreviação de *Create*, *Read*, *Update* e *Delete* [44].

Foi criada uma interface genérica chamada de *BaseCrudRepository*, mostrada na Fig. 23, que pode ser instanciada com um tipo genérico para que a classe que a implementa tenha de implementar também as funções básicas que compõem o CRUD.

```
export interface BaseCrudRepository<T> {
    list: () ⇒ Promise<T[]>;
    searchById: (id: string) ⇒ Promise<T>;
    create: (item: any) ⇒ Promise<T>;
    update: (item: any) ⇒ Promise<T>;
    delete: (id: string) ⇒ Promise<T>;
}
```

Fig. 23. Interface básica de CRUD.
Fonte: elaborado pelo autor.

O domínio de usuário, por contar com funções diferentes, não implementa esta interface. Este utiliza apenas três rotas simples, de criação e pesquisa de usuário e login, futuramente esta segunda irá ser separada em um domínio de autenticação à parte, quando a função de *logoff* for implementada.

As rotas estão protegidas pelo filtro de autenticação que verifica a presença do *Bearer token* no cabeçalho da requisição HTTP que chega na API.

As respostas de rotas as rotas são as chamadas *Promises*, objetos nativos do JavaScript que representam operações assíncronas que ainda não foram finalizadas, como uma promessa de que aquele dado será ele mesmo no final das contas [45].

Ao serem resolvidas, as *Promises* são puramente respostas HTTP com cabeçalho e corpo contendo um JSON específico para cada rota representando o dado requisitado.

Para evitar lidar com requisições assíncronas e ter de adaptar o código inteiro para isso, o Angular utiliza, como já citado anteriormente, os *Observables*.

Estes objetos, mais poderosos, possuem a capacidade de lidar com as requisições antes que elas sejam resolvidas, através dos operadores da mesma biblioteca a qual pertencem, o RxJS.

Com eles, é possível cancelar uma requisição, tentar novamente dispará-la ou captar erros de resposta antes de tratar o conteúdo dela em si, isto torna o código não só mais legível para manutenção como mais fácil de ser alterado para tratar determinados cenários indesejados ou não.

VI. Testes e Resultados

Os testes de software formam uma etapa crucial no ciclo de desenvolvimento [46]. Construir ou manter um projeto que mantenha um determinado nível de qualidade é um desafio constante para qualquer produto de mercado.

É através desta etapa que o software é avaliado a fim de evidenciar se seu funcionamento atende aos requisitos ou ao planejamento proposto, podendo ter impactos diretos no cronograma e no gerenciamento do projeto como um todo.

No ramo de desenvolvimento, o próprio desenvolvedor pode ser responsável pelos testes, porém, existem profissionais cuja carreira é dedicada apenas a isso [47], sendo até mesmo desnecessário, em certos casos, o conhecimento de programação.

A necessidade de um profissional adequado para os testes se dá por conta da complexidade do processo. Existem mais de 15 tipos de testes diferentes, dentre manuais e automatizados, dependendo do tipo de software que está sendo avaliado.

Neste projeto foram utilizados os tipos de testes descritos nesta seção para encontrar diversos tipos de erros que foram corrigidos ao longo do desenvolvimento.

A. *Teste Funcional*

Este tipo de teste visa avaliar o produto sob uma ótica de caixa-preta, no qual o testador não sabe o que ocorre dentro do código em si, apenas sabe que dado determinado *input*, o código deve dar um *output* equivalente [48].

Os dados de saída devem obedecer às regras de negócio estabelecidas durante a fase de levantamento de requisitos. Por exemplo, um agendamento de sessão deve ser marcado apenas pelas horas e minutos, sempre deixando em zero os segundos e milésimos, para evitar problemas de comparação de horário em outras situações.

Este tipo de teste foi utilizado com mais ênfase no módulo de servidor. Nos módulos de front-end existiram poucas etapas que não dependiam da API para funcionar.

Dentre os problemas encontrados, destacam-se seguintes:

1. Erros relacionados a um *schema* mal formulado retornam uma resposta cujo corpo contém um objeto de erro com a versão da ORM, sem descrição;
2. Erros em que *endpoints* não estavam levando à rota correta devido à hierarquia sequencial de como elas devem ser declaradas no código;
3. Erros ao acessar as rotas devido a falhas na implementação da criptografia de senha, gerando uma comparação errada de valores;
4. Falha na autenticação devido a erros na implementação do token JWT;
5. Falhas na busca de dados de sessão devido à formatação da data discrepante entre os dados enviados e os que eram salvos no banco;
6. Falha em bloquear a escolha de assentos previamente reservados em novas reservas;
7. Falha em agrupar as sessões por dia, fazendo com que não sejam exibidas;
8. Falhas em aberturas de *pop-ups* com dados previamente carregados por componentes pais;
9. Falha no roteamento de página de login para a página inicial, causando um loop infinito devido a um erro no *path*.

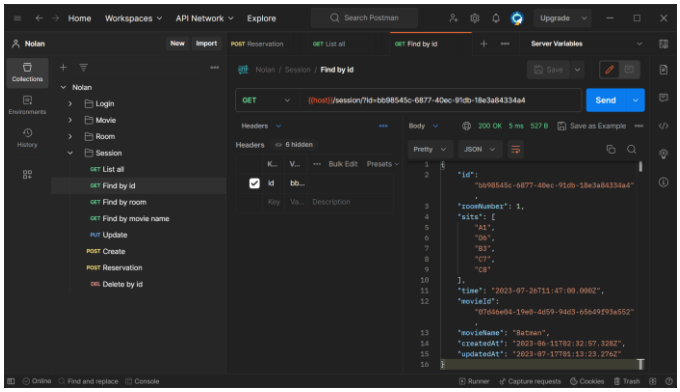

Fig. 24. Uso do Postman para testar o fluxo de dados.
Fonte: elaborado pelo autor.

Para realizar os testes de servidor, foram utilizados o DBeaver e o Postman, a fim de assegurar que os dados estavam sendo gravados e buscados do banco de dados com as devidas regras de negócio aplicadas.

As requisições enviadas e respostas recebidas podem ser visualizadas na Fig 24. Dados adicionais como erros e suas causas são exibidos na seção de resposta ou no console do programa.

No caso do *front-end*, a ferramenta mais utilizada foi o DevTools do Google Chrome, no qual foi possível verificar a estilização dos componentes e simular interações de usuário como passar o mouse por cima, clicar ou focalizá-lo. O painel da ferramenta pode ser visualizado na Fig. 25.

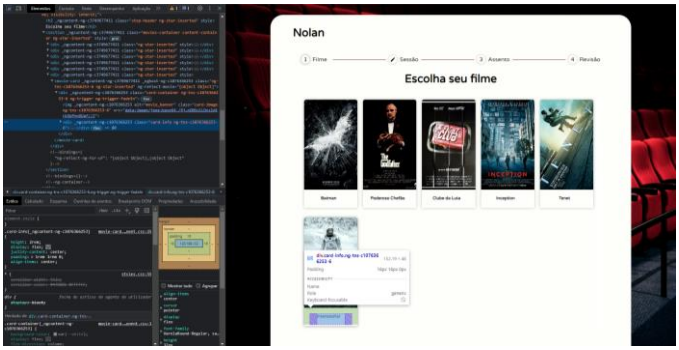

Fig. 25. Uso do DevTools para testar o front-end.
Fonte: elaborado pelo autor.

B. *Teste de Integração*

Os testes de integração, como o nome sugerem, servem para avaliar o comportamento de diferentes partes do projeto que se integram para uma funcionalidade comum [49].

Tendo em vista que os módulos de *front-end* dependem completamente do *back-end* para funcionarem, a maioria dos testes neles foi de integração.

A maioria dos erros encontrados reflete uma divergência entre os dados que são esperados e os recebidos, em ambas as frentes.

Dentre os mais impactantes, destacam-se os seguintes:

1. Erro no formato de envio do token de autenticação, fazendo com que o as rotas, exceto de login, não pudessem ser acessadas;
2. Erros ao enviar dados do tipo *string* que deveriam ser do tipo *number*, dentro do corpo da requisição;
3. Erros ao enviar tipos de data sem segundos e milissegundos, enquanto a API requer este tipo de detalhamento;
4. Erros na exibição dos horários das sessões devido à diferença de uso do objeto de data entre aplicações, causando diferenças de fuso horário;
5. Falhas ao editar sessões pela utilização de parâmetros na URL ao invés de *queries*.

Nesta fase o Postman também foi utilizado para garantir que os dados recebidos pelo *front-end* estavam sendo tratados corretamente e não indevidamente alterados, ao compará-los com a resposta obtida por ele.

C. Teste de Regressão

O último tipo de teste utilizado é o mais extenso e trabalhoso. Ele procura avaliar o projeto como um todo, simulando a utilização de um usuário final [50].

Nesse aspecto, todos os módulos são colocados à prova para executarem todas as suas funções para que seja possível verificar se o software está apto para utilização.

Com os três módulos sendo executados simultaneamente, foram feitos todos os processos envolvendo os todos os domínios que o projeto abrange.

Foram criados novos filmes, salas e sessões a partir do módulo de administrador, enquanto no de cliente, foi realizado todo o processo de reserva de assento.

Enquanto esses processos foram realizados, o banco de dados foi acompanhado, através do DBeaver para verificar as alterações dos dados e garantir sua integridade.

As situações em que os erros foram esperados foram tratados com mensagens de erro para alertar ao usuário do problema, como, por exemplo, um login malsucedido por credenciais inválidas ou a inserção de um novo filme sem nome.

VII. Conclusões

A proposta deste projeto é praticar o desenvolvimento de um Sistema Web utilizando tecnologias modernas e presentes no mercado e metodologias ágeis.

O sistema possui uma interface intuitiva e de fácil usabilidade, evitando a necessidade de treinamentos extensos e custosos.

Sua performance e modelagem de dados dispensa a necessidade de um servidor robusto para que seja hospedado, gerando economia para hospedagem em nuvem e local.

Ao ser hospedado em nuvem, entretanto, haverá a necessidade de existir uma conexão com a internet, enquanto a implantação local permite que ele opere apenas na rede local.

A acessibilidade, por ser um sistema web, permite que seja acessado por qualquer sistema operacional, em qualquer dispositivo, que possa ter instalado um navegador de internet.

Para pequenos negócios que não possuem grandes orçamentos ou estruturas, o **Nolan** pode ser uma solução viável pela simplicidade e baixo custo.

A. Considerações Finais

Tendo em vista os resultados alcançados, é possível concluir que o projeto é funcional em sua versão atual de protótipo.

Para ser comercializado, entretanto, ainda necessita de atualizações que possam englobar um negócio real, principalmente no âmbito da venda, a qual precisa de uma série de regras de precificação e integrações com sistemas externos de pagamento e emissão de cupom fiscal, por exemplo.

Referências Bibliográficas

[1] UCEFF, “Teoria X prática: entenda a importância de aliar esses dois pontos,” UCEFF, 19 02 2021. [Online]. Available: https://blog.uceff.edu.br/teoria-x-pratica/. [Acesso em 22 07 2023].

[2] 33giga, “Como anda o mercado de trabalho para o programador?,” 33giga, 15 09 2022. [Online]. Available: https://33giga.com.br/mercado-de-trabalho-programador/. [Acesso em 22 07 2023].

[3] JetBrains, “WebStorm, O IDE JavaScript mais inteligente,” JetBrains, [Online]. Available: https://www.jetbrains.com/pt-br/webstorm/. [Acesso em 10 07 2023].

[4] Postman, “Postman API Platform,” Postman, [Online]. Available: https://www.postman.com/. [Acesso em 22 07 2023].

[5] Dbeaver, “Universal Database Tool,” Dbeaver, [Online]. Available: https://dbeaver.io/. [Acesso em 22 07 2023].

[6] Synopsis, “Compare Repositories,” Synopsis, [Online]. Available: https://openhub.net/repositories/compare. [Acesso em 24 07 2023].

[7] Git, “Git,” [Online]. Available: https://git-scm.com/. [Acesso em 24 07 2023].

[8] GitHub, “Olá, Mundo,” GitHub, [Online]. Available: https://docs.github.com/pt/get-started/quickstart/hello-world. [Acesso em 22 07 2023].

[9] Google, “Chrome DevTools,” Google, [Online]. Available: https://developer.chrome.com/docs/devtools/. [Acesso em 24 07 2023].

[10] C. E., “O Que é JavaScript,” Hostinger, 26 05 2023. [Online]. Available: https://www.hostinger.com.br/tutoriais/o-que-e-javascript#:~:text=Ele%20criou%20a%20linguagem%20quando,e%20alertas%20muito%20mais%20simples.. [Acesso em 24 07 2023].

[11] O. Foundation, “About Node.js,” OpenJS Foundation, [Online]. Available: https://nodejs.org/en/about. [Acesso em 24 07 2023].

[12] Krystal, “The Beginner’s Guide to Backend Development (2022 Guide),” LearToCodeWithMe, 02 02 2022. [Online]. Available: https://learntocodewith.me/posts/backend-development/. [Acesso em 24 07 2023].

[13] AWS, “O que é uma API?,” Amazon AWS, [Online]. Available: https://aws.amazon.com/pt/what-is/api/#:~:text=API%20significa%20Application%20Programming%20Interface,de%20servi%C3%A7o%20entre%20duas%20aplica%C3%A7%C3%B5es.. [Acesso em 24 07 2023].

[14] Higor, “Introdução ao Padrão MVC,” DevMedia, 2013. [Online]. Available: https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308.

[15] Zod, “Zod,” Colinhacks, [Online]. Available: https://zod.dev/?id=introduction. [Acesso em 24 07 2023].

[16] TypeScript, “What is TypeScript?,” Microsoft, [Online]. Available: https://www.typescriptlang.org/. [Acesso em 22 07 2023].

[17] Express, “Express - framework de aplicativo da web Node.js,” Fundação Node.js, [Online]. Available: https://expressjs.com/pt-br/. [Acesso em 24 07 2023].

[18] Mozilla, “Cross-Origin Resource Sharing (CORS),” Mozilla, 19 07 2023. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CORS. [Acesso em 24 07 2023].

[19] W. Willians, “O que é DDD – Domain Driven Design,” Fullcycle, 19 08 2019. [Online]. Available: https://fullcycle.com.br/domain-driven-design/. [Acesso em 24 07 2023].

[20] Cadu, “ORM : Object Relational Mapper,” DevMedia, 2011. [Online]. Available: https://www.devmedia.com.br/orm-object-relational-mapper/19056. [Acesso em 24 07 2023].

[21] Prisma, “Prisma Client, Intuitive Database Client for TypeScript and Node.js,” Prisma, [Online]. Available: https://www.prisma.io/client. [Acesso em 22 07 2023].

[22] Prisma, “Prisma Migrate,” Prisma, [Online]. Available: https://www.prisma.io/docs/concepts/components/prisma-migrate. [Acesso em 22 07 2023].

[23] B. Kriger, “Front end: o que é, para que serve, como aprender essa especialidade!,” Kenzie, 12 06 2023. [Online]. Available: https://kenzie.com.br/blog/front-end/. [Acesso em 24 07 2023].

[24] brain, “UX e UI design: você conhece a diferença entre os dois conceitos?,” brain, 25 05 2020. [Online]. Available: https://inovacaobrain.com.br/ux-e-ui-design/. [Acesso em 24 07 2023].

[25] Mozilla, “HTML: Linguagem de Marcação de Hipertexto,” Mozilla, 18 07 2023. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/HTML. [Acesso em 24 07 2023].

[26] Mozilla, “CSS,” Mozilla, 06 11 2022. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/CSS. [Acesso em 24 07 2023].

[27] Mozilla, “JavaScript,” Mozilla, [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. [Acesso em 22 07 2023].

[28] Angular, “What is Angular?,” Google, 28 02 2022. [Online]. Available: https://angular.io/guide/what-is-angular. [Acesso em 22 07 2023].

[29] W. Nascimento, “Entendendo RxJS Observable com Angular,” Medium, 04 01 2017. [Online]. Available: https://medium.com/tableless/entendendo-rxjs-observable-com-angular-6f607a9a6a00. [Acesso em 22 07 2023].

[30] freeCodeCamp, “Como usar e criar diretivas personalizadas no Angular,” freeCodeCamp, 12 10 2022. [Online]. Available: https://www.freecodecamp.org/portuguese/news/como-usar-e-criar-diretivas-personalizadas-no-angular/. [Acesso em 25 07 2023].

[31] PostgreSQL, “About,” PostgreSQL, [Online]. Available: https://www.postgresql.org/about/. [Acesso em 22 07 2023].

[32] J. Santaella, “Metodologia de projetos: o que é e como escolher a ideal?,” José Santaella, 18 11 2022. [Online]. Available: https://www.euax.com.br/2022/11/metodologia-de-projetos/. [Acesso em 22 07 2029].

[33] G. Losnak, “Metodologia Ágil - O que é?,” Alura, 04 07 2023. [Online]. Available: https://www.alura.com.br/artigos/o-que-e-metodologia-agil?gclid=Cj0KCQjw5f2lBhCkARIsAHeTvlhyq4Sx5FttNzm0YBg07xcqSrIqoBsUnr9gJCdajV0nlOEryl--PKUaAguWEALw_wcB. [Acesso em 25 07 2023].

[34] Amazon, “O que é o Scrum?,” Amazon AWS, [Online]. Available: https://aws.amazon.com/pt/what-is/scrum/. [Acesso em 22 07 2023].

[35] A. Ferreira, “Scrum Master: quem é e o que faz,” K21, 13 02 2019. [Online]. Available: https://k21.global/br/blog/quem-e-scrum-master?gad=1&gclid=Cj0KCQjw5f2lBhCkARIsAHeTvljwFodDA0jCpWKrmTJCfE8R40oXZ5SWOohP199QBm7_TXBSR442Be8aAhCTEALw_wcB. [Acesso em 25 07 2023].

[36] R. Vieira, “O que é um Product Owner?,” Medium, 27 01 2017. [Online]. Available: https://medium.com/produto-di%C3%A1rio/o-que-%C3%A9-um-product-owner-c44bb29a9f66. [Acesso em 25 07 2023].

[37] A. P. Quiterio, “Análise de Requisitos,” InfoEscola, [Online]. Available: Análise de Requisitos. [Acesso em 20 07 2023].

[38] V. E. S. Souza, “Banco de Dados – Análise de Requisitos,” UFES, 05 2014. [Online]. Available: http://www.inf.ufes.br/~vitorsouza/archive/2020/wp-content/uploads/academia-br-bdep-requisitos.pdf. [Acesso em 25 07 2023].

[39] Joel, “MER e DER: Modelagem de Bancos de Dados,” DevMedia, 2014. [Online]. Available: https://www.devmedia.com.br/mer-e-der-modelagem-de-bancos-de-dados/14332. [Acesso em 22 07 2023].

[40] L. Moreno, “Guide: The Art of Minimalism in UI Design,” Sympli, 10 02 2021. [Online]. Available: https://sympli.io/blog/the-art-of-minimalism-in-ui-design. [Acesso em 22 07 2023].

[41] JWT.io, “Introduction to JSON Web Tokens,” Auth0, [Online]. Available: https://jwt.io/introduction. [Acesso em 25 07 2023].

[42] Kerlyn, “Uma breve introdução sobre BCrypt,” Medium, 10 12 2019. [Online]. Available: https://medium.com/reprogramabr/uma-breve-introdu%C3%A7%C3%A3o-sobre-bcrypt-f2fad91a7420. [Acesso em 22 07 2023].

[43] Angular, “EventEmitter,” Google, [Online]. Available: https://angular.io/api/core/EventEmitter. [Acesso em 25 07 2023].

[44] Mozilla, “CRUD,” Mozilla, 09 07 2023. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Glossary/CRUD. [Acesso em 22 07 2023].

[45] Mozilla, “Promise,” Mozilla, 19 07 2023. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise. [Acesso em 22 07 2023].

[46] Objective, “Testes de Software: Definição, Conceitos e Exemplos,” Objective, 09 06 2022. [Online]. Available: https://www.objective.com.br/insights/testes-de-software/. [Acesso em 22 07 2023].

[47] J. P. Soares, “Quality Assurance (QA) e sua importância no desenvolvimento de software,” Treinaweb, 23 10 2020. [Online]. Available: https://www.treinaweb.com.br/blog/quality-assurance-qa-e-sua-importancia-no-desenvolvimento-de-software. [Acesso em 22 07 2023].

[48] DevMedia, “Testes funcionais de software,” DevMedia, 2012. [Online]. Available: https://www.devmedia.com.br/testes-funcionais-de-software/23565. [Acesso em 22 07 2023].

[49] D. Kriger, “O que é teste de integração e quais são os tipos de teste?,” Kenzie, 08 10 2021. [Online]. Available: https://kenzie.com.br/blog/teste-de-integracao/. [Acesso em 25 07 2023].

[50] DevMedia, “Teste de Regressão,” DevMedia, 2011. [Online]. Available: Teste de Regressão. [Acesso em 22 07 2023].

[51] A. Nitahara, “Estudo mostra que pandemia intensificou uso das tecnologias digitais,” Agência Brasil, 25 11 2021. [Online]. Available: https://agenciabrasil.ebc.com.br/geral/noticia/2021-11/estudo-mostra-que-pandemia-intensificou-uso-das-tecnologias-digitais. [Acesso em 22 07 2023].

Marcelo Stehling de Castro graduou-se em Engenharia Elétrica pela Universidade Federal de Juiz de Fora (1992), com mestrado em Engenharia Elétrica pela Universidade Estadual de Campinas (1995) e doutorado em Engenharia Elétrica pela UnB (2010). Docente Associado da Universidade Federal de Goiás, tendo ingressado em 1996. Possui experiência na área de engenharia de redes, computação paralela e distribuída, comunicações óticas e tecnologias alternativas de última milha (BPL, ZigBee, Wi-Fi). Desenvolve pesquisas em temas que incluem redes de comunicação (5G, Gigabit Wi-Fi), *Smart Grids*, *Smart Cities*, *Smart Campus*, tecnologia da informação e comunicação e gestão aplicadas a projetos de redes de telecomunicações.

Caio Vinícius Pinto Zirretta é graduando em Engenharia de Computação na Universidade Federal de Goiás. Atualmente é desenvolvedor de software na TOTVS, em Goiânia. Tendo experiência com desenvolvimento de projetos de ERP no ramo de varejo e distribuição e Sistemas Web.