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

Diego Amancio Pereira

BooKollection

CAMPINA GRANDE - PB
2022

Diego Amancio Pereira

BooKollection

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

Orientador: Professor Dr. João Arthur Brunet Monteiro.

CAMPINA GRANDE - PB
2022

Diego Amancio Pereira

BooKollection

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

BANCA EXAMINADORA:

Professor Dr. João Arthur Brunet Monteiro

Orientador – UASC/CEEI/UFCG

Professor Dr. Wilkerson de Lucena Andrade

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de Setembro de 2022.

CAMPINA GRANDE - PB

RESUMO

Os sites e aplicativos direcionados a colecionadores de obras literárias são importantes, pois possuem funcionalidades capazes de organizar coleções, selecionar a ordem de leitura e opinar sobre os títulos presentes na plataforma. Todavia, faltam funções importantes para o usuário nas plataformas existentes acerca da sua coleção, por exemplo, as informações financeiras acerca do quanto foi investido, conservação dos livros, detalhamento das obras, entre outras. Diante do constante aumento no valor dos livros e queda na sua qualidade [26], nota-se uma decisão mais seletiva por parte do usuário, que por sua vez, demandará de mais tempo, devido à escassez de informações mais específicas acerca dos títulos.

Este trabalho teve como objetivo criar uma plataforma voltada para colecionadores de obras literárias, trazendo diferenciais, tais como um comparador de livros com informações sobre a obra (material, quantidade de páginas, preço, resumo), notas baseadas na opinião dos usuários na plataforma (custo benefício e dificuldade de adquirir um título ou coleção) e um leitor de código de barras que, ao escanear um livro irá fornecer um identificador, o International Standard Book Number (ISBN). Isto possibilita ao usuário realizar uma busca da obra no site e obter dados sobre os títulos sem a necessidade de tê-los de forma manual.

BooKollection

Diego Amancio Pereira
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
diego.pereira@ccc.ufcg.edu.br

João Arthur Brunet Monteiro
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
joao.arthur@computacao.ufcg.edu.br

Resumo

Os sites e aplicativos direcionados a colecionadores de obras literárias são importantes, pois possuem funcionalidades capazes de organizar coleções, selecionar a ordem de leitura e opinar sobre os títulos presentes na plataforma. Todavia, faltam funções importantes para o usuário nas plataformas existentes acerca da sua coleção, por exemplo, as informações financeiras acerca do quanto foi investido, conservação dos livros, detalhamento das obras, entre outras. Diante do constante aumento no valor dos livros e queda na sua qualidade [26], nota-se uma decisão mais seletiva por parte do usuário, que por sua vez, demandará de mais tempo, devido à escassez de informações mais específicas acerca dos títulos.

Este trabalho teve como objetivo criar uma plataforma voltada para colecionadores de obras literárias, trazendo diferenciais, tais como um comparador de livros com informações sobre a obra (material, quantidade de páginas, preço, resumo), notas baseadas na opinião dos usuários na plataforma (custo benefício e dificuldade de adquirir um título ou coleção) e um leitor de código de barras que, ao escanear um livro irá fornecer um identificador, o *International Standard Book Number* (ISBN). Isto possibilita ao usuário realizar uma busca da obra no site e obter dados sobre os títulos sem a necessidade de tê-los de forma manual.

Palavras-Chave

Aplicação web, desenvolvimento de software, colecionador de livros

Link para repositórios estruturados

https://github.com/BooKollection

Link para a aplicação

https://bookollection-frontend.herokuapp.com/

1. INTRODUÇÃO

Durante a era da digitalização, em que as pessoas utilizam a internet para leitura digital, o colecionismo de obras literárias é algo que se mantém presente, de acordo com a pesquisa [1] realizada pela Nielsen BookScan e o Sindicato Nacional dos Editores de Livros (SNEL). No mercado literário brasileiro, foram movimentados cerca de 1,3 bilhões de reais no ano de 2021, representando um aumento de 40% em relação ao ano anterior [11]. Este público inicialmente utilizava fóruns, blogs para compartilharem suas coleções, trocar experiências e conhecimentos adquiridos nas obras literárias.

Todavia, com o avanço tecnológico, blogs e fóruns de obras literárias estão se tornando obsoletos em consequência do surgimento de plataformas onde os colecionadores podem selecionar a ordem de leitura, opinar sobre os títulos presentes na plataforma, organizar a sua coleção e expô-la para outros usuários. No site SKOOB criado em 2009, com o foco de ser uma rede social, milhares de usuários trocam opiniões, livros, publicam a sua coleção, fazem grupos de leitura e checam editoras. Neste ambiente, o usuário pode cadastrar sua obra, ter uma grande variedade de obras brasileiras e poder criar tags para organização. Em um aspecto americano, temos o goodreads que contempla essas funcionalidades do SKOOB mas com maior detalhamento, uma biblioteca em inglês e melhor sistema de recomendação. Entretanto, ocorrem problemas em ambas as plataformas no aspecto de acessibilidade, sendo elas:

1. Idioma: o SKOOB é exclusivamente em português e o GOODREADS é em inglês, e isso pode limitar o público. Esse cenário ocorre com frequência no Brasil, visto que, a pessoa que não sabe inglês encontra grande dificuldade para usar a plataforma;
2. Responsividade: como ambos os sites possuem aplicativo mobile, as versões web não funcionam apropriadamente em ambiente de dispositivo móvel, dificultando sua utilização.

Em adição, essas plataformas focam em aspectos de leitura e na interação dos usuários, mas carecem de funcionalidades que ajudem o colecionador a tratar os problemas que serão descritos a seguir.

1.1 Financeiro

Considerando o cenário inflacionário presente e a escassez de matéria prima usada na fabricação dos livros, bem como o aumento no preço, por exemplo, do papel, que teve um aumento de 65% no seu valor apenas no ano de 2022 [2], nota-se uma diminuição do poder aquisitivo dos usuários voltados para este tipo de conteúdo. Isto faz com que o público seja mais seletivo quando for comprar livros, levando-os a utilizar softwares como planilhas para poder acompanhar os gastos, sendo uma tarefa exaustiva quando se tem uma grande coleção.

1.2 Conservação

Devido a muitos usuários não possuírem condições ideais para conservação de livros, como um local com pouca exposição à luz e umidade, muitas vezes as obras tendem a ter uma vida útil menor, resultando num tom amarelado no papel e insetos como traças.

1.3 Aspectos subjetivos

Em função de um grande número de obras, necessita-se de tempo para que o usuário possa avaliar se vale a pena colecionar ou não determinada obra, visto que muitos títulos possuem uma quantidade vasta de volumes. Por isso se faz necessário pesquisar a viabilidade da aquisição por aspectos de disponibilidade e valor.

2. SOLUÇÃO

Este trabalho teve por objetivo criar uma plataforma direcionada para os colecionadores de obras literárias, auxiliando na administração de uma coleção. Para cada volume ou coleção, a plataforma possuirá as informações descritas anteriormente, na qual as de cunho subjetivo serão alimentadas pelos usuários da plataforma e as de cunho físico e básico (editora, idioma, tipo de capa, dimensões e formato ideal da embalagem de polipropileno) serão disponibilizadas pela própria plataforma. Essas informações poderão ser usadas em um comparador de livros, a fim de se mostrar as vantagens e desvantagens de cada exemplar.

Além disso, o colecionador poderá informar na hora do cadastro da coleção, a data da compra e o valor pago no exemplar. Com isso, os dados podem ser usados para acompanhar os gastos feitos durante a sua coleção por unidade de tempo (dia, mês, ano) e poder informar o seu estoque de itens auxiliares utilizados para manutenção na coleção, como por exemplo a quantidade de embalagens de polipropileno separados por tamanhos. Além do foco em funcionalidades, a plataforma também possui o foco na acessibilidade e escalabilidade (capacidade de aumentar a produção sem perder a qualidade) ao público alvo, por isso, está sendo utilizado a internacionalização (i18n) [12], no qual o site tem suporte para as linguagens inglesa e portuguesa, podendo suportar outras línguas e a adoção da biblioteca de código aberto o vLibras [11]. Esse tipo de suporte proporcionará o acesso a pessoas com deficiência auditiva, no qual traduz conteúdos digitais (texto, áudio e vídeo) em Português para linguagem de sinais.

3. FUNCIONALIDADES

No sistema existem três perfis possíveis para um usuário, sendo eles: 1. autenticado; 2. o não autenticado; e o 3. administrador.

Para as rotas consideradas públicas, ou seja, páginas que qualquer tipo de usuário pode acessar, é possível ver todas as últimas obras, volumes e autores cadastrados (Figura 1).

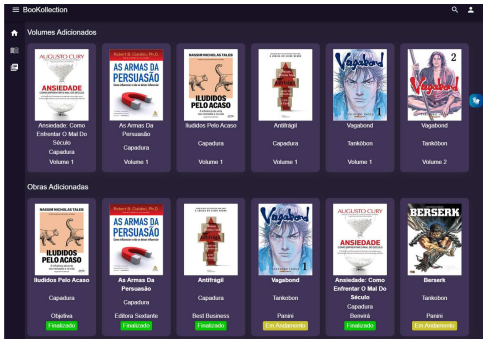

Figura 1. Tela inicial da aplicação

O usuário autenticado terá permissão para adicionar um volume à própria coleção informando, opcionalmente, preço pago, data da aquisição e nota para o volume ou obra adicionada (nota, dificuldade para aquisição), como pode ser visto na figura 2.

Figura 2. Janela para adicionar um volume a coleção

Para melhor visualização, este usuário poderá saber quais volumes ele não possui através de um *label* que fica acima da foto do volume (figura 3).

Figura 3 Opções para o usuário adicionar um volume

Outra funcionalidade é na página da coleção, que possibilita o leitor se informar acerca do número de obras literárias, volumes presentes na coleção, valor da coleção (figura 4) e ao adicionar um volume na aba de obras literárias um cartão referente a obra no qual o volume está contido (figura 5). Desse modo, será possível, ao abrir o cartão da obra literária, ver informações como nome, editora, dimensões, e na parte de volumes, todos os materiais publicados em ordem cronológica de publicação.

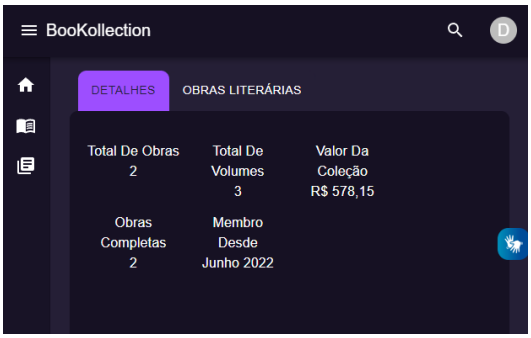

Figura 4. Tela dos detalhes da coleção do usuário

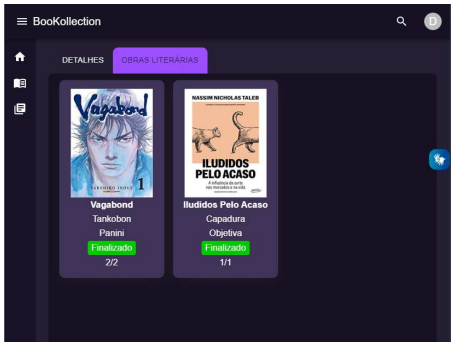

Figura 5. Tela das obras literárias da coleção do usuário

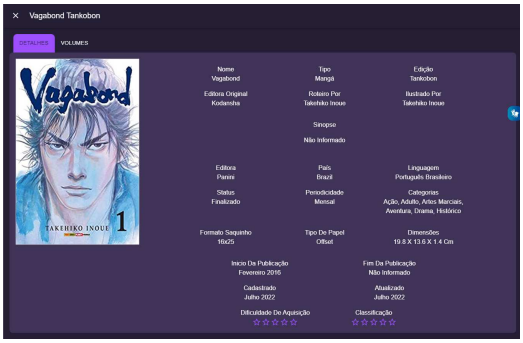

Figura 6. Tela dos detalhes de uma obra

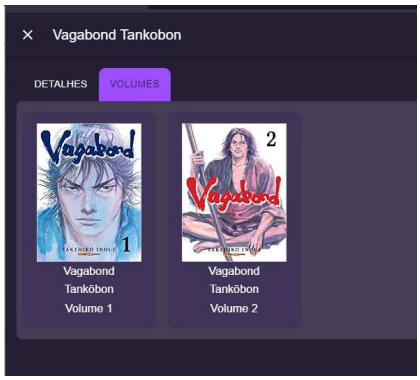

Figura 7. Tela dos volumes de uma obra literária

O administrador, por sua vez, poderá fazer tudo que um usuário autenticado tem acesso, e criar, editar e deletar um livro, autor, obra. Além disso, terá informações que são traduzidas de acordo com o idioma no navegador do usuário.

Contudo, o usuário terá informações sobre obras, volumes e autores, caso esteja *logado*, podendo ver também informações que foram passadas sobre a aquisição do mesmo, como data de aquisição e valor pago.

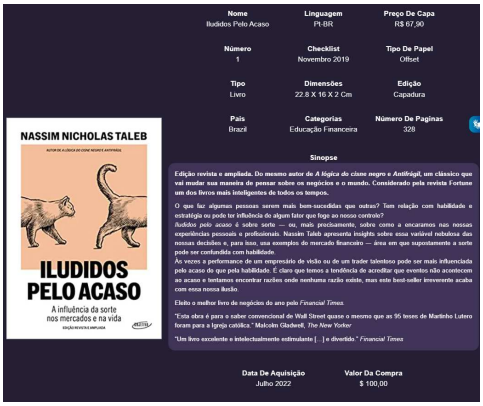

Figura 8. Detalhes de um volume

4 ARQUITETURA

Para descrever a arquitetura do BooKollection será utilizada a documentação C4 [27], no qual é constituído pelo contexto, *container,* componentes e código. Cada um desses elementos é responsável por explicar um nível de abstração, partindo da parte do mais alto nível do sistema, explicando o como o usuário interage com o sistema e como ele interage com os outros sistemas, até um nível mais detalhado. Assim, quanto menor o nível, mais detalhada é a documentação, trazendo um melhor entendimento da solução.

4.1 Contexto

O BooKollection utiliza serviços externos para algumas funcionalidades do sistema, sendo elas autenticação (OAuth Google), repositório de imagens (imgur) e a API para cotação do valor de uma moeda para outra (awesome API) como visto na figura 9.

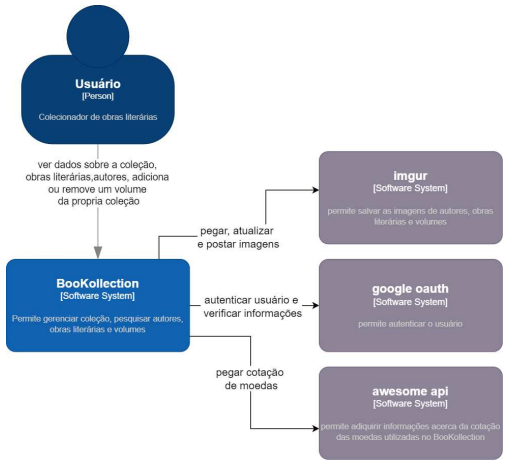

Figura 9. Contexto do BooKollection

4.2 Containers

A aplicação possui 2 containers, aplicação web e o servidor.

Aplicação web serve de interface para o usuário, podendo fazer requisições GraphQL [6] para o servidor, podendo criar, consultar, atualizar e deletar informações. Isso trará, flexibilidade nas informações enviadas, visto que o cliente decide o que o servidor precisa prover.

Para garantir que as informações enviadas para o *login* no sistema são válidas, é utilizado o serviço de autenticação com a Google nos dois containers. Para guardar as imagens utilizadas na aplicação, é usado o *imgur*, um serviço de hospedagem de imagens.

O servidor possui a regra de negócio da aplicação, que provê os dados para a aplicação web, se conectando com o banco de dados e acessa a API de cotações de moedas (awesome API) como visto na figura 10 .

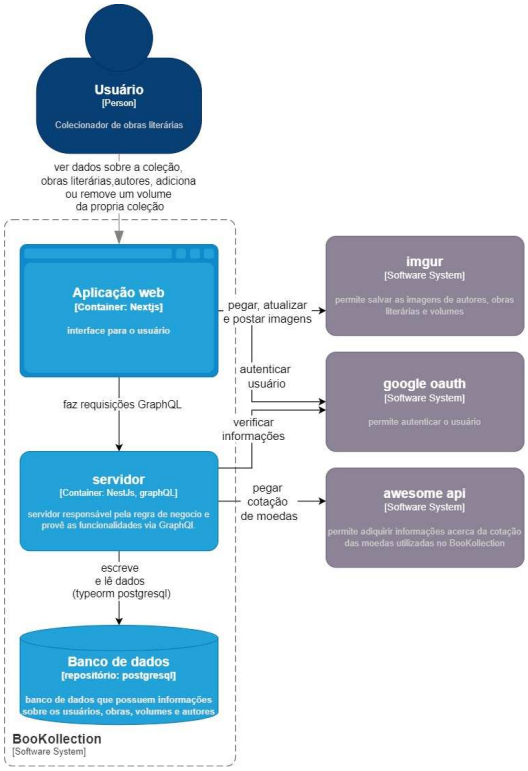

Figura 10. Containers do BooKollection

4.3 Componentes

4.3.1 Servidor

O servidor possui módulos responsáveis por uma ou mais entidades no banco de dados. Cada módulo, exceto pelo o de autenticação, possui a seguinte estrutura:

1. Resolver: responsável pela comunicação com o cliente através do GraphQL e repassar as informações para a camada de serviço;
2. Serviço: responsável pela regra de negócio (domínio), formatando os dados que serão repassados para o cliente. Em seguida, envia os dados já processados para a camada de banco de dados e acessa as APIS externas da aplicação;
3. Repositório: responsável pelo acesso ao banco de dados.

Como dito anteriormente, o módulo de autenticação possui uma estrutura diferente, pois contém estruturas que são utilizadas em outros módulos. A figura 11 mostra o fluxo da informação no momento do login dentro da autenticação:

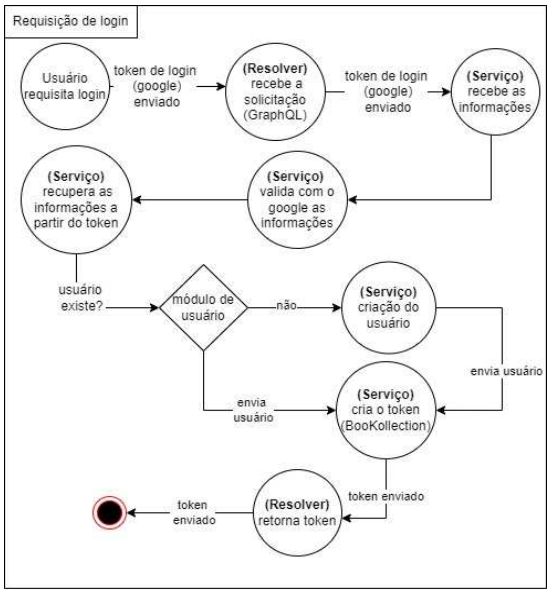

Figura 11. Fluxo do módulo de autenticação

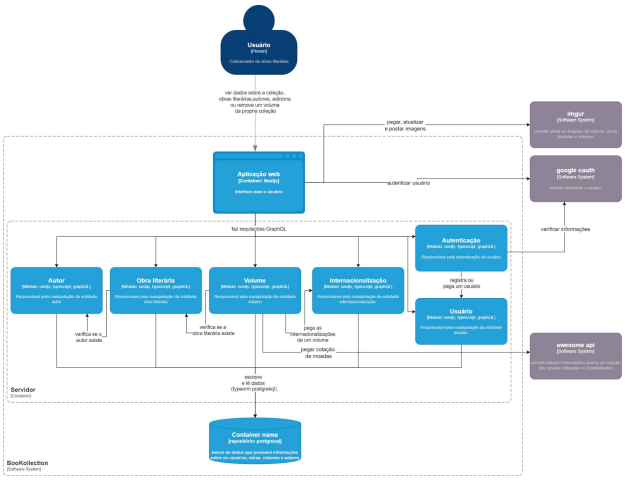

Figura 12. Visão geral do servidor

4.3.2 Cliente

O cliente está estruturado, seguindo os princípios do *atomic design* [10], sendo dividido nas seguintes partes:

1. Componentes: possui todos os componentes auxiliares a serem usados na aplicação. Os componentes são divididos em três partes: átomos, moléculas e organismos, partindo dos componentes mais simples para os mais complexos
2. GraphQL: responsável por definir mutações, consultas e instanciar o cliente do graphQL para comunicação com o servidor;
3. *Pages*: são as páginas na aplicação, responsáveis pela parte de roteamento e lógica;
4. Estruturas compartilhadas:

a. *Shared*: possui os arquivos que são compartilhados por toda a aplicação, como constantes, i18n, e tipagem;
b. *Store*: contém o gerenciamento do estado global e do componente de carregamento e das informações do usuário (id, nome, email e token);
c. *Styles*: contém as configurações globais de css e definição do tema.;
d. Utils: contém funções auxiliares que podem ser usadas em todos os lugares da aplicação.

5. *Template*: contém a parte *html* das páginas, diminuindo o acoplamento da parte lógica com a parte visual;

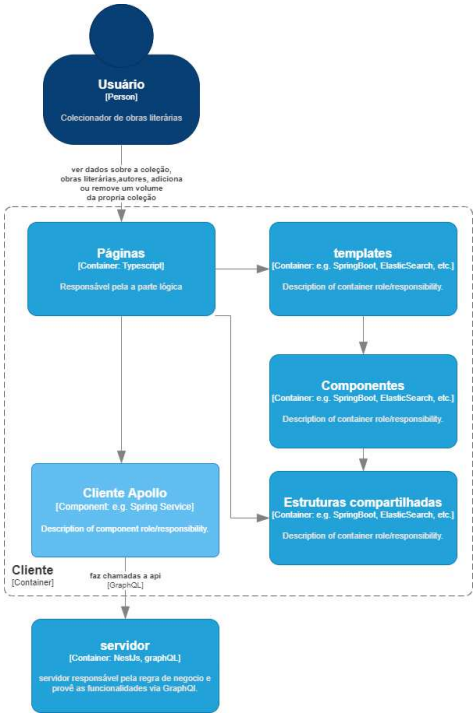

Figura 13. Componentes do cliente.

4.4 Implementação

4.4.1 Tecnologias do front-end

Na implementação do cliente, foi utilizado o nextJs [9], um framework para React onde há separação de cliente e servidor. Assim é possível a renderização estática de conteúdo pelo lado do servidor sem a necessidade do javascript habilitado e otimizar a renderização de imagens, trazendo mais performance. Em adição temos o uso integrado do Typescript e a facilidade na implementação da internacionalização [12].

Para facilitar o desenvolvimento, foi utilizado o materialUI [15] para uso de componentes, evitando uma quantia significativa de tempo que seria gasto na criação, pois a personalização é muito mais simples e também é possível definir o tema de cores no sistema. Para gerenciar o estado global e componentes na aplicação, foi utilizado o Redux [16], visto que, muitos componentes utilizados globalmente seriam incluídos em múltiplos lugares do sistema, resultando em redundância. Para o componente de carregamento e de toast (aviso) e para as requisições GraphQL [6] foi utilizado o Apollo Client [17].

Para ser possível o acesso global do Redux e do tema é necessário a utilização de um contexto, o que significa que ele engloba toda a aplicação. Nesse caso, existem três contextos na aplicação: google, store (Redux) e tema (MateriaUI).

4.4.2 Autenticação cliente

A autenticação no cliente foi implementada utilizando OAuth, um serviço da Google. Usando este serviço é enviado o token, nome e email.Com essas informações é enviado uma requisição para o servidor na hora do login para a obtenção das credenciais do mesmo (figura 14).

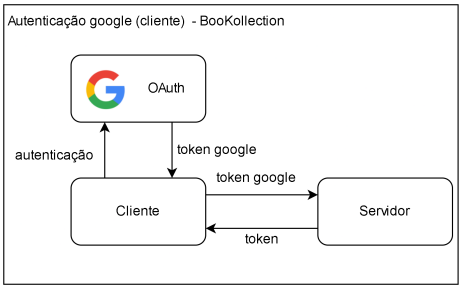

Figura 14. Autenticação google (cliente)

4.4.3 Tecnologias do back-end

O servidor foi desenvolvido usando Nestjs [1], framework que possibilita a modularização, uso integrado do TypeScript e escalabilidade da aplicação, acoplados a ele o GraphQL [2]. A comunicação entre o cliente e servidor é gerenciada pelo servidor Apollo que proporciona facilidade no desenvolvimento e criação da API, facilita o uso do Jest [10] para criação de testes de unidade e integração.

Para banco de dados, foi decidido usar o TypeORM [3] com PostgreSQL para garantir uma camada de abstração para manipulação do SQL [9] e o uso das migrações. Esses elementos são responsáveis por montar, modificar e desmontar o esquema do banco de dados automaticamente, mitigando muitos dos erros humanos gerados na hora de manipular um esquema SQL.

4.4.4 Estrutura do back-end

O projeto feito em Node Js foi dividido em 3 pastas, sendo elas a *config*, o *modules e o shared*. Também foram criadas bibliotecas que visam a qualidade de código e dos commits, sendo eles o EsLint, Prettier, Husky e Commit-lint.

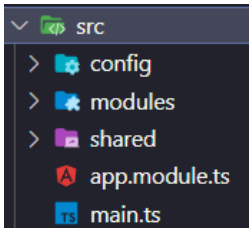

Figura 15. Organização do projeto.

A pasta config registra os enums utilizados nas consultas GraphQL. Ele tem a finalidade de padronizar informações, como a linguagem a ser utilizada (pt-BR, en-US) e define a conexão com o banco de dados a partir do arquivo de configuração **.env**.

A pasta shared possui funções compartilhadas entre todos os módulos, por exemplo os DTO, migrações e utilidades.

A pasta modules é organizada em subpastas, representando uma entidade no sistema e seguindo o padrão do DDD (Domain-Driven Design) [4]. Essa Estrutura é orientada à especialização do domínio, ou seja, o módulo possui seções com responsabilidades distintas, sendo elas importante em todo o fluxo da aplicação: Dto, infra, interfaces, services.

O DTO (Data Transfer Object) representa objetos que são usados na manipulação de informações entre os submódulos.

A camada infra é responsável pelo recebimento da requisição e pelo acesso ao banco de dados, sendo o primeiro chamado de GraphQL (HTTP) e o segundo de *database*. A seguir descrevemos cada passo com maiores detalhes.

GraphQL (HTTP): é responsável por padronizar para o cliente a forma de comunicação entre ele e o servidor. O GraphQL [2] obriga que o servidor siga exatamente a estrutura da informação requisitada pelo o cliente, ou seja, criando uma exatidão nos dados que serão transferidos entre ambos. Após receber os dados do cliente, os mesmos são encaminhados para a camada de *services*.

A camada de *Services* é onde está toda a regra de negócio do módulo, ou seja, é a parte especializada que manipula os dados seguindo as regras definidas na plataforma.

O bloco de interfaces, serve de intermediário para as camadas Services e Infra, no qual cada um é representado por uma interface que dita as funcionalidades presentes na camada. Além disso, por conta da injeção de dependência presente no NestJs [1], é possível que esse modelo funcione, pois no arquivo **.module,** presente em um módulo, o objeto responsável em cumprir a função da camada em que está inserido e implementar essa interface.

Database: é responsável por todo o assunto relativo à persistência SQL no módulo usando o TypeORM, possuindo os arquivos de repositório e entidade.

Entidade: pelo TYPEORM é possível mapear uma tabela no banco de dados usando notações como @Column, usado para representar uma coluna.

Migração: criação que automatiza o create e drop da entidade no banco. Todas as migrações ficam em uma pasta compartilhada no código fonte devido a regras do TypeORM, no qual cada arquivo possui um timestamp no seu nome para definir a ordem de execução.

4.4.2 Módulo de autenticação

Para simplificar a autenticação, foi utilizado o OAuth2 da Google. O servidor recebe um token e é feito uma requisição para a Google pedindo as informações do usuário id, email e nome. Após isso, é feito uma busca local através do módulo de usuário, passando o id como parâmetro. Caso o usuário não seja encontrado, ele é criado em seguida e retorna um *json*, contendo:

1. *Role*: papel do usuário no sistema (usuário, Administrador);
2. Token: encriptado, usando um método de envio de um token para autorização (JWT) que possui *payload* id, role e nome;
3. *Name*: fornecido no cadastro.

4.4.3 Estrutura do módulo de autenticação

Diferente da estrutura dos outros módulos, a parte de autenticação possui as seguintes partes adicionais em relação aos outros módulos:

1. JWT: responsável pela segurança nas partes de infraestrutura na aplicação (HTTP), possuindo:
    a. Guardas: responsáveis por interceptar a rota autenticada, usando JWT;
    b. Decoradores: responsável por pegar o contexto *GraphQL*, extrair informações, por exemplo, se o usuário está fazendo a requisição e o papel dele (*role*);
    c. *Service*: responsável por criar o token a partir da chave secreta, passada nas variáveis de ambiente (.env);
    d. *Strategy*: responsável por descriptografar o token.

2. GraphQL: responsável por receber a requisição de login.

5. Avaliações dos usuários

Para poder avaliar o desempenho do sistema, foi aplicado um questionário com um total de 5 pessoas (figura 16).

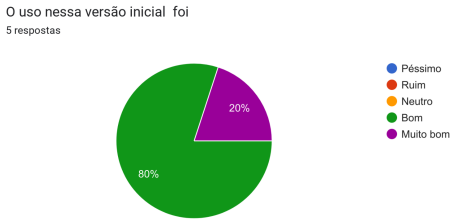

Figura 16. Avaliação sobre a satisfação no uso

Como visto nos gráficos, a maioria das notas sobre a versão inicial foram boas no MVP, vendo-se que nem todas as

funcionalidades foram implementadas. Isto gerou uma nota mediana. Este foi o resultado do primeiro grupo de teste, que apresentou muitas sugestões de modificação na plataforma.Todavia, a partir da terceira pergunta do formulário, é possível ter em saber quais as funcionalidades são as mais prioritárias para o desenvolvimento futuro (. Isto pode ser visto na seção 6.3.

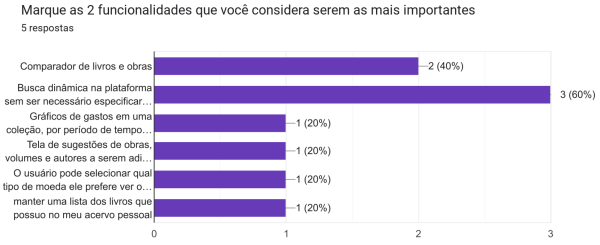

Figura 17 Gráfico sobre as funcionalidades futuras

Na última pergunta, foi abordado quais as partes do sistema poderiam melhorar e foi obtido respostas em três tópicos, design, funcionalidades futuras e bugs, a seguir temos algumas das sugestões dadas pelos usuários:

1. Design
    a. Tamanho do componente que mostra os detalhes sobre uma obra".
    b. Não está claro o que significa "finalizado" e "em andamento". O que foi finalizado? O que está em andamento? (figura 5).
    c. Uma fonte maior para o nome do autor".
    d. Menor número de clicks para o usuário realizar a ação desejada, melhor. Por essa razão, sugiro tirar esse botão de detalhes dos autores e das obras.
2. Funcionalidades futuras
    a. futuramente, login pelo facebook, para conseguir compartilhar a coleção entre outras ferramentas no feed.
    b. Colocar uma máscara de dinheiro no momento de informar o preço da obra.
    c. Disponibilizar uma opção que remova uma obra cadastrada (e também uma opção para desfazer, caso o usuário apague acidentalmente.
    d. O botão referente a ferramenta Vlibras não está funcionando.

6. Experiência

6.1 Processo de desenvolvimento

Inicialmente, foram escolhidas as tecnologias a serem utilizadas no sistema, a partir de tecnologias que estão em alta no mercado e a experiência adquirida em outros projetos.

Em seguida foram decididas as *features* a serem criadas no sistema, criação do esquema relacional (banco de dados) e UML [21] utilizando o draw io [20].

Para a organização das atividades a serem executadas, foi utilizado a metodologia kanban [3], em que foram criadas colunas referentes às etapas do desenvolvimento (releases), no qual possuem cartões representando as atividades.

Cada cartão de uma release possui uma tag que representa a frente na qual está incluída (frontend e backend), uma descrição, uma lista de atividades, responsável pela atividade, data limite e imagens trazendo esquema relacional para o backend e design para o frontend.

Para versionamento de código, foi utilizado a ferramenta github [18] e o processo gitlow para organização do desenvolvimento em versões, no qual temos o ramo main, onde fica o código desenvolvido testado e validado a ser usado em produção e os ramos criados a partir dele (release), sendo nomeados no seguinte padrão, release numero-tema.

Cada release tem um tema que determina o escopo das atividades a serem desenvolvidas, como administrador, onde todas as features criadas serão referente a este tipo de usuário, em um período de duas semanas.

Para desenvolver uma atividade, o desenvolvedor irá ler a especificação, indicar no cartão ao qual será o responsável, em seguida criará um ramo no github, a partir do ramo referente a release atual, no seguinte padrão release número-nome.

Após terminar o desenvolvimento, o cartão será movido para a coluna de validação, criado um pull request para revisão e após validado será concluído e movido para a coluna de completo e feito a junção do ramo do desenvolvedor com o da release atual, sendo o mesmo feito a junção com o main.

6.2 Desafios

Durante o desenvolvimento, ocorreram problemas com origem nos detalhes sobre as notas e valores de volumes, visto que, inicialmente foi desenvolvido o módulo de coleção do usuário, no qual era responsável por "guardar" as aquisições, valores da coleção e outras métricas. Quando foi criado a entidade "user volume" referenciando a transação de aquisição de um volume, ocorreram problemas nas importações dos módulos referentes a coleção, visto que, como vários módulos necessitavam desse módulo para referenciar o usuário, indisponibilizando o mesmo a importar outros módulos para construir os aspectos quantitativos e qualitativos da coleção, por exemplo, valor total da coleção, pois ao fazer essa importação, resultando em dependências circulares, ou seja, o módulo ao importar o outro, era importado ele e quebrava o código.

Para resolver esse problema, o módulo de coleção foi excluído e substituído pelo usuário, centralizando todas as ligações nele em formato de um para muitos [22].

Ocorreram problemas com versionamento de bibliotecas no frontend e backend, quando foi necessário atualizar as versões no package.json.

O primeiro foi o *framework* React, no qual o mesmo foi atualizado e um componente responsável pelo login com a google, só aceitava versões desatualizadas do react e com isso foi necessário substituí-lo por uma versão mais atualizada..

O segundo desafio foi com o typeorm, pois antes da atualização a versão desse ORM era 0.2.45 mas no momento da atualização para 0.3.1 ocorreram problemas de compatibilidade com o *framework* do back-end (*NestJs*), pois o mesmo precisou ser atualizado para uma versão compatível. Após essa atualização, foi preciso adaptar todo o código e os scripts de criação do banco de dados do BooKollection que teve uma duração de 4 dias.

Mesmo com esse empecilho a atualização dessa biblioteca de banco de dados trouxeram melhor escalabilidade, menor acoplamento e facilidade na manutenção para a aplicação, visto que, agora é possível injetar a camada de repositório de uma entidade fora do módulo do mesmo.

6.3 Trabalhos futuros

Futuramente, já foi planejado o desenvolvimento de funcionalidades que foram apontadas durante o desenvolvimento, sendo elas:

1. Comparador de livros e obras;
2. Busca dinâmica na plataforma sem ser necessário especificar o tipo (obra literária, volume, autor);
3. Gráficos de gastos em uma coleção, por período de tempo e estatísticas de aquisição de volumes, especificando quais são os gêneros literários mais adquiridos;
4. Tela de sugestões de obras, volumes e autores a serem adicionados;
5. Tela de relatos de bugs na plataforma;
6. O usuário pode selecionar qual o tipo de moeda ele prefere ver, os valores monetários na plataforma e o idioma.
7. Leitor de código de barras.

7. REFERÊNCIAS

[1] O mercado de livros cresce quase 40% e fatura R$ 1,3 bi desde o início do ano.

https://veja.abril.com.br/cultura/mercado-de-livros-cresce-quase-40-e-fatura-r-13-bi-desde-inicio-do-ano/

[2] Livros: preço do papel dispara e tiragens encolhem.
https://www1.folha.uol.com.br/mercado/2022/03/preco-do-papel-dispara-e-editoras-encolhem-tiragem-de-livros-e-hqs.shtml.

[3] Como a metodologia Kanban é aplicada ao desenvolvimento de software.
https://www.atlassian.com/br/agile/kanban

[4] Fluxo de trabalho de Gitflow.
https://www.atlassian.com/br/git/tutorials/comparing-workflows/gitflow-workflow

[5] Nestjs documentação,introdução.

https://docs.nestjs.com/

[6] GraphQL documentação, introdução.

https://graphql.org/learn/

[7] TypeORM documentação, introdução.

https://typeorm.io/#/

[8] O que é DDD – Domain Driven Design.

https://fullcycle.com.br/domain-driven-design/

[9] Nextjs documentação, introdução.

https://nextjs.org/docs

[10] Atomic Design: o que é, como surgiu e sua importância para a criação do Design System.

https://medium.com/pretux/atomic-design-o-que-%C3%A9-como-surgiu-e-sua-import%C3%A2ncia-para-a-cria%C3%A7%C3%A3o-do-design-system-e3ac7b5aca2c

[11] VLibras, tradução automática para tornar a web mais acessível.

https://www.gov.br/governodigital/pt-br/vlibras

[12] What is i18n?

https://lingoport.com/what-is-i18n/

[13] O que é sql.

https://www.alura.com.br/artigos/o-que-e-sql

[14] Jest através no Nestjs.

https://docs.nestjs.com/fundamentals/testing

[15] MaterialUI, uma biblioteca de componentes para ReactJs.

https://mui.com/pt/

[16] Redux uma biblioteca para controle de estado em javascript.

https://redux.js.org/

[17] Apollo Client uma biblioteca para controle do GraphQL.

https://www.apollographql.com/docs/react/

[18] GitHub é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão.

https://github.com/

[19] Fluxo de trabalho do gitflow.

https://www.atlassian.com/br/git/tutorials/comparing-workflows/gitflow-workflow

[20] Draw io, A maneira mais fácil para as equipes do Confluence colaborarem usando diagramas .

https://drawio-app.com/

[21] UML, linguagem-padrão para a elaboração da estrutura de projetos de software.

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

[22] Relationships in SQL – Complete Guide With Examples.

https://blog.devart.com/types-of-relationships-in-sql-server-database.html

[23] O que é ORM?

https://www.treinaweb.com.br/blog/o-que-e-orm

[24] Mangá: o que é, origem, tipos, como ler - Brasil Escola

https://brasilescola.uol.com.br/artes/o-que-e-manga.htm

[25] O que são comics?

https://www.desenhodg.com/2012/08/o-que-sao-comics.

[26] Porque o livro é tão caro no brasil.

https://super.abril.com.br/cultura/por-que-o-livro-e-caro-no-brasil

[27] The C4 model for visualizing software architecture.

https://c4model.com