UNIVERSIDADE FEDERAL DO MARANHÃO
ENGENHARIA DA COMPUTAÇÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA - CCET

MATEUS DA SILVA OLIVEIRA

APLICATIVO WEB PARA O GERENCIAMENTO DE CLUBES DE LEITURA E LIVROS

São Luís
2023

MATEUS DA SILVA OLIVEIRA

APLICATIVO WEB PARA O GERENCIAMENTO DE CLUBES DE LEITURA E LIVROS

Trabalho de Conclusão de Curso II, apresentado ao curso de Engenharia da Computação, como requisito para a obtenção do Título de Bacharel em Engenharia da Computação. Centro de Ciência Exatas e Tecnológicas da Universidade Federal do Maranhão

Orientador: Prof. Dr. Davi Viana dos Santos.
Coorientadora: Profa. Dra. Patrícia de Maria Silva Figueiredo.
Colaboradora: Erlane Maria de Sousa Alcântara.

São Luís
2023

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

DA SILVA OLIVEIRA, MATEUS.
Aplicativo web para o gerenciamento de clubes de leitura e livros / MATEUS DA SILVA OLIVEIRA. - 2023.
50 f.

Orientador(a): Davi Viana dos Santos.
Curso de Engenharia da Computação, Universidade Federal do Maranhão, SÃO LUÍS, 2023.

1. Clube de leituras. 2. Gerenciamento. 3. Livros. 4. Software. 5. Solução tecnológica. I. Viana dos Santos, Davi. II. Título.

MATEUS DA SILVA OLIVEIRA

APLICATIVO WEB PARA O GERENCIAMENTO DE CLUBES DE LEITURA E LIVROS

Trabalho de Conclusão de Curso II, apresentado ao curso de Engenharia da Computação, como requisito para a obtenção do Título de Bacharel em Engenharia da Computação. Centro de Ciência Exatas e Tecnológicas da Universidade Federal do Maranhão

BANCA EXAMINADORA

Profº Dr. Davi Viana dos Santos
(Orientador)
Universidade Federal do Maranhão

Profº Dra. Patrícia de Maria Silva Figueiredo (Coorientador)
Universidade Federal do Maranhão

Profº Dr. Sérgio Souza Costa
Universidade Federal do Maranhão

Profº Ms. Alana de Araújo Oliveira Meireles Teixeira
Universidade Federal do Maranhão

São Luís
2023

RESUMO

O trabalho aqui desenvolvido teve como intuito desenvolver um aplicativo voltado para o gerenciamento de clubes de leitura e de livros, ele se baseou na proposta oferecida na dissertação de mestrado de (ALCÂNTARA, 2022) que teve como objetivo principal propor uma solução tecnológica para administração e compartilhamento de leituras literárias, bem como promover a leitura. O Clube de Leituras disponibiliza ferramentas robustas tanto para membros quanto para administradores de clubes, facilitando o gerenciamento do clube, seus participantes além de permitir o gerenciamento de leituras paras os leitores. O Aplicativo foi desenvolvido com ferramentas, frameworks e arquitetura frequentemente usada nos âmbitos acadêmicos e mercado como Express, React. A partir deste trabalho, um software foi desenvolvido, para auxiliar clubes de leituras, e leitores, nas suas leituras e encontro.

Palavras-chaves: gerenciamento; clube de leituras; livros; software; solução tecnológica.

ABSTRACT

The work developed here was intended to develop an application aimed at managing reading and book projects, it was based on the proposal offered in the master's thesis of (ALCÂNTARA, 2022) whose main objective was to propose a technological solution for the administration and sharing of literary readings, as well as to promote reading. The Clube de Leituras provides robust tools for both members and club administrators, facilitating the management of the club, its participants, as well as allowing the management of readings for readers. The Application was developed with tools, frameworks and architecture often used in academic and market areas such as Express, React. Based on this work, software was developed to help reading clubs and readers in their readings and meetings.

Keywords: management; book club; books; software; technological solution.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

FGV	*Fundação Getúlio Vargas*
UI	*User Interface*
AI	*Artificial Inteligence*
REST	*Representational State Transfer*
API	*Application Programming Interface*
DDD	*Domain Drive Desing*
HTTP	*Hypertext Transfer Protocol*
JSON	*Javascript Object Notation*
MVC	*Model - Control - View*
SOA	*Service-Oriented Architecture*
PWA	*Progressive Web Apps*
SSR	*Server Side Render*
SPA	*Single-Page Application*
CSR	Client-Side Rendering
JWT	*JSON Web Token*
TI	Tecnologia da Informação
HTTP	Hypertext Transfer Protocol

Sumário

1 INTRODUÇÃO

A tecnologia moldou a vida no século XXI, é inimaginável um mundo sem acesso aos acervos culturais e literários compartilhado de forma quase instantânea de outros povos. Através de um smartphone ou outro aparelho eletrônico é possível ouvir uma música ou ler um livro produzido a quilômetros de distância.

No Brasil, atualmente estima-se que há mais de 424 milhões de aparelhos digitais, entre: computadores, notebooks, tablets e smartphones (FGV, 2022), o que representa mais de dois aparelhos por habitante. Diante disso, questionasse o quão conectado e quais são as formas de consumo dos brasileiros, segundo a Forbes (FORBES, 2021), o Brasil é o pais que mais consome aplicativos no mundo em média são consumidas cinco horas por pessoa, e claro, os aplicativos mais usados são as redes sociais.

Comparativamente, o mercado de livros cresceu cerca de 40% e fatura 1,3 bilhão de reais, isso é o que vale a 31,79 milhões de exemplares de livros vendidos[1]. Isso revela um aumento no consumo de livros, podendo implicar o aumento da leitura dos brasileiros.

Por outro lado, a pesquisa intitulada de Retratos da Leitura no Brasil, a única pesquisa em âmbito nacional que tem por objetivo avaliar o comportamento do leitor brasileiro, realizada durante os anos de 2007 à 2019 e pelo grupo Instituto Pró-Livro (IPL) e com apoio do Itaú Cultural e IBOPE Inteligência, revela dados nada favoráveis ao Brasil. Na 5º edição, revelou que para o ano de 2019, 48% não são leitores e que em média são lidos somente cinco livros por ano, e 15% dos brasileiros, com mais de cinco anos, alegam que não leem porque não compreendem ou têm dificuldades para ler. Ainda nessa pesquisa, foi exposto que 20% dos entrevistados só começaram a se interessar por leitura por ter participado de grupos, oficinas ou clubes de leitura (Instituto Pró-Livro, 2020).

E é nesse contexto que os clubes de leitura desempenham um papel importante. Um clube de leitura, ou também clube de livros, é um grupo de pessoas com interesse em discutir determinado livro e cada grupo de leitura tem sua identidade própria, suas regras e interesses mas todos têm como objetivo discutir sobre livros. Foi a partir do objetivo de facilitar gerenciamento de clubes de leituras e livros, influenciado pela dissertação de mestrado de (ALCÂNTARA, 2022), que surgiu a ideia para o desenvolvimento do aplicativo Clube de Leituras .

Assim, com o intuito de democratizar a leitura, ampliar pensamento crítico e criar formadores de opinião, surge a necessidade de Clube de Leituras um aplicativo com a intenção de juntar pessoas que queiram criar boas discussões em torno de um livro,

[1] Disponível em: https://veja.abril.com.br/cultura/mercado-de-livros-cresce-quase-40-e-fatura-r-13-bi-desde-inicio-do-ano/ - acessado em 11/07/2022

amparados pelo apoio de clubes.

1.1 Justificativa

A especialista em Docência de Língua, Andréa Schmitz-Boccia em sua pesquisa "Clubes de leitura: a construção de sentidos em situações de leitura colaborativa" busca compreender a importância do clube de leitura para a formação de um leitor. Em sua pesquisa ela observou que os participantes dos clubes de leituras desenvolvem mais hábito e qualidade de leitura, de certa forma gerado pelo compromisso com o grupo. (BOCCIA, 2012).

Assim, baseado na argumentação de que grupos de leituras favorecem o incentivo à leitura, e aliado ao trabalho desenvolvido por (ALCÂNTARA, 2022) surge a necessidade do Clube de Leituras: uma aplicação que permite a criação de uma comunidade literária que fortalece e incentiva o hábito de leitura, além de criar dados para o uso de políticas públicas.

1.2 Objetivo Geral

Este trabalho visa desenvolver um aplicativo para o gerenciamento de clubes de leituras e livros, além de utilizar as tecnologias e arquiteturas do desenvolvimento web.

1.3 Objetivos Específicos

- Projetar e desenvolver um aplicativo para gerenciar clube e livros, utilizando tecnologias recentes de desenvolvimento.
- Avaliar a aceitação do usuário em relação ao aplicativo e realizar alterações a partir disso.

2 REFERENCIAL TEÓRICO

O desenvolvimento de software foi modelado em toda sua trajetória através de padrões, tendo como intuito melhorar, facilitar e organizar. Este não é um movimento recente: a criação de arquitetura é um processo que ocorreu desde antes das criações das máquinas modernas, antes mesmo da computação, baseando-se em áreas de estudos mais padronizadas como a matemática. Para dar continuidade ao trabalho vamos apresentar alguns tópicos de arquitetura e tecnologias que são importantes para o entendimento e desenvolvimento deste trabalho.

2.1 Clube de Leituras

Segundo (SOUZA; COSSON, 2011) ler é a competência cultural mais valorizada, e a falta da leitura pode ser vista de maneira negativa enquanto a presença, é sempre vista de maneira positiva. Assim, a leitura é fundamental nas nossas vidas permeando todas as suas nuances e definindo aquilo que somos: seres humanos (SOUZA; COSSON, 2011).

A prática do letramento literário pode ser efetivada em três movimentos. O primeiro, é o contato do leitor com a obra, onde é exigido a experiência e bagagem pessoal social do leitor; o segundo, trata-se da partilha do seu conhecimento por meio da leitura em grupo; e o terceiro, é o compartilhamento de experiências vindas dos seus colegas (COSSON; LUCENA, 2022). Então, o letramento em grupo é fundamental e uma estratégia importante para formação de um leitor.

Assim, para realizar este trabalho e buscar uma solução para o gerenciamento de clubes de leituras, foi definido um clube de leitura como um grupo de pessoas que se reúnem periodicamente para debater sobre um livro escolhido de maneira coletiva ou individual e que compartilham suas experiências e visão acerca do que foi lido (FERREIRA, 2021). Atualmente existem diversos clubes de leituras no país, que estão organizados de maneiras variadas e podemos encontrar alguns localizados em redes sociais como o instagram, sendo alguns deles o Chicas e Dicas, Leituras Decoloniais, Clube Latinas, Clube Traça, dentre outros (FERREIRA, 2021).

2.2 REST API

Antes de iniciar a aplicação a cerca do REST e suas restrições é importante explicar a distinguir alguns termos que são usados no decorrer deste trabalho e que em outros trabalhos podem ter outro significado. O primeiro é cliente, em inglês *client*, é a aplicação

ou software que o usuário final, ou seja, aquele para o qual o software é criado, tem acesso e interage através de uma interface. O servidor, em inglês *server*, é a aplicação ou software, que processa as requisições do cliente, onde usuário final não tem acesso diretamente, e na maioria das vezes o servidor é usada pra manter os segredos de negócio.

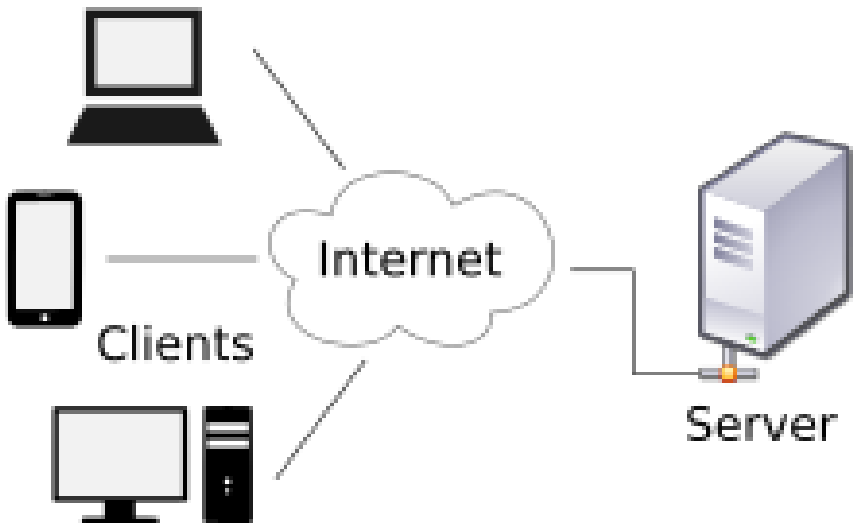
Assim, podemos prosseguir, A REST (Representational State Transfer) é um estilo de arquitetura web criada pelo cientista da computação Roy Fielding, usada para distribuir *hypermedia systems* (FIELDING, 2000) utilizando o protocolo HTTP (Hypertext Transfer Protocol). Como descrito por Roy Fielding,

> The design rationale behind the Web architecture can be described by an architectural style consisting of the set of constraints applied to elements within the architecture.(FIELDING, 2000, p.76)

o REST é um conjunto de restrições, que através destas pode-se criar diversas aplicações ou API. Ele se baseia através dos seguintes critérios: *Client-Server*, *Stateless*, *Cache*, *Uniform Interface*. Estes critérios vão ser explorado em subtópicos mais abaixos.

Iniciando por *Client-Server* prescreve que um servidor oferece uma quantidade de serviços, escuta solicitações, enquanto o cliente faz as solicitações desejadas enviando pedidos ao servidor como mostrado na figura 1. Separando a estrutura desta forma, pode-se obter maiores vantagens, dentre elas: permitir a comunicação de clientes em múltiplas plataforma, além de manter uma maior escalabilidade e simplicidade dentro do componente servidor.

Figura 1 – Modelo Cliente-Servidor



Fonte: Retirado de <https://en.wikipedia.org/wiki/Client-server_model#/media/File:Client-server-model.svg>

Em seguida, a *Stateless* estabelece que cada solicitação do cliente ao servidor deva conter toda informação necessária para que o servidor atenda completamente a requisição.

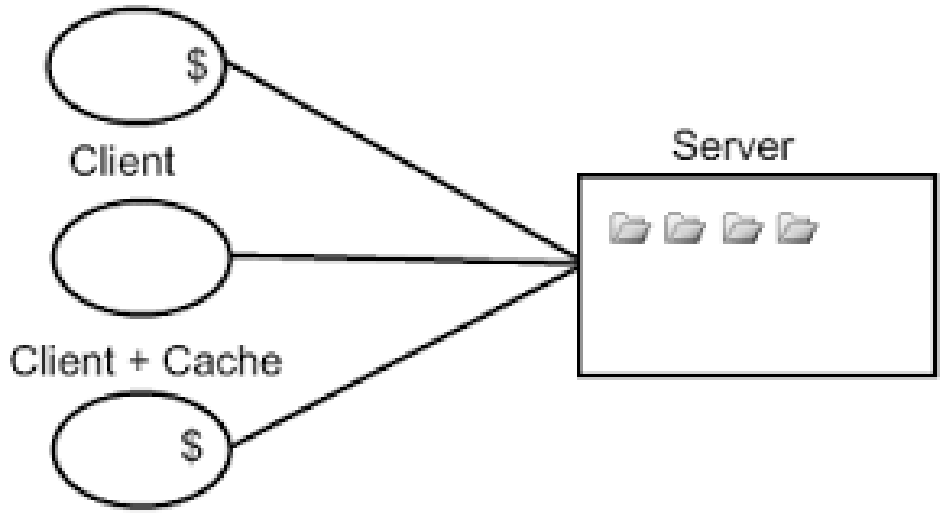
Assim, nenhuma informação das requisições anteriores dos clientes irão ser armazenadas, forçando o cliente a manter inteiramente o estado da sessão.

Essa restrição garante que o servidor não precise observar outros dados, além do que já existe na requisição, para garantir a solicitação. Mitiga ainda a possibilidades de erros, pois o servidor vai entregar exatamente o que o cliente quer.

Por outro lado, esta restrição oferece uma desvantagem na diminuição do desempenho da rede, visto que o servidor pode sofrer sobrecarga de requisição repetitivas. Isto é bastante visto em aplicações/clientes implementados através de laços de repetição infinitos não tratados, criado uma areá de ataque, se não tratar corretamente. (FIELDING, 2000)

A terceira restrição descrita pelo Roy Fielding é referente a *cache*. Essa restrição prescreve que uma solicitação deve conter a informação de quê implicitamente ou explicitamente precisa de um dado cacheado ou não. Se uma resposta puder ser armazenada em cache, um cache de cliente terá o direito de reutilizar esses dados de resposta para solicitações equivalentes posteriores.

Figura 2 – *Cache* - comunicação cacheável e *stateless*



Fonte: (FIELDING, 2000).

A *layered system* determina que o sistema deva ser composto por camadas com hierarquia, que organiza os tipos de servidores. As camadas podem ser para cuidar da segurança, tratar a carga, dentre outras e, nessa hierarquia cada camada só "visualiza"a camada adjacente na qual ela esta interagindo.

Por último, a *uniform interface* descreve que deve-se ter uma Interface Uniforme. Esta última restrição é sem dúvida a restrição característica do REST, é a que separa-a de todas as outros estilos, inclusive é muito comum compreender uma aplicação REST baseada somente nessa restrição.

A restrição de interface aponta que: cada recurso deve ser identificado; há manipulação de recursos por meio de representações, ou seja, o cliente deve conseguir manipular as respostas recebidas e as respostas devem conter informações de como processá-las e, por fim, o cliente deve ter apenas o caminho inicial da aplicação. Os outros recursos podem ser acessados através de hiperlinks.

2.3 *Clean Architecture*

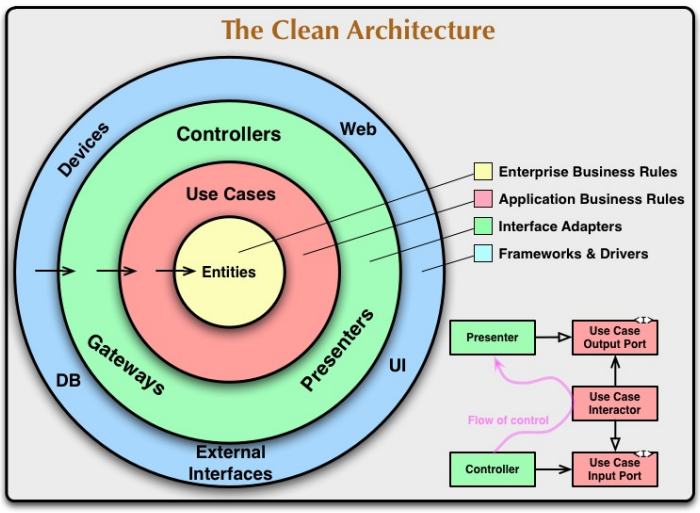
O *Clean Architecture* ou em português Arquitetura Limpa, foi uma abordagem proposta por Robert Martin baseando-se na Arquitetura Hexagonal, *Onion Architecture* dentre outras.

Robert Martin percebeu uma familiaridade entre essas arquiteturas, a separação de interesses. As arquiteturas observadas por Robert Martin apresentam uma separação de códigos em camadas, unindo-as através de portas ou interfaces, e cada camada só tem acesso à camada adjacente.

Robert Martin prôpos a separação em camadas de Entidades, Casos de Uso, Adaptadores e Frameworks, facilmente representada por camadas de círculos concêntricos, onde a camada mais profunda seria Entidades e a do mundo mais externo Frameworks.

Desta forma, tem-se um software independente de frameworks, de databases e um software desacoplado e testável. Abaixo abordaremos cada camada.

Figura 3 – *Clean Architeture - Diagrama*



Fonte: Página retirada da internet. Disponível em <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. Acesso em: 20 jun. 2023.

A entidade está no centro da arquitetura, esta camada é responsável pelas regras de negócios. Uma entidade pode ser uma classe ou qualquer objeto com métodos, desde que

possam ser usadas por várias aplicações e que sejam independentes de qualquer aplicação específica. Com isso, é observável que as regras aqui são generalizadas e de alto nível e estão menos propensas a mudanças. (MARTIN, 2012)

A camada de casos de uso vão implementar regras mais específicas da aplicação. Ela está entre a camada de controle e a camada de entidade. Nesta camada são implementadas todos os casos de uso da aplicação, ou seja, são definidas as interações dos atores com o sistema, de forma que o fluxo nessa camada não interfira na camada de entidades, assim como nada no ambiente externo altere nessa camada. Além disso, cada caso de uso representa uma funcionalidade completa e também quais são as etapas necessária pra conseguir realizar um objetivo específico.

A camada de adaptadores conectará as camadas mais internas, caso de uso e entidades, com a camada mais externa interface e sistemas externos. Essa camada engloba o controlador, *gateway* e *presenter*. Ela é responsável por receber todas as interações e requisições do usuário, ou de outro sistema, e fazer o tratamento inicial dessas interações pra assim passar para as camadas mais inferiores. Dentro de um sistema web, a camada do controlador é usado para fazer o tratamento de requisições HTTP, validar e extrair informações e encaminhá-las para os casos de uso, após isso, camada *presenter* receber os retornos do casos de uso formatar os dados e passar de maneira amigável para o cliente ou interface do usuário. De acordo com o Uncle Bob,

> It is this layer, for example, that will wholly contain the MVC architecture of a GUI. The Presenters, Views, and Controllers all belong in here.(MARTIN, 2012)

2.4 Aplicação Web

A seguir será detalhado alguns termos e informações acerca do aplicação web, sobre o que foi utilizado no cliente-servidor. Inicialmente abordaremos as formas de criar aplicações cliente-servidor, Posteriormente, será descrito internamente sobre cada uma.

Existem diversas formas de fazer de renderizar o cliente, dentre elas podemos citar o *Server Side Render* (SSR), *Client Side Render* (CSR), o Static Site Generation (SSG) e Hybrid Rendering.

No SSR a renderização é feita diretamente no servidor, e funciona da seguinte forma: o cliente acessa uma determinada página, o navegador faz uma requisição ao servidor e o conteúdo HTML vem pronto para o cliente, assim o cliente não precisa esperar o download do css e javascript para mostrar o conteúdo da página. Todas as páginas da aplicação são requisitadas ao servidor, e por sua vez o servidor constrói todas. Durante muito tempo, essa foi a única forma de desenvolver aplicações.

Com a melhoria dos dispositivos e computadores pessoais, criou-se novos mecanismo para a criação de aplicações como o CSR. Ao contrário do SSR, no CSR o cliente é

responsável por renderizar a página, e para isso, o navegador faz a requisição dos arquivos javascript e o css necessários e executa-os dentro do dispositivo do cliente uma única vez e a partir disso todas as página web vão sendo renderizadas dinamicamente pelo cliente. Esse método é comum em aplicações web construídas com frameworks JavaScript como React, Angular ou Vue.js.

Existem outras formas como SSG que gera cada página no servidor antecipadamente e envia ao cliente em cada requisição. E também o Hybrid Rendering que é uma mistura do SSR e CSR, onde algumas partes vão sendo construídas pelo servidor, geralmente as partes mais custosas computacionalmente, enquanto o cliente renderiza as partes mais simples ou que precisam de dados dinâmicos essa forma dinâmica é comum de ser utilizada em frameworks como o Next.js e Nuxt.js.

O React é uma biblioteca ou framework *frontend*, ou seja para o cliente, desenvolvido pelo time do Facebook é de código aberto e tem foco em criar interfaces de usuário em páginas web. O React é um dos muitos frameworks modernos que utilizam o paradigma de programação reativa, que enfatiza a criação de aplicativos que são capazes de lidar com mudanças de estado e fluxos de dados de forma eficiente e escalável. Isso torna o React ideal para criar aplicativos web dinâmicos e interativos, como jogos, aplicativos de mídia social e plataformas de comércio eletrônico. Ele permite que os desenvolvedores criem componentes reutilizáveis e altamente flexíveis que são capazes de responder de forma rápida e precisa aos inputs do usuário e outros eventos em tempo real.

Além disso aliado o React pode ser usado com bibliotecas de gerenciamento de estado como o Zustand ou Redux, que também são usados para resolver problema da transporte de dados entre componentes em uma aplicação, e permite que diferentes partes do aplicativo acessem e atualizem o estado de forma centralizada. O Redux é uma ferramenta muito poderosa e apresenta mecanismo para realizar a previsibilidade do estado da aplicação e a imutabilidade dos dados, é baseado em *store*, *actions*, e *reducers*, que respectivamente, servem para armazenar os estados da aplicação, reagir a ações, e especificam como o estado deve ser atualizado em cada ação. Por outro lado, o Zustand se baseia na mesma proposta do Redux, mais simplifica algumas das complexidades do Redux, como a necessidade de definir redutores e ações, ao invés disso utiliza *store hooks* para consumir e alterar o estado.

Para o servidor, a discussão envolve o Express[1], um framework web que disponibiliza uma gama de ferramentas para lidar com requisições HTTP, criação de rotas, gerenciar sessões, lidar com autenticação e etc. Durante o desenvolvimento utilizou-se o Ngnix[2] que é um servidor web e proxy reverso. Ele é servidor utilizado para lidar com as requisições e resposta HTTP além de hospedar o aplicativo web. Outro assunto pertinente é a

[1] https://expressjs.com/pt-br/
[2] https://www.nginx.com/

autenticação por meio de tokens, ao invez da autenticação comum por meio da senha e do acesso de usuário, o servidor cria um token, como o JWT (*JSON Web Token*), e utiliza-se ele para realizar a autenticação.

Para acesso ao banco de dados, foi pensando no *repository pattern* que permite abstrair os detalhes de acesso aos dados e fornece uma interface consistente para interagir com a base de dados. Ele encapsula a lógica de persistência e recuperação dos dados através de interface com métodos simples: ***create***, ***delete***, ***get***, ***getAll*** e ***update***.

3 METODOLOGIA DA PESQUISA

Neste capítulo, são descritos os passos seguidos para o desenvolvimento deste trabalho e quais ferramentas e métodos foram utilizados. O trabalho foi divido em cinco etapas: o primeiro passo foi uma prospecção de aplicativos e posteriormente a realização do levantamento de requisitos. O terceiro consistiu no desenvolvimento dos modelos e diagramas. Em seguida, foram desenvolvidos os primeiros esboços e protótipos de telas. E a última etapa, foram definidas as ferramentas de trabalho e o desenvolvimento do trabalho que será explicado no próximo capítulo.

3.1 Prospecção de aplicativos

Na primeira etapa, foi feita uma prospecção de aplicativos e sites com o objetivo de avaliar aplicativos que têm o objetivo igual ou parecido ao proposto por esse trabalho: gerenciar livros, leituras e clubes. Essa prospecção teve como finalidade buscar recursos ou requisitos que poderiam ser útil ou que permitisse uma distinção para esse trabalho. Foi realizado uma pesquisa exploratória através, principalmente, do Google Play[1] que teve como observado 64 aplicativos de leitura e nenhum de gerenciamento de clubes de leitura, mas esse número sobe um pouco quando é observado as páginas web. Na Tabela 1, temos aplicações que foram observadas e que serviram como base para o projeto aqui desenvolvido. Os projetos dessa tabela apresentam, em algum ponto, algo relacionado a esse trabalho: compartilhamento de experiência literária, encontros virtuais de leitura, distribuição de livros.

Tabela 1 – Sites de gerenciamento de clubes de leituras e livros

	Nome	URL
1	TAG Livros	https://site.taglivros.com
2	Goodreads	https://www.goodreads.com
3	Literal Club	https://literal.club
4	StoryGraph	https://app.thestorygraph.com
5	Bookclubs: Organize your book club	https://bookclubs.com

O site TAG Livros é um clube de assinatura de livros onde todo mês o assinante recebe um *kit* literário. O Goodreads é um site de recomendação de livros e conta com *reviews*, recomendações e discussões sobre livros. Nele é possível receber recomendações baseadas nas suas leituras anteriores e nas dos seus amigos. O Literal Club é um site para descobrir, organizar e discutir leituras. No Literal Club é possível criar objetivos individuais de leitura, organizar biblioteca, identificar o progresso de um livro, buscar

[1] Disponível em: https://play.google.com/

livros por código de barras, importar sua biblioteca para outros sites como o Goodreads e Storygraph, criar notas e destaques, escanear e compartilhar trechos de livros com câmera, seguir amigos e juntar a clubes de leitura. O StoryGraph é similar ao Goodreads ou seja é um catálogo de livros. O Bookclubs é um site de clubes de leituras e leitores que permite organizar seu clube, começar um novo ou encontrar um clube. O Bookclubs apresenta recursos como avaliar livros, criar e gerenciar clubes, cadastro de reuniões e enviar convites para outras usuários.

3.2 Requisitos Funcionais e Não Funcionais

Dando continuidade, a segunda etapa foi o levantamento de requisitos. Para identificar os requisitos necessários, após a prospecção de aplicativos, foi analisado suas funcionalidades e identificado o que poderia ser incorporado ou diferenciado no nosso projeto. Nessa etapa foram levantados os Requisitos Funcionais e Requisitos Não Funcionais, para determinar quais as principais entregas da plataforma. Abaixo, serão detalhados as entradas e saídas dos requisitos que foram levantados, sendo todo o processo validado por uma bibliotecária, que está auxiliando a aplicação;

1. Cadastrar todos os livros que o clube já leu.

   Esse item se refere ao cadastro de um livro na estante do clube, o livro deve entrar na estante do clube de duas maneiras: após a discussão sobre o livro em um encontro cadastro no sistema ou adicionado diretamente pelo administrador do clube.

2. Definir a agenda de encontros.

   Os clubes cadastrarão quais dias serão realizado os encontros via sistema, podendo abrir ou não uma votação para os membros escolherem a melhor data.

3. Definir a agenda de votações.

   Os clubes criação poderão criar votações que permitiram os membros do clube decidirem quais serão as datas de encontro e o livros a serem lidos.

4. Criar votação dos livros.

   Todos os membros do clube podem votar para a escolha do livro da vez.

5. Cadastrar os clube e política de ingresso.

   Qualquer usuário logado do aplicativo poderá criar um clube e adicionar uma política de ingresso.

6. Realizar curadoria de clubistas.

   Os administradores realizaram a manutenção dos clubistas. Os gerenciadores do clube podem excluir, expulsar, aceitar ou dar uma função aos clubistas.

7. Cadastrar clubistas.

   Permite ao um usuário entrar entrar em um clube.

8. Manter estante de livros do usuário (livros lidos, livros que está lendo, livros desejados, progresso da leitura).

   Um usuário pode gerenciar seus livros, adicionando qual o status que o livros está dentre os valores: livros lidos, livros que está lendo, livros desejados e livros abandonados. Assim como verificar a quantidade de livros e páginas já foram lidos.

9. Cadastrar novos livros.

   Um usuário poderá cadastrar novos livros caso não tenha encontrado um livro específico.

10. Importar base de livros existente.

    Permitir a importação de livros de outros serviços ou aplicações, nesse foi escolhido a base do Google [2].

11. Avaliar livros.

    Classificar um livro em um determinado intervalo de valores.

12. Acompanhar progresso da leitura (quantas páginas já foram lidas).

    Informar quantas páginas já foram lidas de um determinado livro.

13. Registrar comentários sobre os livros

    Avaliar um livro através de comentários.

14. Configurar notificação.

    Notificar um usuário através de notificação de e-mail ou através do sistema quais encontros e livros foram cadastrados pelos clubes em que ele é membro.

Para os requisitos não funcionais, foram elaborados os seguintes:

1. Reservar espaço para logomarca do aplicativo na interface.

   O aplicativo deve reservar um espaço para mostrar a logomarca, permitido criar uma identidade visual.

2. O sistema deve ser implementado com javascript,

   A implementação deve ser feita usando javascript, como mecanismo de aproveitar as facilidades de implementação da linguagem.

[2] https://books.google.com.br/

3. Integrar-se com uma API externa para acesso à capa e número de páginas dos livros.

   Com o objetivo de mostrar a capa e número de páginas de um livro, deverá se integrar a uma API externa.

4. Implementar autenticação de usuários.

   O usuário realizá o login, usando senha e username ou email, mas para validar sua autenticação o sistema deve implementar o acesso de recursos privados via token JWT, garantido que a senha não seja salva no cliente.

5. Criptografar as senhas dos usuários no armazenamento.

   A senha deve ser criptografada usando algoritmo hash.

6. Evitar o cache de senhas no cliente.

   O sistema não deve guardar a senha do usuário no aplicativo, com o objetivo de não criar facilidades para o vazamento de informações sigilosas.

7. Disponibilizar interface responsiva para adequação a diferentes tamanhos de tela.

   O sistema deve oferecer uma interface adaptativa e que seja otimizada para diferentes dispositivos e tamanhos de tela, incluindo desktops, tablets e smartphones.

8. Garantir compatibilidade com os navegadores modernos mais usados (Chrome, Firefox, Opera, Safari).

   O usuário deve ter acesso as funcionalidades do aplicativo nos navegadores mais usuais.

3.3 Diagramas e Modelos

Em seguida, após o levantamentos do requisitos o passo seguinte foi elaborar os modelos de dados, entidade-relacional e diagrama de casos de usos. Para isso, foi utilizado o Diagrams.net [3] um software de desenho gráfico.

O Modelo Entidade Relacionamento foi o primeiro a ser feito, com o intuito de ter uma visão geral sobre a aplicação. Na Figura 4, temos o modelo mais simples e na Figura 9 há uma imagem onde pode-se ver este modelo como todo e como foi projetado o trabalho.

Em seguida, foi feito o diagrama de casos de usos, na figura 11 pode se olhar os casos de uso completo. Na figura 11, é possível observar o ator Usuário que é o ator mais simples do sistema, ele executa uma série de funcionalidades iniciais como registrar, visualizar clubes, efetuar login.

[3] Disponível em: https://app.diagrams.net/

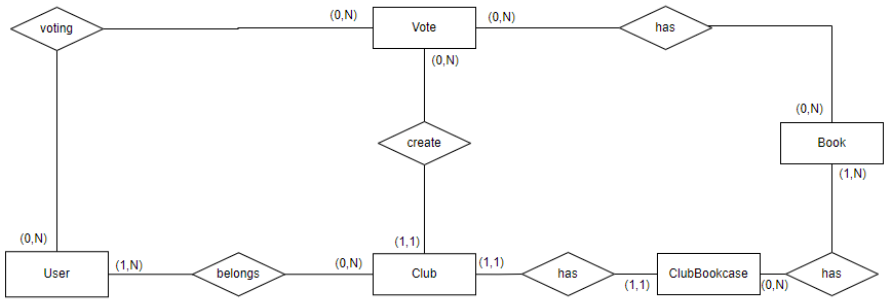


Figura 4 – Modelo Entidade Relacionamento Principal

Em seguida, pode-se observar o ator Clubista que herda todas as ações do usuário. O Clubista é todo usuário cadastrado em um clube com membro. Assim, ele participa ativamente em um clube e realiza ações como cadastrar um voto, cadastrar comentário, etc.

O ator Mediador que herda todas as ações do usuário. O Mediador é todo usuário cadastrado em um clube e designado pelo criador do clube como mediador. Ele é responsável por uma série de manutenções que são necessárias para manter a ordem do clube. E o último ator observado é o Criador, ele herda todas as ações do Mediador, além claro de criar e excluir o clube e adicionar novos mediadores.

Por fim, para finalizar a etapa dos diagramas foi desenvolvido o modelo de dados. A Figura 10, representa o modelo de dados desenvolvido neste trabalho. O Modelo de dados é composto minimamente por quatro Entidades: *Club*, *User*, *Book* e *BookCase*. Durante o processo de desenvolvimento foi observado que estas entidades deveriam ser estendidas para criar algumas funcionalidades e para isso foi criado *MeetClub*, *Election*, *UserBookCase*, *Comments* além de outras tabelas auxiliares . O *Club* é a tabela responsável por salvar as informações de um clube onde é determinado o nome, a descrição e se é um clube privado, ou seja, se será necessário pedir permissão para entrar e visualizar as informações. Para *User*, foi observado que o modelo abstrato de um usuário precisaria de um nome, e-mail e senha. Em *Book* e refletido um livro contendo superficialmente o título, autores e por fim o *BookCase* que será visualizada como a estante do clube.

Partindo para as outras tabelas temos *MeetClub* que é uma funcionalidade da nossa aplicação e será responsável por guardar a informação das reuniões do clube. *Election*, o clube terá uma central de eleição, onde decidirá o livro a ser lido ou o dia da reunião. E por fim, *Comments* que é uma estrutura organizacional de comentários.

3.4 Prototipação de Telas

Com os modelos e diagramas, foi iniciado a prototipação de telas. Nessa etapa utilizamos o Figma[4], um editor gráfico de vetor e prototipagem de projetos de design. Pensando em mobile-first, foram desenvolvidos alguns protótipos de baixa fidelidade e posteriormente foram criados alguns de alta fidelidade. Na Figura 5, pode-se visualizar os primeiros protótipos, os de baixa fidelidade, enquanto na Figura 6 temos os de alta fidelidade.

Figura 5 – Esboço de Baixa Fidelidade

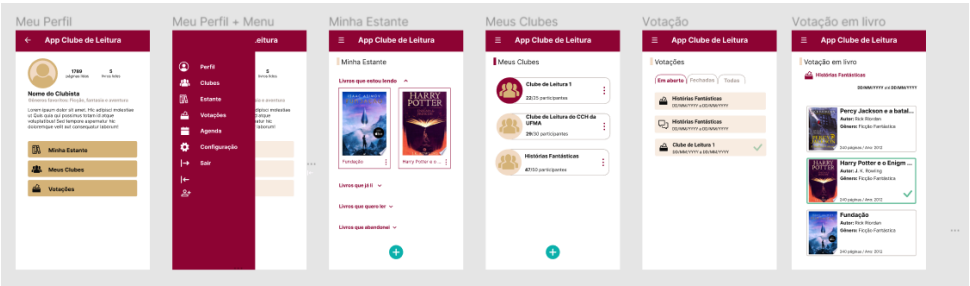


Figura 6 – Protótipo de Alta Fidelidade

[4] Disponível em: https://www.figma.com/

4 IMPLEMENTAÇÃO E DESIGN DO SISTEMA

Neste capítulo, será abordado os decisões técnicas, os problemas observados e artefatos gerados. Inicialmente, será descrito sobre a idealização do projeto e após isto, serão abordados detalhes da aplicação no servidor, seguido do cliente e por fim um relato sobre a API.

4.1 Idealização do projeto

A parte inicial do projeto foi definir o escopo da aplicação, o que seria, pra quem e o porquê. A aplicação nasceu com o objetivo de ser utilizada como gerenciador de leituras e clubes. Neste etapa será realizada uma breve exposição do escopo básico que foi definido na aplicação e também os requisitos.

A aplicação será dividida em duas partes distintas: o gerenciador de leituras, direcionado aos usuários, e o gerenciador de clubes, destinado aos administradores. Cada parte desempenha funções específicas e oferece recursos relevantes para seus respectivos usuários. Na gerência de leituras para os leitores foi pensando em uma arquitetura em que um leitor pudesse gerenciar:

- Estante: Nessa seção, o aplicativo deveria ser capaz de ajudar o leitor a organizar seus livros, a quantidades de páginas lidas, mostrar o estado de leitura, se o livro já foi lido, está sendo relido ou mesmo se foi abandonado.
- Clubes: O leitor além de gerenciar seus livros permitiria o leitor a se inscrever e acompanhar um ou mais clubes.
- Votações: Para leitores que ingressam em clube seria permitido a ele participar das decisões do clube, e permitindo assim que decidam quais livros serão discutidos e quando serão.
- Encontros: Novamente uma aba para clubistas, o clubista deve ser capaz de visualizar os encontros que foram definidos em nos clubes que ele faz parte.

Essa parte foi definida como o gerenciador de clube para o administrador e auxiliares. Alguns aspectos importantes que foram definidos como escopo para o gerenciamento de clubes.

- Estante: Assim, como na seção dos leitores um clube também terá sua própria estante. Com exceção que o clube não irá colocar diretamente um livro, e sim após o término de um encontro.
- Membros: Essa seção faz parte da curadoria de um clube, o clube pode ter a decisão de remover, aceitar, ou transformar o participante do clube.
- Votações: Parte de distribuir a decisão sobre a escolha de livros e de datas para os membros.
- Encontros: Encontros do clubes.

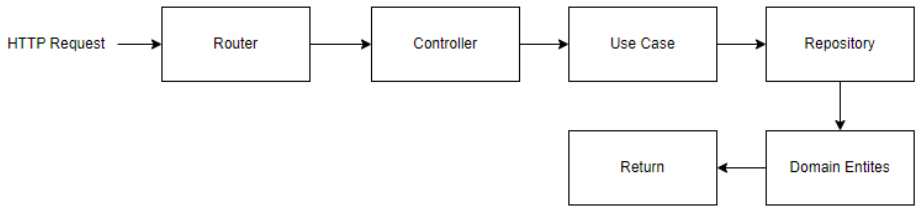
4.2 Servidor

O servidor foi desenvolvido levando em consideração padrões e arquiteturas descritos nos livros de referência como (MARTIN, 2017) e (EVANS, 2004). No entanto, é importante destacar que nem todos os conceitos foram aplicados estritamente, devido à equipe de desenvolvimento ser pequena e composta por desenvolvedores de nível básico. Dado o objetivo de entregar um software de qualidade de forma rápida, foi optado por adotar uma abordagem mais simples e inteligente, seguindo o conselho de (EVANS, 2004, p.72).

Desta forma, os conceitos foram adaptados à realidade deste projeto, visando obter resultados eficientes sem comprometer a qualidade do software. Assim, tentando fazer da melhor maneira possível e evitando *overengineering*, como *microservices*, *noSql*, infraestrutura mais complexa como da *AWS* foi utilizado coisas simples no desenvolvimento.

Assim, no servidor foi utilizado o design de arquitetura em camadas. Entretanto, como foi seguido a arquitetura cliente-servidor a camada mais externa do clean arquitecture será desenvolvida somente no cliente, na figura 7 é possível visualizar as outras três camadas desenvolvidas no servidor.

Figura 7 – Arquitetura do servidor



Fonte: Elaborado pelo autor

A primeira etapa foi criar a conexão do um servidor HTTP utilizando a framework

Express, e para facilitar, foi criado a classe **App** pra agrupar esse servidor web. No trecho de código 1, a classe **App** ela foi simplificado afim de entendimento mas dentro do método *routes* contém todas os parâmetros iniciais para as nossas rotas.

Código 1 – Inicialização da aplicação

```
class AppController {
    express
    constructor(){
        this.express = express();
        this.middlewares();
        this.routes();
    }

    middlewares() {
        this.express.use(express.static('images')};
    }

    routes() {
        this.express.use('/images', express.static('images'));
    }
}
```

A implementação da camada do controlador é precedida após a rotas, como falado anteriormente ela receberá as requisições e aplicará os primeiros tratamentos, passará para o caso de uso e retornará para o apresentador, como pode ser observado no trecho de código 2. No trecho de código 2 é possível visualizar a autenticação do usuário e validação das entradas para o usuário.

Código 2 – Controle do usuário

```
class UserController {
  async editUser() {
    const { userId } = this.request.params;
    const { birthdate, city, nick, fullname, description } =
   this.request.body;

    if (!userIsOwnRequest(this.request)) return NotAuthorizedError();

    const validatedUser = userRequestValidation(userId, {
      birthdate,
      city,
      nick,
      fullname,
      description
    });

    if(validatedUser.error) return ClientError();
```

```
17
18      const user = await this.userCase.getById(userId);
19
20      if (!user) return NotFoundError();
21
22      const updatedUser = await this.userCase.editUser(validatedUser,
        userId);
23
24      if (!updatedUser) return ServerError();
25
26      return { success: updatedUser }
27    }
28 }
```

A camada de casos uso é responsável por fazer a conexão com o *core* e o controlador. Na aplicação desenvolvida, ela também faz o acesso ao banco de dados através do *repository*. No trecho de código 3 temos a representação de um caso de uso, nela está contida a lógica pra cadastrar um usuário, é possível visualizar regras de negócio simples como a verificação de e-mail, criptografia de senha.

Código 3 – Caso de uso criar novo usuário

```
1  async createNewUser({ email, password, fullname }: Partial<User>) {
2      const user = await this.users.getByEmail(email);
3
4      if (user) return { error: 'Esse email já está sendo usado!' }
5
6      const salt = Math.floor(Math.random() * 10 + 1)
7
8      const cpassword = await cryptPassword(password, salt);
9
10     const newUser: User = {
11       salt,
12       roleName: 'primary',
13       password: cpassword,
14       email,
15       fullname
16     }
17
18     return await this.users.create(newUser)
19 }
```

Na camada de entidades, as regras de negócios foram criadas através de contratos com as interfaces das entidades. Esses contratos impedem que um determinada entidade receba valores que não deviam. No nosso caso, não foi criado de entidade tão fortemente engessadas, isso foi feito através dos próprios *type* da linguagem. Mas em projetos maiores,

ou mais complexos, as entidades fornecem métodos para alterar o estado e só permitem alterações válidas de acordo com as regras de negócios.

A camada de gateway que deveria ser criada pra ter acesso aos dados, e que deveria está no mesmo nível dos adaptadores foi usada no nível de casos de usos. Essa camada desempenha um papel fundamental no acesso à base de dados. No contexto da arquitetura de software, foi adotado o *repository pattern*. No trecho de código 4 é possível ver a implementação da interface do *repository pattern*.

Código 4 – Interface do Repository Pattern

```
export interface GenericRepositoryInterface<T>{
  create: (item: T) => Promise<T>;
  get(id: number): Promise<T>;
  getAll: () => Promise<T[]>;
  update: (id: number, item: T)=> Promise<number>;
  delete(id: number);
}
```

Essa interface será utilizado como contrato pra implementar a classe concreta, que pode ser visualizada no anexo E. Desse modo, podemos alterar o banco de dados, além de implementar repetidas vezes essas mesma função pra cada entidade. Após isso, é necessário uma classe ou no nosso caso uma classe pra cada entidade pra servir como porta para o *repository*. No trecho de código 5 temos uma parte do código que mostra como foi implementados esses acessos. Criamos uma classe que ***extends*** a classe do banco e passamos num construtor que será a tabela a ser acessada. Dessa forma podemos instanciar a classe DbElectionRepository em qualquer parte do nosso código que retornar todos os métodos do GenericRepositoryInterface, implementados pelo KnexGererericRepository e para a tabela ***election***.

Código 5 – Pontos de acesso para o repository

```
export class DbCandidateRepository<T extends Candidate> extends
    KnexGenericRepository<T> implements GenericRepositoryInterface<T>{
  constructor() {
    super('candidate')
  }
}

export class DbCommentRepository<T extends Comment> extends
    KnexGenericRepository<T> implements GenericRepositoryInterface<T>{
  constructor() {
    super('comments')
  }
}

```

```
13 export class DbElectionRepository<T extends Election> extends
       KnexGenericRepository<T> implements GenericRepositoryInterface<T>{
14   constructor() {
15     super('election')
16   }
17 }
```

Obviamente, no mundo real os métodos do GenericRepositoryInterface não são suficientes, existem muitos mais necessidades que simplesmente cinco métodos. Por isso, a primeira alteração foi dentro da interface, foi inserido mais quatro métodos, que podem ser observados no trecho de código 6. Esses métodos adicionais garantem outros necessidades do nosso código criar vários itens ao mesmo tempo, filtra por determinadas propriedade da entidade, filtrar por várias propriedade e paginar os dados. Assim todas as entidades podem ter esses métodos em mão.

Código 6 – Metodos adicionais na interface

```
1 createMultiples: (item: T[]) => Promise<T[]>;
2   filter: (item: Partial<T>) => Promise<T[]>;
3   filterArray: (item: keyof T, array: Array<T[any]>) => Promise<T[]>;
4   paginate: (paginate?: Paginated<T>) => GenericRepositoryInterface<T>
```

Ainda assim, somente esses métodos ainda não suficientes. Imaginemos que queremos obter todas eleições do usuário de todos os clubes que ele faz parte. Dessa maneira, precisaria fazer primeiro uma chamada pra todos os clubes que o usuário faz parte. Depois filtrar os id e fazer uma nova busca as eleições de cada clube, organizar e então retorna. Não parece uma solução meio boa, além de que não vale a pena deixar o aplicativo resolver essa questão sendo que bancos de dados são otimizados para isto. Assim, foi feito mais uma mudança no repository pattern: a criação de um repository específico para algumas entidades.

O primeiro passo foi criar uma interface específica, no trecho de código 8 tem-se a uma interface específica pra entidade User. Note que a interface faz uma extensão da interface genérica, com isso ele continua com os métodos padrões, mas agora com um método a mais *getByEmail*, com isso é necessário criar a classe concreta que vai implementar essa interface, como ser visto no trecho de código 9. E por fim, implementar o ponto de acesso 10.

Código 7 – Interface específica de usuário

```
1 interface UserRepositoryInterface<User> extends
      GenericRepositoryInterface<User> {
2   getByEmail: (email: string) => Promise<User>;
3 }
```

Código 8 – Interface específica de usuário

```
interface UserRepositoryInterface<User> extends
    GenericRepositoryInterface<User> {
  getByEmail: (email: string) => Promise<User>;
}
```

Código 9 – Classe Knex para entidade User

```
class KnexUserRepository<T> extends KnexGenericRepository<T> implements
    UserRepositoryInterface<T>{
  async getByEmail(email: string): Promise<T> {
    try {
      return await this.query
        .where(
          'email', email
        )
        .first()
    } catch (err) {
      return err;
    }
  }
}
```

Código 10 – Ponto de acesso para a base de dados da tabela users

```
class DbUserRepository<T extends User> extends KnexUserRepository<T>
    implements UserRepositoryInterface<T>{
  constructor() {
    super('users')
  }
}
```

Desse modo, foi visualizado tudo que foi utilizado na construção do servidor e na figura 12, pode-se ver o caminho completo desde a requisição no cliente até a resposta.

4.3 Cliente

O Cliente que nesse software pode ser chamado de frontend e tem grande importância na maioria dos sistemas, aqui não seria diferente, pois é parte visível ao cliente e não pode haver erros, *bugs* ou lentidão. Dessa forma, é de grande desafio construir um cliente testável, flexível, dinâmico, de fácil manutenção e resiliente. Para isso, é necessários fazer escolhas seguras de bibliotecas, *frameworks*, design e arquiteturas

Pensando nisso, o projeto foi decidido em ser Client Side Render (CSR), monolítico e multiplataforma web e com layout responsivo especialmente pelo pouco espaço de tempo disponível e pela complexidade do software não seria possível fazer uma versão web e

mobile nativa/hibrida. Assim, escolhemos o React como framework principalmente por sua facilidade em dividir interfaces complexas em componentes simples e reutilizáveis, além da possibilidade de utilizar os mesmos componentes no desenvolvimento de aplicativos mobiles com React Native.

O usuário pode ter diferentes formas de acesso ao aplicativo, como anteriormente falado, dentre elas o usuário pode ser: um usuário não autenticado, usuário autenticado, um administrador de clube, um clubista. A partir desse momento, o usuário autenticado e logado com papel de clubistas, mesmo que ele não esteja em clube algum, será chamado de clubista e o usuário autenticado mediador ou criador será chamado de administrador e o usuário não autenticado será chamado de usuário.

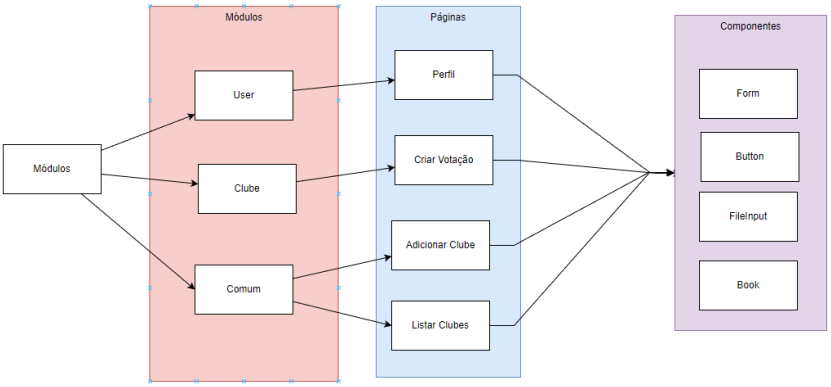
Então a proposta para o *frontend* foi criar um design em que o usuário fosse direcionados para as páginas dependendo do papel que ele assumiria. Então, teremos as duas principais modulação Usuário, para clubistas, e Clube para administrador e também um modulo comum para ambos. Na lista 4.3 podemos visualizar a forma de organização da aplicação no frontend.

- Usuário
    - Reuniões de clubes
    - Configurações
    - Listar Votos
    - Listar Clubes
    - Perfil
- Clube
    - Adicionar Encontro
    - Adicionar Votação
    - Adicionar Encontro
    - Gerenciar Membros
    - Listar Votos
    - Adicionar Encontro
    - Configurações
- Comuns
    - Perfil de Clube
    - Perfil de Usuário

- Adicionar Clube
- Página do Livro
- Estante
- Lista de clubes
- Lista de Livros
- Adicionar Livro
- Cadastro de usuário
- Página Inicial
- Login
- Logout

Outra forma, de organização do frontend está relacionado ao gerenciamento de estado e componentização. Na figura 8 é possível está, basicamente todo container[1] ou pages[2] tem acesso a aos components[3].

Figura 8 – *Módulos*



Fonte: Elaborada pelo autor.

Para gerenciamento e transporte de estados globais foi decidido utilizar a biblioteca Zustand[4], pouco comum de ser utilizada em aplicações web mais muito poderosa e facilmente refatorada se comparada com Redux ou Context[5] do React. Em comparação

[1] Nessa aplicação pode ser entendido como um conjunto de componentes geralmente usado para agrupar um elemento de interface visual complexo
[2] Nessa aplicação pode como um conjunto de componentes geralmente usada para construir uma página web
[3] Nessa aplicação pode ser entendido como pequenos elementos de interface visual
[4] https://github.com/pmndrs/zustand
[5] Hook do React encontrado em https://beta.reactjs.org/apis/react/useContext

com o Redux o Zustand é muito mais simples e não é necessário amarrar a aplicação em torno de um *provider* e os estados são facilmente acessados por meio de *hooks*[6].

Nesta aplicação têm-se duas store, uma pra armazenar os dados do usuário com id, nome, perfil e clubes e outra para armazenar dados do modal como a dado a ser usado pelo modal, o tipo de modal que vai ser aberto e actions para abrir e fechar o modal. Para gerenciamento de estados locais é usado hooks do proprio React, como useState, useCallback, useMemo e useReducer. É importante destacar que esses estados só existem no próprio componente e enquanto o componente está em tela, caso não for nenhum desses o estado não existe. Então para componentes que sua função era criar formulários ou listagem de dados, foi utilizado os React Hooks.

4.4 API

O cliente acessa os dados do servidor através de uma API como já descrito foi seguido o padrão de comunicação REST e o framework express, e nessa seção será detalhado um pouco mais sobre cada rota. O REST descreve que cada recurso deve ter seu identificador, assim na tabela 2 é possível visualizar todos os identificadores.

Tabela 2 – Tabela de Recursos da API

Identificador	**Retorno**
auth	Rota de autenticação, rota que o usuário não logado realiza para receber o token JWT e acessar os outros recursos privados.
meet	Rota referente a todos os serviços de encontros, fornecendo dados para os clubistas ou administradores.
book	Retorna todos os dados relacionados a livros e é destinada a todos os usuários do sistema.
election	Retorna informações sobre eleições, incluindo eleições, candidatos e votos.
genre	Refere-se aos gêneros literários.
user	Engloba todas as operações relacionadas aos usuários, como cadastro, edição e exclusão, dentre outras.
share	Contém elementos que não requerem autenticação e podem ser compartilhados, exemplo: alteração de senha.
club	Tudo que se refere ao funcionamento do clube, como cadastro, edição e exclusão, dentre outras.
bookcase	Refere-se às estantes de livros, tanto do clube quanto do usuário.
upload	Fornece serviços relacionados ao upload de imagens e documentos.
images	Expõe uma rota para que o cliente possa buscar imagens e documentos.

[6] No React é uma função especial que serve pra agrupar comportamentos, efeitos ou estados, geralmente usada quando se tem a possibilidade de remover duplicação de código ou quando se tem um comportamento complexo

Não será possível detalhar todas as rotas criadas na API, visto que são muitas, mas ela seguem os padrões exemplificados na tabela 3. Algumas rotas da API é necessário está autenticados, assim são utilizados tokens JWT, além disso é feito a autorização também via token JWT está é uma maneira de garantir um acesso seguro aos recursos sem a necessidade de salvamento da senha no cliente.

Tabela 3 – Tabela de Endpoints da API

Verbo HTTP	**Rota**	**Retorno**
GET	/books	Retorna a lista de todos os livros cadastrados
GET	/books/:id	Retorna os detalhes de um livro específico
POST	/books	Cria um novo livro
PUT	/books/:id	Atualiza as informações de um livro específico
DELETE	/books/:id	Deleta um livro específico
GET	/books/users/:userId	Retorna os livros de um usuário específico
GET	/books/clubs/:clubId	Retorna os livros de um clube específico

5 RESULTADOS

Com o objetivo de criar um aplicativo web para controle de clubes e gerenciamento de leituras é importante que os resultados a seguir sejam de relevância, o primeiro resultado é a materialização do aplicativo e atualmente pode ser encontrado em <https://www.clubesdeleitura.online/>, além disso foi feito o registro do programa de computador no INPI e pode ser encontrado pelo número de processo BR512022003624-0 que pode ser visualizado no anexo A. Além disso obtivemos outras resultados para mostrar a usabilidade do aplicativo, a eficácia no gerenciamento de clubes e leituras, além de outras métricas, que serão detalhados a baixo.

5.1 Avaliação da usabilidade do aplicativo

A avaliação de usabilidade foi baseado nos teste e uso pelos usuários depois foi coletados dados através de questionário. De acordo com os usuários o aplicativo se mostrou bem fácil de ser utilizado. Os usuários conseguiram navegar pelas funcionalidades do aplicativo de forma eficiente, sem a necessidade de instruções. As afirmações a seguir corroboram com a minha opinião acima: *"Sim, pois consegui realizar os objetivos do aplicativo."* , *"Sim, o aplicativo é muito leve e fluido, fácil de usar e tem funcionalidades muito boas como a busca por clubes de livro"* , *"Acredito que sim, pois o leitor tem controle sobre sua leitura e pode se conectar com outros leitores"*

Apesar das falas positivas acima, ainda há melhorias a se fazer no aplicativo, como mencionado na seguinte citação: *"Senti falta de um botão VOLTAR. Ao clicar Minha estante, a aba TODOS já poderia ficar verde; O botão MAIS que aparece no canto inferior direito da tela precisa de um posicionamento para melhor visualização (posição, tamanho, cor, destaque?). O mesmo sugiro para o menu de três pontos no canto superior direito."* , *"Não foi intuitivo cadastrar meus livros lidos"*

Há ainda algumas exceções em que UI não estava intuitivas como foi o caso da página de criação de eleições, em que os usuários não conseguiram entender a proposta e tiveram dificuldades em criar novas eleições ou votar, outro dificuldade se mostrou e ao escolher os diferentes papéis, como comentado por um usuário: *"Apenas não consegui visualizar como funcionam as tarefas de Encontros e Votação. Seria interessante o cadastro de informações para verificar como funcionam. Fiquei também interessada em saber como funciona a criação de um clube no App"* .

Com todas as avaliações positivas e negativos se sabe exatamente onde se pode melhorar, o que precisa ser aperfeiçoado.

5.2 Eficiência e eficácia do gerenciamento:

O aplicativo permite que os usuários registrem e gerenciem diferentes clubes de leitura. Os resultados, obtidos através da observação de dados na base de dados, mostraram que os usuários conseguiram criar novos clubes, adicionar membros, definir datas de reuniões. Além disso, é importante destacar a importância do próprio usuário, assim foram filtrados alguns comentários que destacam o tema sobre gerenciamento: *"Sim! No que se refere ao gerenciamento das atividades, o app possibilita os usuários organizar suas possíveis leituras e perceber como foi sua relação com as escolhas. (Estante)"* , *"Sim. O aplicativo tem as funcionalidades ideais para organizarmos as leituras, tanto como gerenciador do clube, quanto Clubista ou o leitor em si. Achei muito interessante ter todas essas funções em um só lugar, tornando bem mais fácil o gerenciamento dos livros, dos membros, dos encontros, enfim. Bastante útil"* , *"Sim. O aplicativo tem as funcionalidades ideais para organizarmos as leituras, tanto como gerenciador do clube, quanto Clubista ou o leitor em si. Achei muito interessante ter todas essas funções em um só lugar, tornando bem mais fácil o gerenciamento dos livros, dos membros, dos encontros, enfim. Bastante útil."* , *"Acredito que sim, é fácil de colocar os livros na estante e criar clubes"* , *"Não foi intuitivo cadastrar meus livros lidos"* , *"Apenas não consegui visualizar como funcionam as tarefas de Encontros e Votação. Seria interessante o cadastro de informações para verificar como funcionam. Fiquei também interessada em saber como funciona a criação de um clube no App"*

5.3 Métricas e estatísticas de uso:

A seguir, será mostrado um breve resumo de valores obtidos visualizando a base de dados. Ao todos, foram criados 21 novas contas de usuários habilitados para os testes. Obviamente esse número é bem pequeno, entretanto como o aplicativo está em fase inicial, é de extrema importância que os sejam realizados com poucos usuários para obter melhores análises e garantir que funcione adequadamente antes de disponibilizar para um público maior.

1. Três novos clubes.
2. 28 livros adicionados às estantes dos usuários.
3. Três eleições criadas.
4. Cinco candidatos criados.
5. Dois votos realizados.
6. Cinco comentário realizados.

7. Zero livro adicionado à estante dos clubes.

Pelos dados observados acima, é possível perceber que os usuários estão explorando as funcionalidades. A quantidade de livros adicionadas as estantes pessoas sugere que há uma interesse nessa funcionalidade, além disso observa-se uma partição na criação de eleições, de candidatos e na escolha desses candidatos, o que nos permites deduzir que mesmo funcionalidades complexas, como esta, estão sendo utilizadas. Entretanto não houve adição de livros nos clubes o que pode indicar que os usuários ainda não estão explorando ou utilizando a funcionalidade de adicionar livros às estantes dos clubes ou não estão conseguindo usar. É importante investigar se essas questões estão relacionadas a possíveis erros no código ou a uma interface pouco intuitiva.

5.4 Conclusão

A proposta principal desse trabalho era desenvolver um aplicativo para clubes de leituras e leitores, com o objetivo de proporcionar aos usuários uma experiência aprimorada na organização e participação em clubes de leitura.

Ao longo deste projeto foi percebido a importância de fazer escolhas seguras a partir dos requisitos de um sistema e a evolução constante de acordo com a necessidade e *feedback* dos usuários. Isso ocorre especialmente pelo contato constante com os participantes de clubes de leituras e leitores, que são de fatos os utilizadores do aplicativos. Assim, aliado ao uso de metodologias ágeis foi possível uma adaptação contínua às necessidades dos usuários.

Dessa forma, foram implementadas funcionalidades como a criação e participação em clubes de leitura, adição de livros às estantes pessoais e dos clubes, votação em eleições de livros, entre outras. Ainda, os outros recursos oferecidos, como o acompanhamento do progresso de leitura, registro de comentários e avaliações, fornece aos usuários uma visão abrangente de suas atividades literárias.

Os resultados mostrados no último capítulo demonstram o sucesso e a eficácia do aplicativo e a seguridade em expandir para um público maior. De acordo com alguns usuários o aplicativo tem se mostrado promissor no auxílio de gerenciamento de clubes de leituras e leituras, os dados mostram que há uma facilidade na seleção e na discussão de livros, e além disso a visualização de leituras de outros clubes e membros contribui para uma experiência mais enriquecedora no mundo da leitura compartilhada.

Desse modo, pode-se dizer que o trabalho aqui desenvolvido conseguiu realizar seu objetivo principal criar um aplicativo para gerenciar clube de leituras e leituras. Mas de maneira geral, o aplicativo atinge outros objetivos que são tão importantes quantos o principal, detre os quais, pode se destacar: contribuir para a promoção da leitura, da formação de comunidades literárias e incentivar o compartilhamento de conhecimentos adquiridos pela leitura.

Além disso, o aplicativo pode vim a ser utilizado para criar mecanismo de apoio e sustentar políticas públicas no processo de incentivo, fiscalização, adoção de mediadas para a ampliação da leitura em diversos espaços e comunidades. Dessa forma, acredita-se que o desenvolvimento deste aplicativo permitiu um avanço na criação de comunidades literárias e no compartilhamento de conhecimento.

5.5 Recomendações para trabalhos futuros

No decorrer deste trabalho, a parte mais desafiadora sem dúvida foi a criação de uma interface *UI* intuitiva e que fosse elegante mas de fácil manipulação, especialmente

pela pouco espaço de tempo para concluir o trabalho e pela complexidade deste trabalho.

Portanto, recomenda-se que o trabalho seja continuado com o auxílio de uma interface produzida por design ou pessoas relacionadas à área. A utilização de uma interface produzida ou avaliada por indivíduo com conhecimentos e boas práticas pode ser extremamente benéfica para o projeto.

Durante o desenvolvimento do aplicativo, contamos com a colaboração de colegas para ampliar e criar funcionalidades relacionadas a aspectos colaborativos de atividades e necessidades de grupo. Além disso, também contamos com a contribuição de outro colega para a realização de testes automatizadas e cobrir o máximo da aplicação. A colaboração dos colegas foi fundamental para enriquecer o aplicativo e garantir sua qualidade e eficácia.

Além disso, foi identificada uma demanda por parte dos usuários em transformar o aplicativo em uma espécie de rede social, incluindo recursos como uma linha do tempo (*timeline*) e a possibilidade de compartilhar fotos. A capacidade de compartilhar as leituras e acompanhar o que outros usuários estão lendo, assim como a forma como estão explorando os livros, pode ser extremamente valiosa para incentivar e engajar outros leitores.

Além disso, o trabalho se mostra abre espaço para a implementação de um sistema de recomendação de livros colaborativo, a integração de um sistema de recomendação colaborativo pode aprimorar ainda mais a experiência do usuário no aplicativo, essa funcionalidade permitiria uma melhor descoberta de novos livros e na ampliação de horizonte. E para isso, cria-se uma oportunidade de pesquisa e estudo nas áreas de Data Science, Inteligência Artificial e Aprendizado de Máquina.

Referências

ALCÂNTARA, E. M. d. S. *Clubes de leitura: proposta de um aplicativo para gerenciamento e compartilhamento de leituras literárias e incentivo à leitura*. Tese (Dissertação) — Universidade Federal do Maranhão, 2022. Citado 4 vezes nas páginas 4, 5, 11 e 12.

BOCCIA, A. S. Clubes de leitura: a construção de sentidos em situações de leitura colaborativa. *2236-5729*, volume 2, p. 97–113, 05 2012. Citado na página 12.

COSSON, R.; LUCENA, J. M. d. *Práticas de letramento literário na escola: propostas para o ensino básico*. UFPB: Editora UFPB, 2022. ISBN 978-65-5942-155-8. Citado na página 13.

EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. [S.l.]: Addison-Wesley, 2004. Citado na página 27.

FERREIRA, L. *Clubes de leitura online: conheça 14 opções e saiba como funcionam*. 2021. (Accessed on 06/22/2023). Disponível em: <https://janelasabertas.com/2021/07/29/clubes-de-leitura-online/>. Citado na página 13.

FGV. *Brasil tem 424 milhões de dispositivos digitais em uso, revela a 31ª Pesquisa Anual do FGVcia*. 2022. Disponível em: <https://portal.fgv.br/noticias/brasil-tem-424-milhoes-dispositivos-digitais-uso-revela-31a-pesquisa-anual-fgvcia>. Citado na página 11.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine, 2000. Citado 2 vezes nas páginas 14 e 15.

FORBES. *Top 10 Apps By Downloads And Revenue Q2 2021: Report*. 2021. Disponível em: <https://www.forbes.com/sites/johnkoetsier/2021/07/15/top-10-apps-by-downloads-and-revenue-q2-2021-report/?utm_campaign=forbes&utm_source=twitter&utm_medium=social&utm_term=Carrie&sh=7fed71463295>. Citado na página 11.

Instituto Pró-Livro. *Retratos da leitura no Brasil*. 5th. ed. São Paulo: IPL, 2020. Disponível em: <https://www.prolivro.org.br/wp-content/uploads/2020/12/5a_edicao_Retratos_da_Leitura-_IPL_dez2020-compactado.pdf>. Citado na página 11.

MARTIN, R. *The Clean Architecture*. 2012. Acesso em: 14 de Julho de 2022. Disponível em: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. Citado na página 17.

MARTIN, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston, MA: Prentice Hall, 2017. (Robert C. Martin Series). ISBN 978-0-13-449416-6. Disponível em: <https://www.safaribooksonline.com/library/view/clean-architecture-a/9780134494272/>. Citado na página 27.

SOUZA, R. J.; COSSON, R. *Letramento literário: uma proposta para a sala de aula*. São Paulo: Cultura Acadêmica, 2011. v. 2. 101-107 p. Citado na página 13.

Anexos

ANEXO A – Certificado de Registro de Programa de Computador

REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DA ECONOMIA
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS INTEGRADOS

Certificado de Registro de Programa de Computador

Processo Nº: **BR512022003624-0**

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1° de janeiro subsequente à data de 04/11/2022, em conformidade com o §2°, art. 2° da Lei 9.609, de 19 de Fevereiro de 1998.

Título: App Clubes de Leitura

Data de publicação: 04/11/2022

Data de criação: 02/05/2022

Titular(es): UNIVERSIDADE FEDERAL DO MARANHÃO

Autor(es): DAVI VIANA DOS SANTOS; PATRÍCIA DE MARIA SILVA FIGUEIREDO; ERLANE MARIA DE SOUSA ALCANTARA; MATEUS DA SILVA OLIVEIRA; EMANUELLE DA LUZ LEMOS

Linguagem: OUTROS

Campo de aplicação: CO-05; ED-04; ED-06

Tipo de programa: AP-01

Algoritmo hash: SHA-512

Resumo digital hash:
f6ecd1b77c100dd0cc85019ffe7b54ff24f29a80d9dc9ef740af45243b49d24dbaf85797381cd454077b3e0ce7d46862e2685992a8d91830726578abc6fc019e

Expedido em: 03/01/2023

Aprovado por:
Joelson Gomes Pequeno
Chefe Substituto da DIPTO - PORTARIA/INPI/DIRPA Nº 02, DE 10 DE FEVEREIRO DE 2021

ANEXO B – Modelo entidade relacionamento completo

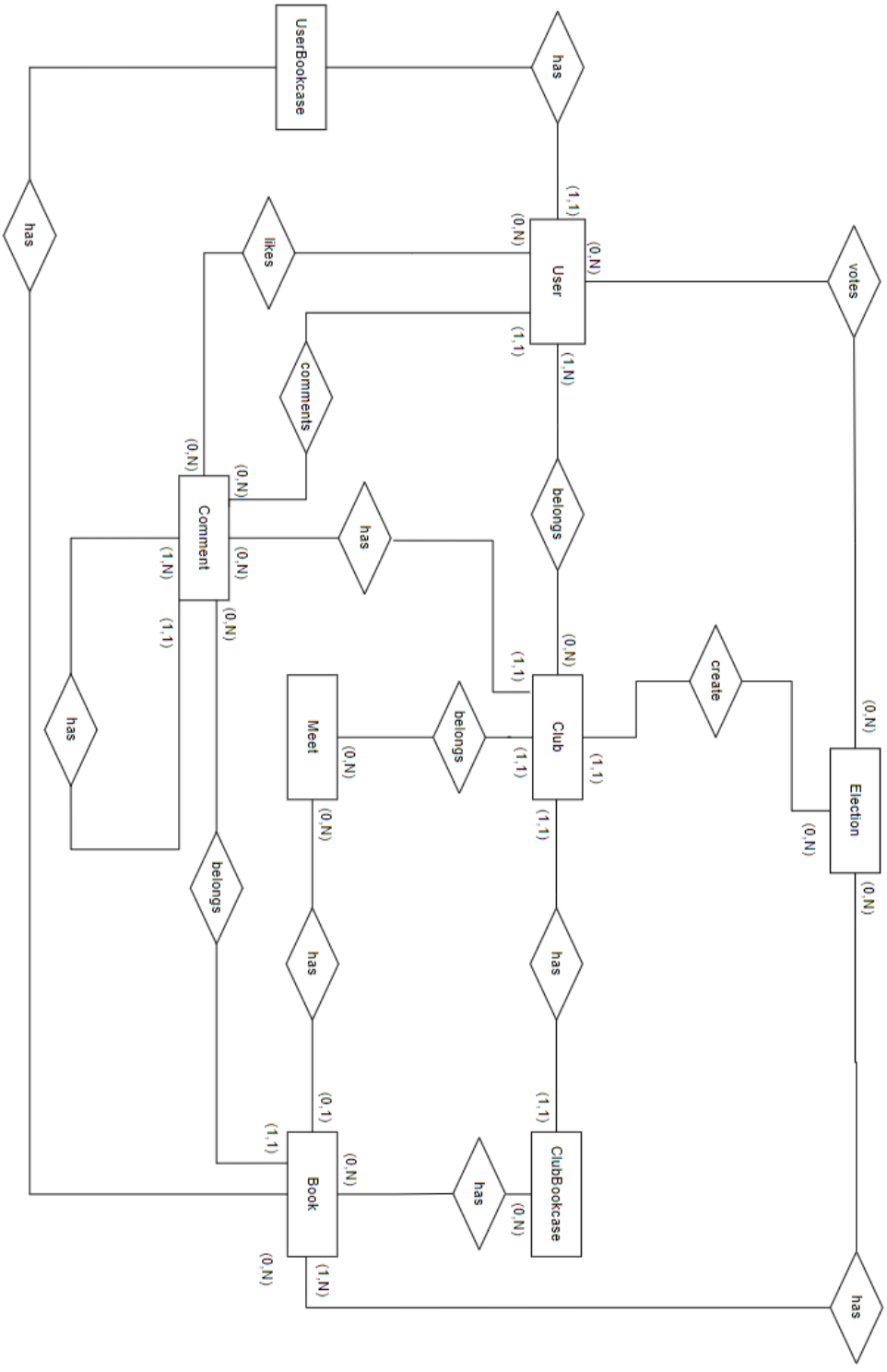


Figura 9 – Modelo entidade relacionamento completo

ANEXO C – Modelo de dados

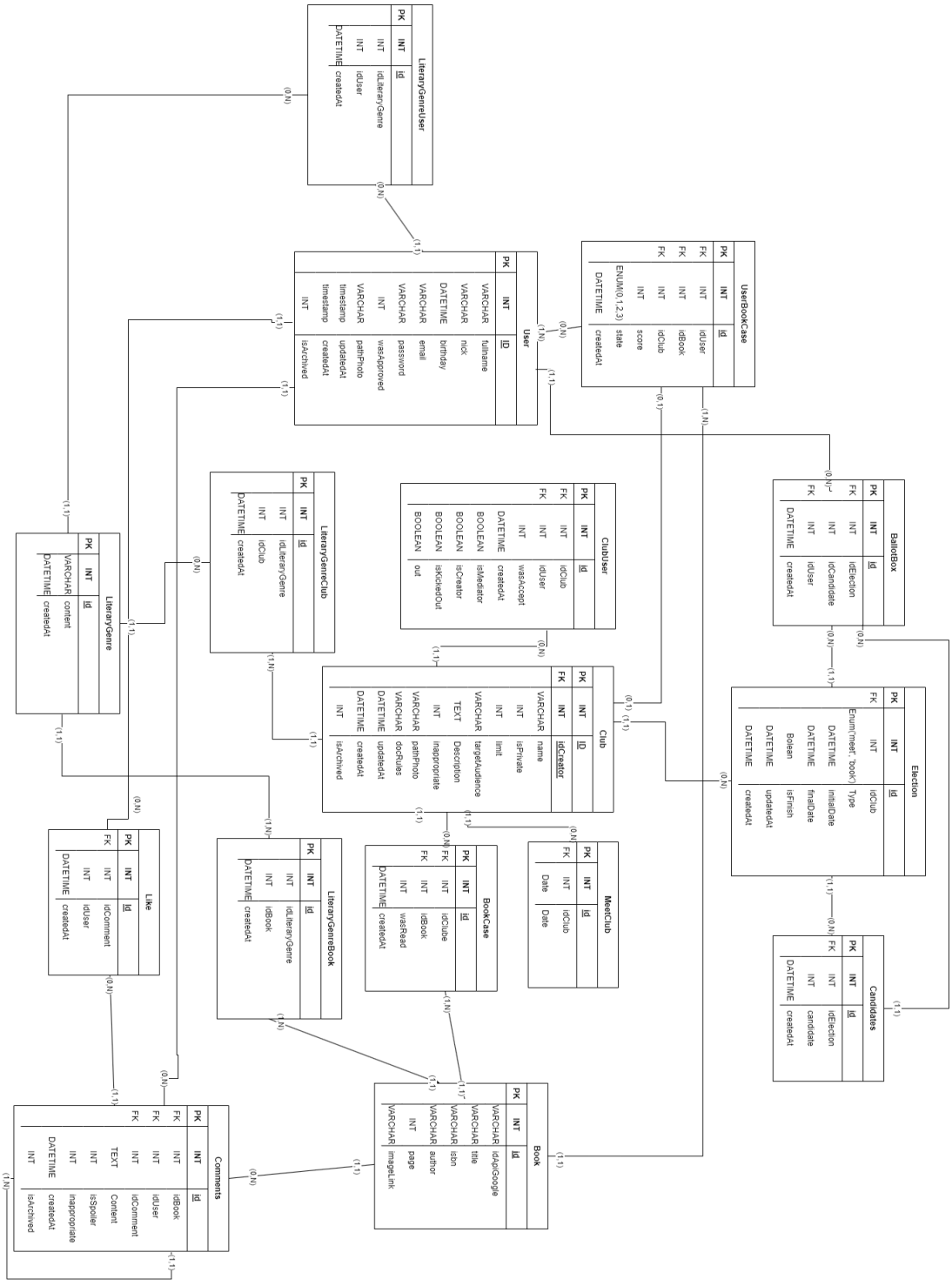


Figura 10 – Modelo de dados

ANEXO D – Casos de usos

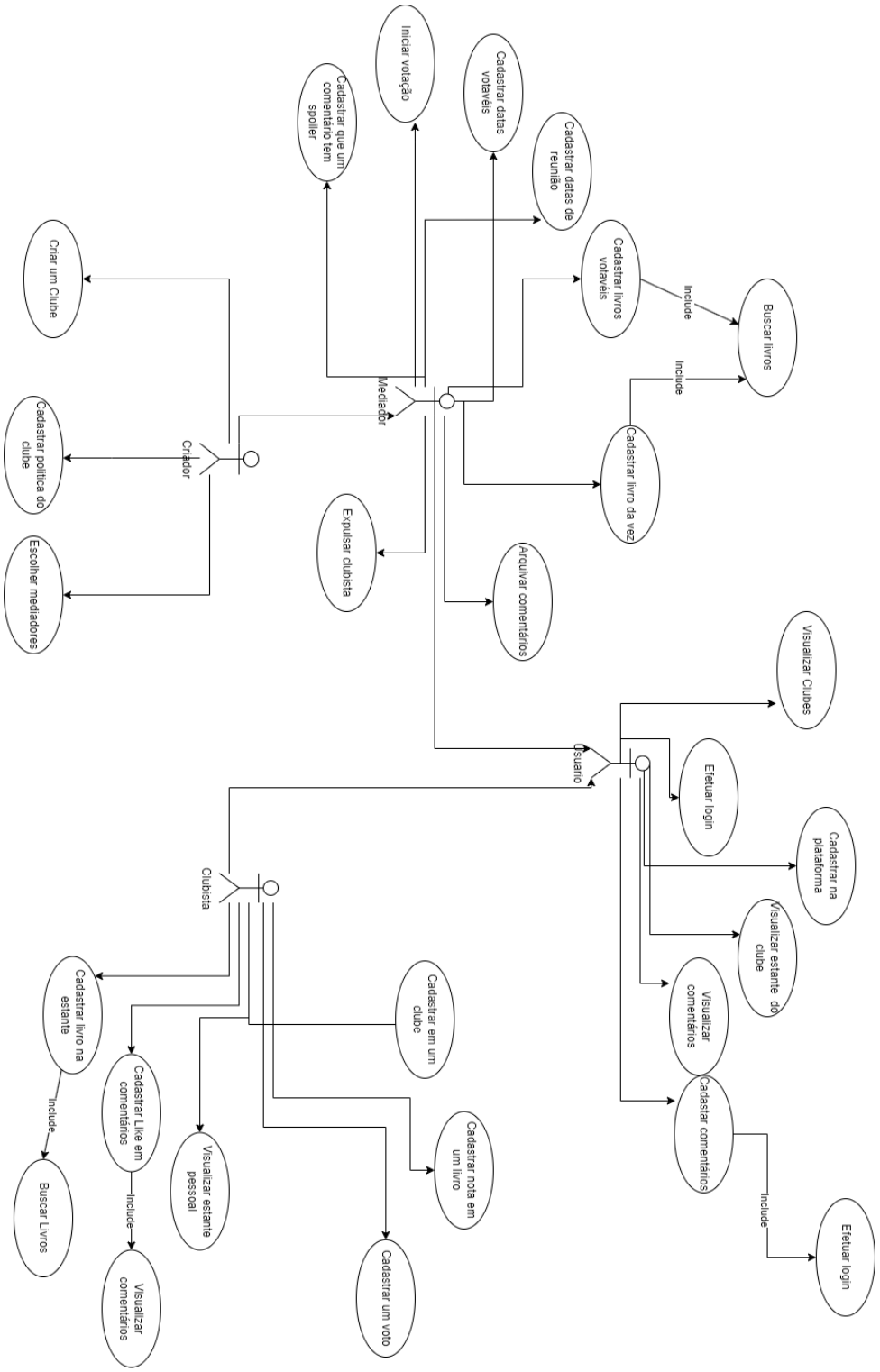


Figura 11 – Casos de usos

ANEXO E – Classe concreta que implementa o repository pattern

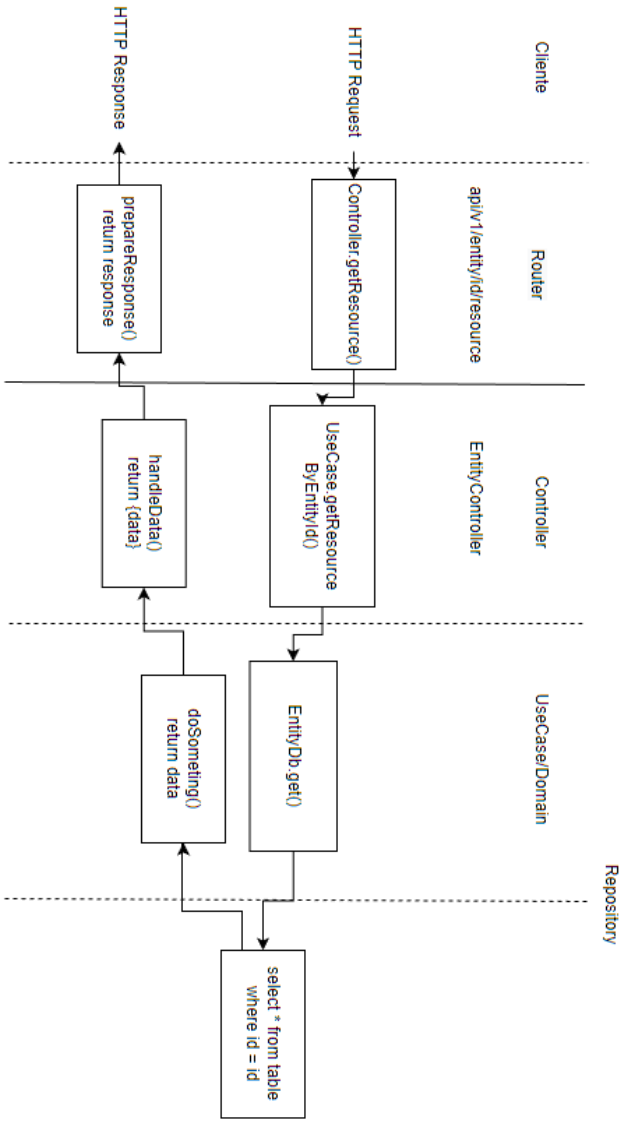
Código 11 – Classe Knex que implementa a classe generica

```
class KnexGenericRepository<T> implements GenericRepositoryInterface<T>{
  protected repository: Table;
  protected query: any;

  constructor(repository: Table) {
    this.repository = repository;
    this.query = knex.from(repository)
  }

  async get(id: number): Promise<T> {
  return await this.query
    .select('*')
    .where(
      'id', id
    )
    .first()
  }

  async getAll(): Promise<T[]> {
      return await this.query
        .select('*')
  }

  async create(data: T) {
      let query = this.query
        .insert(data)
        .toSQL();

      const sql = query.sql.replace("insert", 'insert ignore');

      return await knex.raw(sql, query.bindings)
        .then(async (response: T[]) => {
          const id = response[0]
          const data_ = await knex.from(this.repository)
            .select('*').where(
              'id', id
            )
            .first()

```

```
40            return data_
41          })
42    }
43
44    async update(id: number, data: T | Partial<T>) {
45        return await this.query
46          .update(data)
47          .where({
48            id
49          })
50    }
51
52    async delete(id: number) {
53        return await this.query
54          .delete()
55          .where({
56            id
57          })
58    }
59 }
```

ANEXO F – Arquitetura completa do servidor

Figura 12 – *Arquitetura completa do servidor*



Fonte: Elaborada pelo autor.