UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

Documento de Arquitetura de Software

Versão 2.0

Alexson Almeida - 508294

Daniel Almeida de Freitas - 508077

José Vitor - 509295

Kaio Portela - 495707

Aplicativo de Trocas de Livros	Versão: 2.0
Documento de Arquitetura de Software	Data: 03/09/2023
APP_Doc_Arq_V.2.0	

Histórico de Revisões

Data	Versão	Descrição	Autor
03/09/2023	1.0	Início do documento com sessões importantes.	Daniel Almeida

Indice Analitico

Documento de Arquitetura de Software

1. Introdução

Para se obter uma visão arquitetural do aplicativo de troca de livros ShareBook, este documento vem para elucidar os diversos aspectos do sistema. Suas colocações são para transmitir as escolhas dadas em termos arquitetônicos, o qual a equipe adotou para o desenvolvimento do mesmo.

O objetivo desse documento é classificar de forma clara a arquitetura, o qual a equipe estabeleceu para o desenvolvimento do ShareBook APLICATIVO DE TROCA DE LIVROS, o foco é seguir o desenvolvimento no padrão de arquitetura cliente e servidor uma vez ela se adequa melhor para os requisitos elicitados para a aplicação. Ademais, o lado do servidor - responsável por responder solicitações dos clientes - será construído utilizando a arquitetura clean architecture, baseada em camadas. Por último, o lado do cliente será construído utilizando uma arquitetura Model-View-ViewModel (MVVM), devido ela conseguir separar bem a camada de UI das camadas de dados subjacentes.

No documento está definido o que o ShareBook estará contemplando em termos arquitetônicos, especificando seus casos de usos, visões lógicas, camadas assim como sua implementação e implantação. A equipe se atentou em todas as decisões arquitetônicas presentes neste documento, o que é único para o desenvolvimento do Aplicativo de troca de livros. A aplicação vem se tratando de um ambiente MOBILE, que tende a utilização de um banco de dados o qual estará dentro do ambiente de implementação que é um servidor web, seguindo princípios de boas práticas.

1.1 Finalidade

A finalidade deste documento, é oferecer de modo claro as diversas visões o qual os modelos arquitetônicos que vincula o aplicativo vem a possuir, trazendo consigo as características necessárias para os controles de suas atividades arquitetônicas, assim moldando todo procedimento para o desenvolvimento do sistema.

1.2 Escopo

A arquitetura do aplicativo SahreBook é totalmente fundamentada e detalhada, para moldar uma base para a equipe de desenvolvimento, apresentando como será o comportamento do sistema. Esse documento refere-se às características identificadas no “App_Doc_Riq*” o qual tratam das necessidades e as regras do negócios o qual o ShareBook deve atender, nesse documento contém os registros e tópicos importantes relacionado a arquitetura do sistema, dando diretrizes, e total apoio para o uso de suas tecnologias.

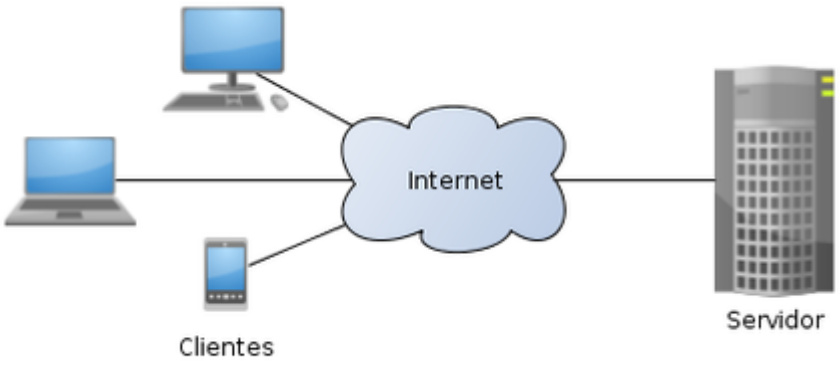
Definições, Acrônimos e Abreviações

Abreviação	Acrônimos	Descrição
MVVM	Model-View-ViewModel	Padrão de arquitetura do Software
API	Application Programming Interface	Conjunto de funções e procedimentos que permitem a integração de um ou mais sistemas.
App_Doc_Req	Aplicativo Documento Requisitos	Identificador do documento de Requisitos do <NOME DO APLICATIVO>
RNF	Requisito Não Funcional	Identificador de um requisito não funcional.
RF	Requisito Funcional	Identificador de um requisito funcional.
SO	Sistema Operacional	Software responsável por alocar e gerenciar recursos de um sistema.

2. Representação Arquitetural

2.1. Arquitetura Cliente e Servidor

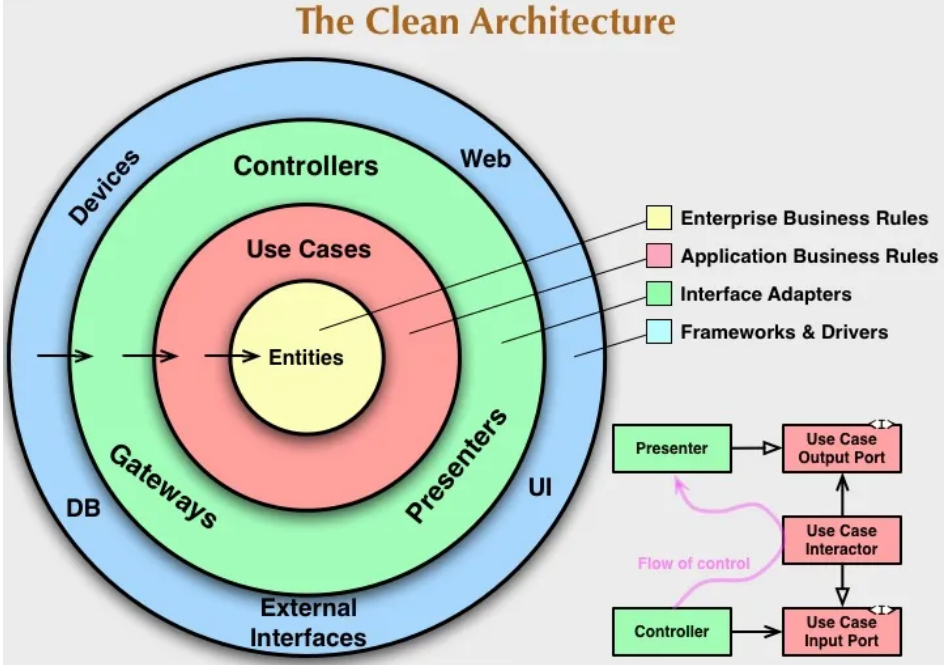
Como dito anteriormente o projeto será desenvolvido usando uma arquitetura cliente e servidor que é dividida em mais dois tipos de arquiteturas subsequentes, sendo conectadas por meio de uma API. A arquitetura cliente servidor é uma arquitetura de aplicação distribuída, ou seja, na rede existem os fornecedores de recursos ou serviços a rede, que são chamados de servidores, e existem os requerentes dos recursos ou serviços, denominados clientes. Essa escolha se dá principalmente por esse tipo de arquitetura se adequar bem aos requisitos do projeto e por ser esse modelo permite a separação das responsabilidades entre o cliente e o servidor, possibilitando um melhor controle sobre o fluxo de informações e expansão do sistema no futuro.



2.2. Clean Architecture (Lado do Servidor)

Uma vez que estamos seguindo uma arquitetura cliente-servidor, temos a oportunidade de separar nossa aplicação em dois fluxos diferentes, por consequência, também podemos escolher tecnologias e arquiteturas diferentes para cada fluxo. Sendo assim, o nosso lado do servidor será construído utilizando uma arquitetura baseada em camadas chamada de Clean Architecture. Falando sobre suas camadas, a camada mais interna é chamada de “Entities”, tem como objetivo organizar as classes responsáveis pelas regras de negócio comum a vários sistemas e implementar regras de negócios genéricas. Sua segunda camada é a “Uses Cases” e tem como objetivo implementar regras de negócio específicas do sistema. Na terceira camada de dentro para fora temos a camada de “Controllers”, camada responsável por mediar a interação entre a camada mais externa da arquitetura (sistemas externos) e as camadas centrais (Casos de Uso e Entidades) e implementação dos endpoints da REST da API que iremos criar. Já na camada mais externa temos a camada de “Frameworks Externos”, camada essa onde ficam classes de bibliotecas, frameworks e quaisquer sistemas externos.

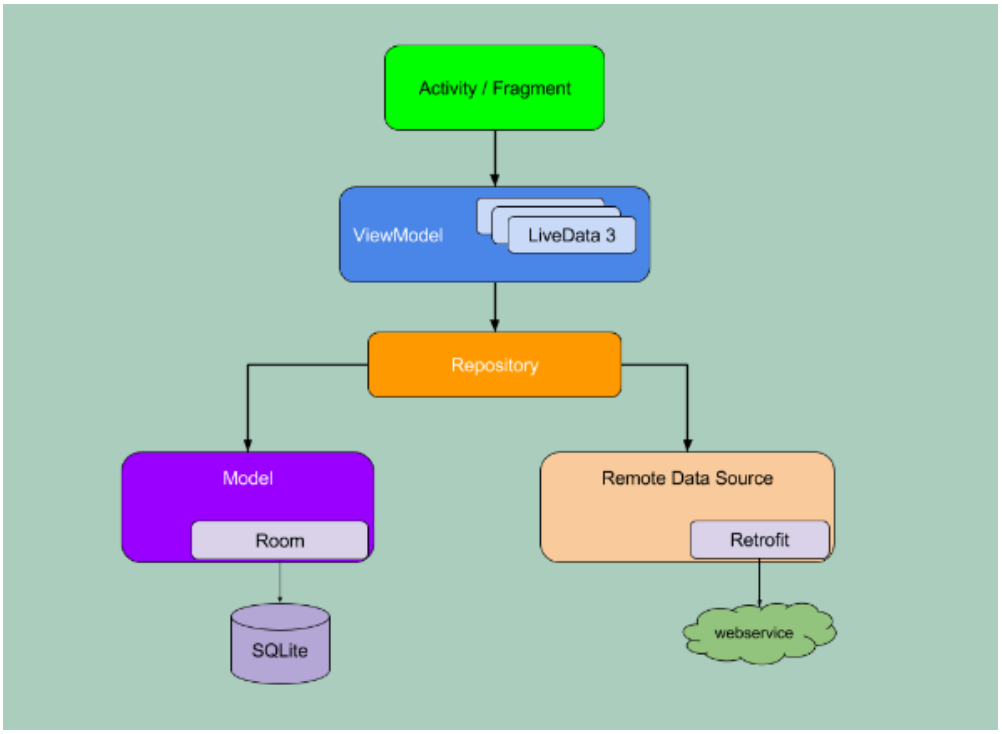
Sendo assim, escolhemos essa arquitetura por ela separar bem as responsabilidades de uma aplicação, além de facilitar o desenvolvimento de códigos, permitir uma melhor manutenção, uma boa atualização e menos dependências no projeto. Ademais, vale ressaltar que outros pontos importantes para a escolha foi o fato dessa arquitetura ser muito bem testável e independente de qualquer banco de dados ou agente externo.



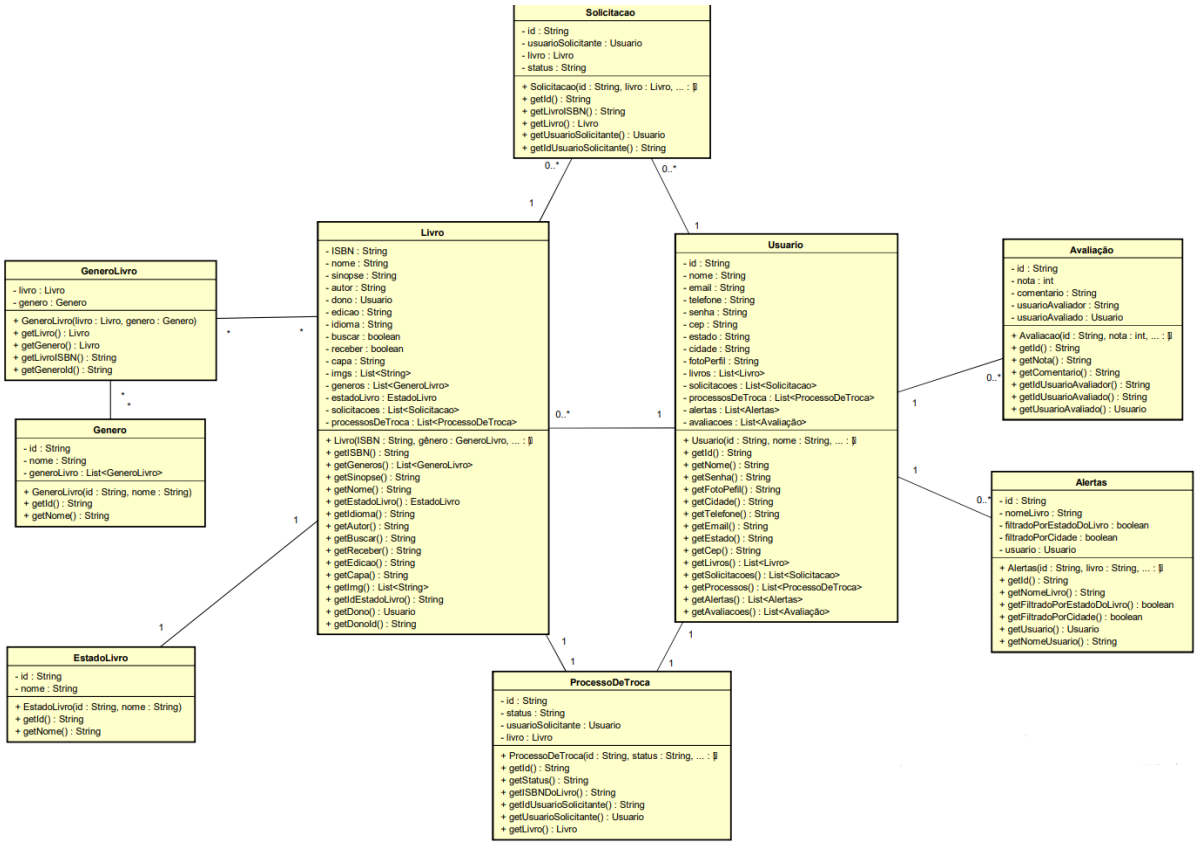
2.3. Arquitetura Android MVVM (Lado do Cliente)

Seguindo a arquitetura cliente servidor, o lado do cliente será desenvolvido baseado na arquitetura MVVM - arquitetura Android baseada em responsabilidades - que visa separar a camada de UI das camadas de dados subjacentes. A primeira camada dessa arquitetura é a “Activity/ Fragment”, camada essa responsável pela visualização e interações com o usuário, a segunda é a camada ViewModel, responsável pelas regras de negócios da aplicação. Já a última camada é responsável pelos dados da aplicação, chamada por repository sua finalidade é fazer acessos a serviços externos por meio do retrofit ou acessar um banco de dados interno, exemplo o SQLite.

Sendo assim, escolhemos essa arquitetura por ela separar as responsabilidades da nossa camada de UI das camada de lógica e acesso a dados da aplicação. Além disso, esse tipo de arquitetura aumenta a testabilidade do aplicativo, uma vez que a separação dessas responsabilidades diminui o acoplamento entre o código escrito pela equipe e o código do SO, ou seja, diminui as chances seu de algum teste ficar impossibilitado de ser executado na JVM por existir alguma dependência que necessite de classes do SO para ser executada.



3. Visão de Processos



O diagrama de classes descreve como são compostas em termos de código as entidades que irão compor o sistema, fornecendo atributos, métodos e relacionamentos entre as entidades. Os métodos presentes nas classes consistem principalmente em construtores, getters e setters, pois os métodos de manipulação e comunicação são fornecidos por outras entidades gerenciadas pelo framework utilizado. Segue uma descrição de cada uma das classes:

Usuário: classe básica do sistema, representa os leitores que irão acessar a aplicação e trocar seus livros. Nessa classe, temos os atributos básicos para possibilitar cadastro e manutenção dos dados no banco como id, email e senha. Além disso, para gerenciar a interação entre os usuários e fornecer as informações necessárias para decidir acerca do processo de troca, temos dados como telefone, cep, cidade, estado e uma foto de perfil. Fechamos a classe com as listas de outras entidades que os usuários estão relacionados, que são: livros publicados, solicitações de troca, processos de troca, avaliações de outros usuários sobre ele e alertas que ele criou.

Livro: outra que consiste em uma das principais classes do sistema. Temos dados de identificação do livro, como o ISBN, um código que funciona como um identificador do livro, o título, autor, sinopse, edição, a capa do livro, algumas fotos do estado dele, o próprio estado que o usuário acredita que o livro está, uma lista de gêneros para saber em qual ele se situa e o idioma em que está escrito. Temos também o identificador do usuário que é dono dele e listas de solicitações de troca desse livro e processos de troca que ele participa.

Solicitação: classe que descreve os pedidos de troca que usuários fizeram para aquele livro. Aqui temos, como de praste, um id para tratar cada solicitação como única, o usuário que fez aquela solicitação, o livro que ele solicitou e o status em que ela está.

Processo de troca: classe que descreve o processo no qual um usuário aceitou um pedido de troca e agora estão aguardando até a troca ser realizada efetivamente. Temos um id para identificar o processo individualmente, o livro que foi trocado, o usuário que solicitou a troca e o status em que ela se encontra.

Avaliação: classe que descreve as avaliações que os usuários fizeram um do outro. Temos o id da avaliação, a nota que o usuário deu para o outro, um texto de comentário, o usuário que foi avaliado e o id do usuário avaliador.

Alerta: consiste nos alertas de determinados livros criados pelo usuário, para o caso dele querer algum livro específico que não esteja disponível para troca atualmente no sistema. Temos como atributos o id de identificação do alerta, o nome do livro que o usuário gostaria de ser notificado quando lançar, o usuário que aquele alerta pertence e informações sobre a preferência do usuário, se ele precisa ser filtrado pelo estado do livro ou pela cidade.

Gênero: classe que descreve os possíveis gêneros de livros. Temos o id do gênero, o nome dele (terror, aventura etc.), e uma lista de GêneroLivros que vamos explicar a seguir:

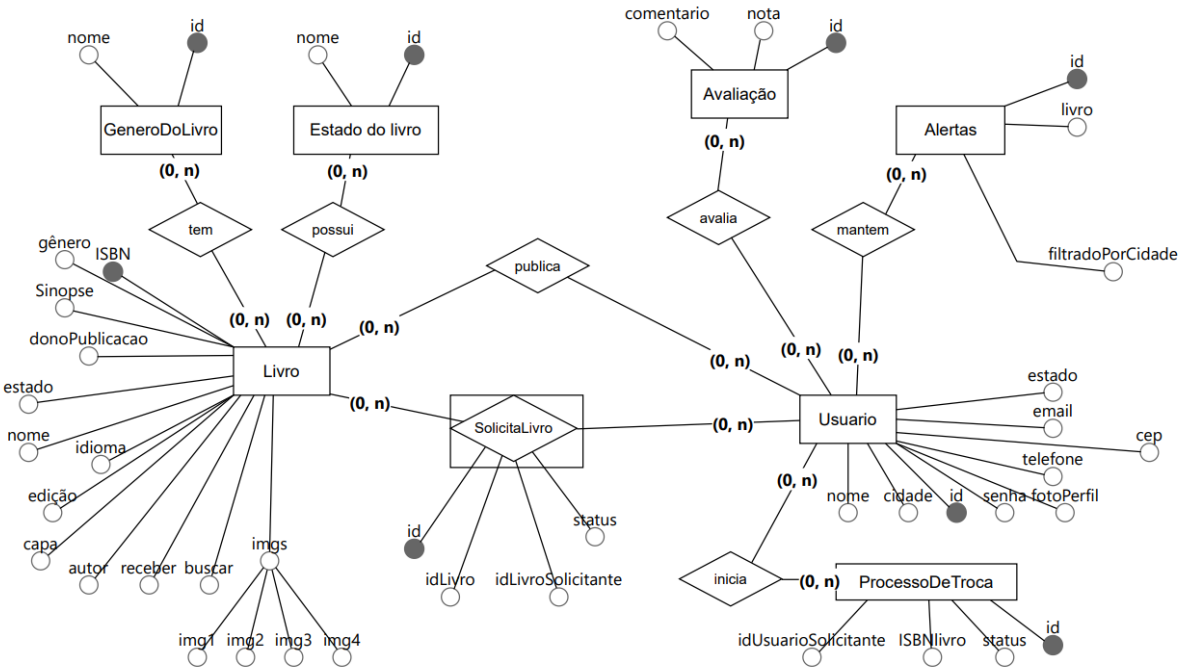
GeneroLivro: uma classe intermediária que auxilia a comunicação do relacionamento M x N entre o livro e seu gênero. Temos o gênero e o livro em específico como atributos dessa classe.

EstadoLivro: representa os possíveis estados que um livro pode estar (novo, marcado, muito marcado etc). Temos um id para marcar cada estado individualmente e o nome daquele estado.

4. Visão de Processos

4.1. Visão de Dados

Nota: A descrição da modelagem conceitual funcionará, também, para a lógica.



A modelagem conceitual e lógica do banco de dados descrevem como o banco de será estruturado em termos de entidades e relacionamentos entre elas. A modelagem conceitual é apenas mais geral e descreve alguns aspectos a mais. De uma maneira geral, no sharebook, temos o banco estruturado da seguinte forma:

Usuário: tabela que guarda e gerencia os usuários do sistema. Nessa tabela, temos como dados o id do usuário, que será a chave primária da tabela e é do tipo uuid, ou seja, string e dados relacionados à identificação do usuário e da criação do seu perfil, como nome, cidade, telefone, cep, cidade e estado onde o usuário mora, ambas do tipo string, tendo também a foto de perfil que é armazenada como uma string com a url de armazenamento do arquivo em um serviço de armazenamento à parte, como EC2 e Supabase. Temos também dados de criação e gerenciamento da conta do usuário, consistindo em um email e uma senha, ambas do tipo string. Também é importante citar que a tabela usuário se relaciona com outras tabelas. Os usuários mantém seus alertas de notificação de livros, iniciam processos de troca, publicam livros para troca, tem avaliações e se relacionam com a tabela livro através da entidade associativa solicita_livro, que representa os pedidos de troca que os usuários fazem a outros, também podendo iniciar um processo de troca através da confirmação de uma solicitação de troca. Nesses casos, temos uma lista com cada uma dessas entidades para cada usuário. Por fim, resta citar que, dos atributos presentes nessa tabela, apenas a foto do perfil é passível de não existir, lembrando, claro, que as listas podem estar vazias.

Livro: a tabela que armazena os livros que serão manipulados entre os usuários conforme forem sendo realizadas trocas. Aqui temos o ISBN do livro, que funciona como chave primária da tabela, e é do tipo string.

Temos também dados relacionados à identificação do livro, como: título, autor, sinopse, edição, idioma que o livro está e uma foto que é a capa do livro, ambas do tipo string, lembrando que a foto é uma url. Além disso, guardamos o usuário que publicou aquele livro, uma lista de imagens que mostram visualmente como está o livro e são strings que representam as urls, um dado que é o estado do livro, que é instância de outra tabela e uma lista de gêneros, que estão também armazenados em outra tabela. Também há informações daquele livro sobre se, quando o usuário o publicou para troca, ele estaria disponível para ir buscar a depender da localização ou se ele precisa que venham buscar, consistindo em dados do tipo boolean. Por último, é importante citar que, como dito anteriormente, os livros se relacionam com os usuários através do ato de publicação dos mesmos e da solicitação de trocas. Nessa tabela, apenas a sinopse é passível de não existir

Solicitacao: tabela de relacionamento entre as tabelas usuário e livro. Representam os pedidos de troca que um usuário faz a outros. São armazenados aqui o id do usuário que solicitou a troca, o isbn do livro que ele quer trocar por um dos seus, o isbn do livro que ele escolheu para trocar, o status de como está a solicitação atualmente e o id daquela solicitação, ambas do tipo string.

ProcessoDeTroca: tabela que representa o processo de troca, uma vez que a solicitação já foi aceita. Temos como atributos o id daquele processo, o isbn do livro a ser trocado, o id do usuário que solicitou a troca e o status da troca. Ambos, mais uma vez, do tipo string.

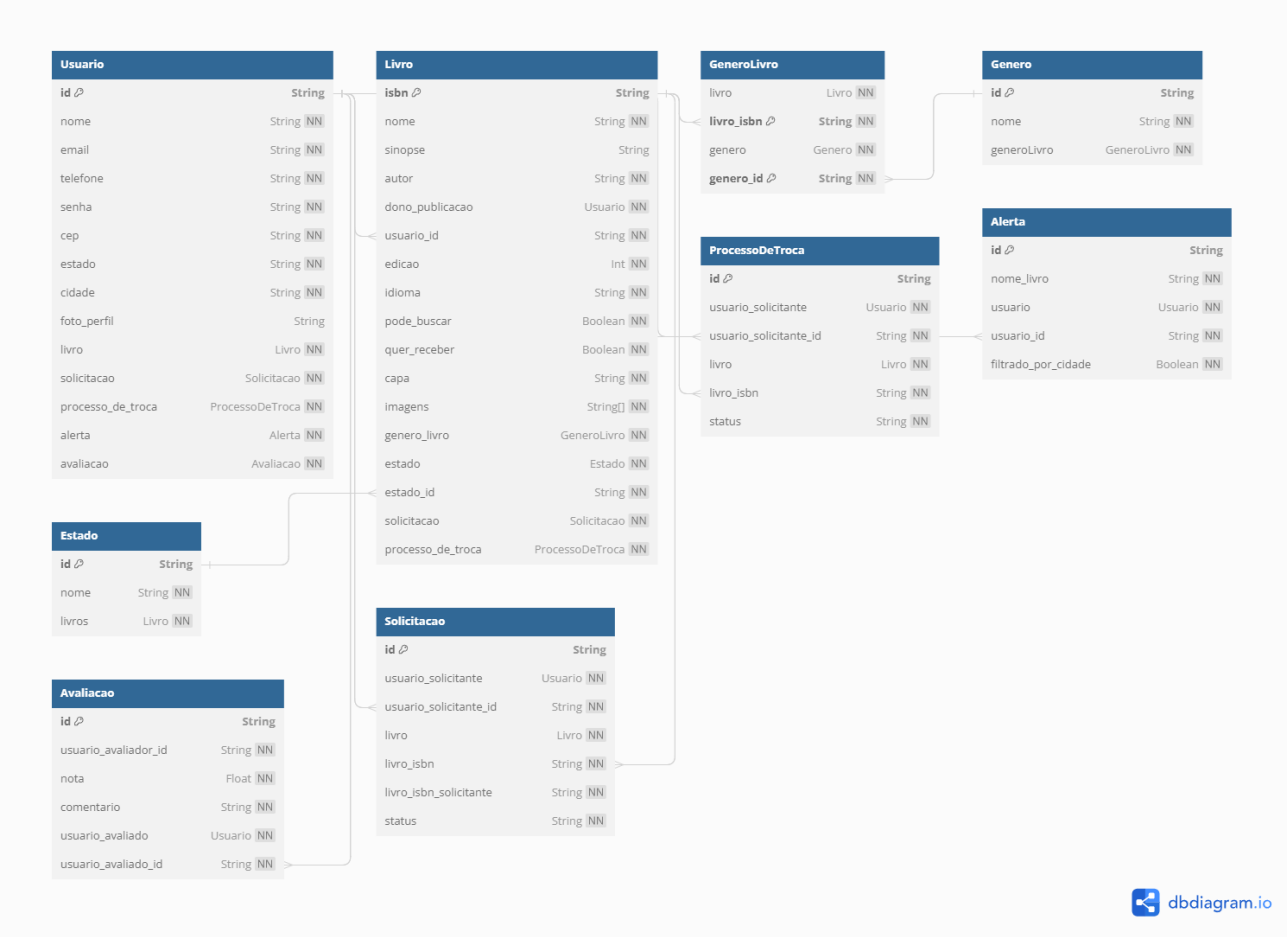
Alerta: consiste nos alertas de notificações de livros que o usuário criou para si. Aqui, temos como atributos um id para o alerta, o id do usuário e o nome do livro para o alerta, esses todos do tipo string. Além disso, temos um atributo boolean que representa se o usuário quer que o alerta filtre os resultados pela cidade em que ele mora.

Avaliação: tabela que representa as avaliações de um usuário que os outros fizeram. Temos como atributos o id daquela avaliação, a nota que o usuário avaliador atribuiu, um texto de comentário e os ids do usuário que foi avaliado e o que avaliou.

EstadoDoLivro: aqui é onde é representado os estados possíveis em que se encontram os livros publicados pelos usuários. Temos o id de cada estado e um nome para o estados como atributos, ambos do tipo string. Além disso, temos uma lista de livros que se encontram naquele estado.

Genero: Tabela que armazena os gêneros dos livros presentes na aplicação. Temos como atributos o id de cada gênero e o nome do gênero do tipo string, e uma lista de GeneroLivro para listar todos os livros que pertencem àquele gênero.

GeneroLivro: tabela que faz a conexão entre os livros e os tem géneros. Temos como atributos os isbn de cada livro e o id de cada gênero dele, ambos sendo do tipo string e constituindo a chave primária dele.



5. Tecnologias e Ferramentas

Levando em consideração os modelos de arquiteturas mostrados anteriormente, a equipe definiu as seguintes tecnologias e ferramentas para serem usadas durante o processo de desenvolvimento.

Nome	Descrição	Onde será usado(a)	Justificativa para o uso
Node.Js	Plataforma que interpreta o Javascript no lado do servidor.	Servidor	Familiaridade do time com a linguagem de programação JavaScript e experiência em construções de API REST utilizando Node.Js
NestJS	Framework para criar aplicativos baseado em	Servidor	Suporte para a criação de API REST, fácil integração com diferentes bancos de dados e ORMs,

	Node.Js para o lado do servidor.		e possui uma arquitetura escalável baseada em conceitos modulares do Angular, que se encaixa perfeitamente com a arquitetura proposta para o projeto.
Typescript	Linguagem de programação fortemente tipada baseada em JavaScript.	Servidor	Aumenta a produtividade no desenvolvimento e previne erros de tipagens que possam vir a ter com o Javascript. Ademais, o typescript já vem por padrão no NestJs.
PostgreSQl	Banco de dados relacional.	Servidor	Se adequa melhor aos requisitos do projeto por ser relacional. Ademais, o time tem mais experiência em trabalhar com o Postgres.
Prisma	ORM para acesso ao banco de dados.	Servidor	Facilita o acesso ao banco de dados, tirando a necessidade de escrever queries em SQL puro.
Jest	Framework de testes baseado em JavaScript.	Servidor	Fornece uma estrutura de testes sem a necessidade de muitas configurações, com bom desempenho e boas opções de análise.
Kotlin	Linguagem de programação para o desenvolvimento de aplicativos Android.	Cliente	É mais performático em relação ao java e é uma das restrições da cadeira de mobile.
Jetpack Compose	Kit de ferramentas moderno recomendado pelo Android para criar IUs nativas. Ele simplifica e acelera o desenvolvimento da IU no Android.	Cliente	É mais performático em questões de renderizações e é mais flexível que o XML em relação a construção de interfaces.