

# DOCUMENTAÇÃO DA APS GERAL



## Documentação do Projeto de Sistema de Aluguel de Salas

### Visão Geral

Este projeto é uma aplicação web desenvolvida para gerenciar o aluguel de salas. Usuários podem visualizar salas disponíveis, alugar e desalugar salas. O sistema também suporta funcionalidades como autenticação de usuários, relatório de uso das salas e visualizações dinâmicas baseadas no estado da disponibilidade das salas.

### Tecnologias Utilizadas

O projeto é dividido em duas partes principais: o frontend (cliente) e o backend (servidor). Abaixo estão as tecnologias específicas utilizadas em cada parte.

# Frontend



## React

**O que é:** React é uma biblioteca de JavaScript desenvolvida pelo Facebook para construir interfaces de usuário de forma declarativa, eficiente e flexível. Ela permite a criação de componentes de UI reutilizáveis, que gerenciam seu próprio estado.

Como está sendo usada:

**Componentes:** Neste projeto, React é usado para definir e gerenciar os componentes visuais, como páginas de login, listagem de salas, e navegação.

**Estado e Ciclo de Vida:** React gerencia o estado (dados que variam ao longo do tempo) de cada componente e o ciclo de vida desses componentes, permitindo que a interface do usuário reaja a mudanças de maneira eficiente.

## Axios

**O que é:** Axios é uma biblioteca cliente HTTP baseada em promessas que funciona tanto no navegador quanto em node.js. Facilita o envio de requisições HTTP para APIs externas ou internas e a gestão de respostas.

Como está sendo usada:

**Requisições HTTP:** No projeto, Axios é utilizado para fazer chamadas API ao servidor Express, solicitando dados das salas ou enviando informações de usuário para autenticação e outras operações relacionadas ao usuário e às salas.

**Tratamento de Respostas:** Axios também gerencia as respostas do servidor, permitindo que o front-end atualize a interface com base nos dados recebidos ou trate erros que possam ocorrer durante as requisições.

## React Router

**O que é:** React Router é uma biblioteca para roteamento dinâmico em aplicações React, que permite a navegação entre diferentes componentes sem a necessidade de recarregar a página.

Como está sendo usada:

**Navegação:** Usada para gerenciar as rotas da aplicação, como a transição entre a tela de login, registro e a visualização das salas. Isso ajuda a criar uma experiência de usuário suave e rápida, onde componentes são re-renderizados conforme o usuário navega, sem recarregar a página inteira.

**Roteamento Dinâmico:** Permite que diferentes URLs levem a diferentes partes da aplicação, como detalhes de uma sala específica ou a página de perfil do usuário.

## CSS

**O que é:** CSS (Cascading Style Sheets) é a linguagem usada para descrever a apresentação de documentos HTML. É crucial para definir o layout, cores, fontes e outros aspectos visuais de uma página web.

Como está sendo usada:

**Estilização:** No projeto, CSS é utilizado para definir o estilo visual dos componentes React, incluindo cores, alinhamentos, distribuição de espaço, efeitos visuais como sombras e transições. Isso garante que a aplicação não apenas funcione bem, mas também tenha uma aparência atrativa e responsiva.

**Responsividade:** CSS também é usado para fazer ajustes de design que respondem a diferentes tamanhos de tela e dispositivos, garantindo que a aplicação seja acessível e utilizável em smartphones, tablets e desktops.

Essas ferramentas juntas formam a base tecnológica do projeto, permitindo que ele seja moderno, eficiente e fácil de usar.

## Backend



### Node.js

**O que é:** Node.js é uma plataforma de desenvolvimento que permite executar JavaScript no servidor. É construído em cima do motor V8 do Google Chrome e é projetado para construir aplicações de rede escaláveis.

Como está sendo usado:

**Servidor:** Node.js é o ambiente de execução para o servidor do seu projeto. Ele lida com solicitações HTTP, gerencia sessões de usuário, e processa lógica de negócios.

**Não-bloqueio:** Node.js opera em um modelo de I/O não-bloqueante que o torna eficiente, especialmente para aplicações web que necessitam processar um grande volume de dados em tempo real.

## Express

**O que é:** Express é um framework para Node.js que oferece um conjunto robusto de recursos para aplicativos web e móveis. É minimalista e flexível, fornecendo ferramentas poderosas para roteamento e middleware com uma execução rápida.

Como está sendo usado:

**Gerenciamento de Rotas:** Express é usado para definir as rotas que respondem a diferentes tipos de solicitações HTTP (GET, POST, DELETE, etc.) para URLs específicos.

**Middleware:** Facilita o uso de middleware para tratar requisições, enviar respostas e manipular erros.

**APIs:** Facilita a criação de APIs para interação entre o frontend e o backend.

## PostgreSQL

**O que é:** PostgreSQL é um poderoso sistema de gerenciamento de banco de dados relacional de código aberto. É conhecido pela sua robustez, performance e compatibilidade com padrões.

Como está sendo usado:

**Armazenamento de Dados:** Utilizado para armazenar todas as informações críticas do seu projeto, incluindo dados de usuários, informações sobre salas e registros de transações de aluguel.

**Relações:** Permite criar relações complexas entre diferentes tipos de dados, o que é ideal para as interconexões entre usuários, salas e aluguéis.

## bcrypt.js

**O que é:** bcrypt.js é uma biblioteca de Node.js para hashing de senhas. Ela ajuda a transformar senhas inseridas por usuários em strings de hash seguras para armazenamento seguro no banco de dados.

Como está sendo usado:

**Segurança:** Antes de armazenar uma senha no banco de dados, bcrypt.js é usado para hashá-la, o que protege as informações do usuário contra acesso não autorizado, mesmo se o banco de dados for comprometido.

## jsonwebtoken (JWT)

**O que é:** JWT é um padrão compacto, seguro para transmitir informações entre partes como um objeto JSON. Essa informação pode ser verificada e confiada porque é assinada digitalmente.

Como está sendo usado:

**Autenticação e Autorização:** Após o login, um token JWT é gerado e enviado ao usuário, que então o utiliza para acessar rotas protegidas no aplicativo. Isso garante que as sessões do usuário sejam mantidas de forma segura e eficiente.

## pg (Node PostgreSQL client)

**O que é:** pg é um cliente PostgreSQL não-bloqueante para Node.js. Ele fornece funcionalidades para conectar e executar consultas ao banco de dados PostgreSQL a partir do Node.js.

Como está sendo usado:

**Interagir com o Banco de Dados:** pg é usado para executar consultas SQL, inserir dados no banco, atualizar registros, e recuperar informações conforme necessário pelo aplicativo.

Essas tecnologias juntas compõem a estrutura do backend do projeto, permitindo que ele funcione de forma eficiente, segura e escalável.

# Banco de dados



## 1. Consulta Aninhada

Uma consulta aninhada em SQL é uma query que usa outras queries como parte de suas condições de filtro. Este tipo de consulta é extremamente útil para realizar buscas complexas que dependem dos resultados de outras consultas menores ou condições múltiplas.

```
SELECT id, nome, localidade, imagem_url, disponibilidade, capacidade, quantidade_alugueis
FROM salas
WHERE nome ILIKE '%termo%' AND localidade = ANY($2::text[])
ORDER BY id
LIMIT $3 OFFSET $4
```

### Explicação Detalhada

1. **SELECT:** Este comando é usado para selecionar dados de uma base de dados. Aqui, ele está sendo usado para selecionar várias colunas (`id, nome, localidade, imagem_url, disponibilidade, capacidade, quantidade_alugueis`) da tabela `salas`.
2. **FROM:** Este comando especifica a tabela da qual os dados devem ser retirados, neste caso, a tabela `salas`.
3. **WHERE:** Este é um filtro que especifica quais linhas devem ser retornadas. É usado para estabelecer as condições que as linhas devem satisfazer para serem selecionadas.

- `nome ILIKE '%termo%'`: Esta condição utiliza o operador `ILIKE`, que é uma versão insensível a maiúsculas e minúsculas do `LIKE`. Aqui, ele busca por salas cujo nome contenha o 'termo' especificado, permitindo flexibilidade na busca.
  - `localidade = ANY($2::text[])`: Esta condição verifica se a coluna `localidade` corresponde a qualquer elemento no array passado como segundo parâmetro. O operador `ANY` permite que a coluna `localidade` seja comparada com uma lista de valores, aumentando a dinâmica da consulta.
  - `disponibilidade = true`: Esta condição filtra as salas que estão disponíveis, ou seja, aquelas cujo campo `disponibilidade` está marcado como `true`.
4. `ORDER BY id`: Esta cláusula é usada para ordenar os resultados pelo campo `id` da tabela `salas`. A ordenação ajuda na organização dos dados retornados, facilitando a navegação e visualização.
  5. `LIMIT $3 OFFSET $4`: Estes comandos controlam a paginação dos resultados.
    - `LIMIT`: Limita o número de linhas retornadas na consulta à quantidade especificada pelo parâmetro `$3`.
    - `OFFSET`: Começa a contar as linhas a partir de um número específico, permitindo "pular" um número definido de linhas antes de começar a retornar as linhas na resposta.

## Consulta Aninhada?

Embora este SQL não contenha um `SELECT` dentro de outro explicitamente, ele se qualifica como uma consulta "aninhada" em um sentido mais amplo devido à complexidade dos filtros aplicados simultaneamente e à integração com parâmetros externos que permitem uma busca dinâmica e multifacetada. Isso cria uma camada de condições que são avaliadas de maneira sobreposta (ou "aninhada") para produzir o conjunto final de resultados, refletindo o conceito de encadeamento de condições que é típico das consultas aninhadas mais complexas.

## 2. Função Simples

Em bancos de dados, uma função (ou função definida pelo usuário) é um bloco de procedimentos que pode executar operações complexas e retornar um resultado. É útil para encapsular lógicas frequentemente utilizadas.

```
CREATE OR REPLACE FUNCTION contar_salas_disponiveis()
RETURNS integer AS $$

DECLARE
    total integer;
BEGIN
    SELECT COUNT(*) INTO total FROM salas WHERE disponibilidade = true;
    RETURN total;
END;
$$ LANGUAGE plpgsql;
```

Esta função retorna o número de salas disponíveis para aluguel.

O código é um exemplo de uma função SQL escrita em PL/pgSQL, a linguagem procedural para o PostgreSQL. Essa função `contar_salas_disponiveis()` é projetada para contar e retornar o número de salas disponíveis em um banco de dados. Vou detalhar cada parte do código:

1. **CREATE OR REPLACE FUNCTION contar\_salas\_disponiveis():**
  - **CREATE OR REPLACE FUNCTION:** Este comando é usado para criar uma nova função ou substituir uma existente se ela já estiver definida. Usar "OR REPLACE" é útil para atualizar uma função sem precisar excluí-la primeiro.
  - **contar\_salas\_disponiveis():** Este é o nome da função. Não há parâmetros passados para esta função, indicado pelos parênteses vazios.
2. **RETURNS integer:**
  - **RETURNS integer:** Esta parte especifica o tipo de dado que a função irá retornar, que neste caso é um inteiro (integer). Este é o total de salas disponíveis que a função calculará.
3. **AS \$\$:**
  - **AS \$\$:** Isso introduz o corpo da função. O \$\$ é um delimitador de string que permite incluir aspas simples na string sem precisar escapá-las. É usado aqui para começar e terminar o bloco de código da função.
4. **Bloco DECLARE:**
  - **DECLARE:** Este comando é usado para declarar variáveis locais que serão usadas na função.
  - **total integer;:** Declara uma variável chamada total do tipo integer. Esta variável será usada para armazenar o resultado intermediário da consulta SQL.
5. **Bloco BEGIN ... END;:**
  - **BEGIN e END;:** Estes comandos delimitam o início e o fim do bloco de execução da função. Todo o código que faz o trabalho da função está entre esses dois comandos.

- `SELECT COUNT(*) INTO total FROM salas WHERE disponibilidade = true;:` Esta é a consulta SQL que faz o trabalho real da função. Ela conta todas as salas (COUNT(\*)) onde a coluna disponibilidade é verdadeira (true). O resultado dessa contagem é armazenado na variável total com o uso do comando INTO.

#### 6. RETURN total;:

- `RETURN total;:` Este comando retorna o valor armazenado na variável total. Como a função é definida para retornar um integer, e total é um integer, isso completa a função retornando o número de salas disponíveis.

#### 7. \$\$ LANGUAGE plpgsql;:

- `$$:` Encerra o bloco de código iniciado após o AS \$\$.
- `LANGUAGE plpgsql;:` Especifica a linguagem na qual a função está escrita, que neste caso é PL/pgSQL. Esta é uma linguagem procedural padrão para PostgreSQL, usada para criar funções mais complexas que simples consultas SQL.

## 3. View

Uma view é uma tabela virtual baseada no resultado de uma consulta SQL. Ela é utilizada para simplificar consultas complexas, encapsular lógicas de acesso aos dados ou proteger dados sensíveis ao limitar o acesso a colunas específicas.

```
CREATE VIEW salas_disponiveis AS
SELECT id, nome, capacidade
FROM salas
WHERE disponibilidade = true;
```

Esta view lista as salas que estão atualmente disponíveis.

### Vantagens e Necessidade de Usar Views

#### 1. Simplificação de Consultas Complexas:

Uma view pode simplificar o acesso a dados que requerem múltiplas junções, condições, ou transformações, consolidando tudo isso em uma única consulta simples. No exemplo do projeto, a view `salas_disponiveis` simplifica o acesso às salas que estão disponíveis, evitando a necessidade de repetir a condição `WHERE disponibilidade = true` em várias consultas diferentes.

#### 2. Segurança de Dados:

Views podem ser usadas para restringir o acesso a dados sensíveis. Por exemplo, se a tabela `salas` contiver colunas com informações sensíveis (como detalhes de segurança ou dados

pessoais dos proprietários), a view pode limitar a exposição dessas informações, mostrando apenas id, nome e capacidade.

### **3. Consistência dos Dados:**

Utilizar views garante que todas as aplicações e usuários que acessam os dados recebam informações consistentes e calculadas da mesma forma, pois a lógica está encapsulada na view e não em cada aplicação que faz a consulta.

### **4. Manutenção e Reutilização de Código:**

As views centralizam a lógica de negócios que seria duplicada em várias consultas SQL espalhadas por diversas aplicações. Isso facilita a manutenção e alterações, pois qualquer mudança na lógica de consulta precisa ser feita apenas uma vez na view, e não em múltiplas localidades.

### **5. Performance:**

Embora as views em si não armazenem dados (são recalculadas toda vez que são acessadas), o uso de views pode melhorar a performance ao reduzir a complexidade das consultas que os desenvolvedores ou sistemas precisam escrever. Além disso, algumas bases de dados suportam views materializadas, que são uma forma de armazenar os resultados da view como se fossem uma tabela física, melhorando significativamente a performance em consultas repetidas.

### **6. Abstração de Dados:**

As views permitem a abstração da estrutura da base de dados. Usuários e desenvolvedores podem interagir com a base de dados através de views sem conhecer os detalhes da estrutura subjacente das tabelas, o que é especialmente útil em sistemas complexos.

## **4. Trigger**

Esse código SQL é um ótimo exemplo para demonstrar como funções e triggers (gatilhos) podem ser usados para automatizar o log de mudanças em um banco de dados, uma prática comum em sistemas de gestão de bancos de dados para garantir a integridade e rastreabilidade dos dados. Vou explicar cada parte do código.

```
CREATE TABLE sala_log (
    id SERIAL PRIMARY KEY,
    sala_id INT,
    status BOOLEAN,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    acao VARCHAR(255)
);
```

```

CREATE OR REPLACE FUNCTION log_alteracao_sala()
RETURNS TRIGGER AS $$ 
BEGIN
    IF TG_OP = 'UPDATE' THEN
        INSERT INTO sala_log (sala_id, status, acao)
        VALUES (NEW.id, NEW.disponibilidade, CASE WHEN NEW.disponibilidade THEN 'Desalugada' ELSE 'Alugada' END);
        RETURN NEW;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_alteracao_sala
AFTER UPDATE OF disponibilidade ON salas
FOR EACH ROW
EXECUTE FUNCTION log_alteracao_sala();
|

```

## Parte 1: Criação da Função log\_alteracao\_sala()

### 1. CREATE OR REPLACE FUNCTION log\_alteracao\_sala():

- **CREATE OR REPLACE FUNCTION:** Este comando cria uma nova função ou substitui uma existente se ela já estiver definida no banco de dados. Isso facilita a manutenção do código, permitindo atualizações sem a necessidade de remoção manual prévia.
- **log\_alteracao\_sala():** Nome da função, que sugere que seu propósito é logar alterações em uma sala.

### 2. RETURNS TRIGGER:

- Indica que a função irá retornar um objeto do tipo TRIGGER. Isso significa que a função está destinada a ser usada como parte de um trigger, não sendo chamada diretamente, mas automaticamente em resposta a eventos no banco de dados.

### 3. Corpo da Função:

- **BEGIN ... END;:** Blocos que delimitam o início e o fim do procedimento da função.
- **IF TG\_OP = 'UPDATE' THEN:** Uma condição que verifica se a operação que disparou o trigger foi um UPDATE. TG\_OP é uma variável especial de triggers que contém o tipo de operação que ativou o trigger (INSERT, UPDATE, DELETE).
- **INSERT INTO sala\_log:** Comando que insere um novo registro na tabela sala\_log. Os campos sala\_id, status, acao são preenchidos respectivamente com:
  - **NEW.id:** NEW é uma pseudo-registro que contém os valores da linha após a atualização.
  - **NEW.disponibilidade:** Indica o estado atual da disponibilidade da sala.
  - **CASE WHEN NEW.disponibilidade THEN 'Desalugada' ELSE 'Alugada':** Uma expressão condicional que determina o texto da ação com base no novo estado de disponibilidade.
- **RETURN NULL;:** Em triggers que são disparados após um evento (AFTER triggers), geralmente retorna-se NULL, pois o resultado da função não altera a operação que ocorreu no banco de dados.

## Parte 2: Criação do Trigger trigger\_alteracao\_sala

### 1. CREATE TRIGGER trigger\_alteracao\_sala;

- **CREATE TRIGGER:** Comando para criar um trigger.
- **trigger\_alteracao\_sala:** Nome do trigger, intuitivamente nomeado para refletir sua função.

### 2. AFTER UPDATE OF disponibilidade ON salas;

- **AFTER UPDATE OF disponibilidade:** Especifica que o trigger deve ser disparado após uma atualização na coluna disponibilidade da tabela.
- **ON salas:** Especifica a tabela na qual o trigger será aplicado.

### 3. FOR EACH ROW;

Indica que o trigger deve ser executado para cada linha afetada pela operação de UPDATE.

### 4. EXECUTE FUNCTION log\_alteracao\_sala();

Especifica qual função deve ser executada quando o trigger é ativado.

## Informações Interessantes

- **Triggers** são extremamente úteis para manter a integridade dos dados, realizar auditorias e implementar regras de negócio automaticamente dentro do banco de dados.
- **Segurança e Rastreabilidade:** Triggers como o demonstrado ajudam a manter um histórico de alterações, o que é crucial para sistemas onde a segurança e a auditoria são importantes.
- **Automação:** Reduz a necessidade de intervenção manual para manter registros de log, pois o processo é automatizado.

## 5. Stored Procedure

Uma stored procedure é um conjunto de instruções SQL que você pode salvar, para que o código possa ser reutilizado muitas vezes. Pode aceitar parâmetros e é usada para automatizar processos mais complexos.

```

CREATE OR REPLACE PROCEDURE get_room_report()
LANGUAGE plpgsql
AS $$

BEGIN
    -- Verifica se a tabela temporária já existe e a deleta se for verdade
    DROP TABLE IF EXISTS temp_room_report;

    -- Cria a tabela temporária
    CREATE TEMP TABLE temp_room_report (
        id INT,
        nome VARCHAR(255),
        quantidade_alugueis INT
    );

    -- Insere os dados na tabela temporária
    INSERT INTO temp_room_report (id, nome, quantidade_alugueis)
    SELECT id, nome, quantidade_alugueis
    FROM salas
    ORDER BY id;

    -- A tabela temporária agora contém os dados e será usada para gerar relatórios
END;
$$;

```

**CREATE OR REPLACE PROCEDURE get\_room\_report()**

- **CREATE OR REPLACE PROCEDURE:** Este comando é usado para criar um novo procedimento armazenado ou substituir um existente. Utilizar "OR REPLACE" é prático porque permite atualizar o procedimento sem precisar excluí-lo e recriá-lo manualmente, facilitando a manutenção do código.
- **get\_room\_report():** Nome do procedimento, que sugere que ele gera um relatório sobre salas.

**LANGUAGE plpgsql**

- **LANGUAGE plpgsql:** Indica que a linguagem usada para definir o procedimento é PL/pgSQL, que permite a criação de procedimentos mais complexos e com maior controle de lógica do que o SQL padrão.

**AS ...;**

- **AS \$\$ e \$\$;:** Delimitam o corpo do procedimento. O uso de \$\$ é uma maneira de evitar conflitos com aspas simples que podem aparecer no código SQL.

## Corpo do Procedimento

**DROP TABLE IF EXISTS temp\_room\_report;**

- **DROP TABLE IF EXISTS:** Este comando verifica se uma tabela temporária chamada temp\_room\_report já existe. Se existir, ela é deletada. Isso é útil para garantir que o procedimento não falhe devido à presença de uma tabela antiga que já contém dados ou que não está mais atualizada.

```
CREATE TEMP TABLE temp_room_report (...);
```

- **CREATE TEMP TABLE:** Cria uma tabela temporária que existe apenas durante a sessão atual do banco de dados. Tabelas temporárias são úteis para armazenar dados intermediários que não precisam ser persistidos permanentemente.
- A tabela temp\_room\_report inclui as colunas:
  - **id INT:** Identificador da sala.
  - **nome VARCHAR(255):** Nome da sala.
  - **quantidade\_alugueis INT:** Quantidade de aluguéis registrados para a sala.

```
INSERT INTO temp_room_report (id, nome, quantidade_alugueis) SELECT ...
```

- **INSERT INTO temp\_room\_report:** Insere dados na tabela temporária.
- **SELECT id, nome, quantidade\_alugueis FROM salas ORDER BY id;:** Esta consulta seleciona os dados da tabela salas, provavelmente uma tabela permanente que armazena informações sobre as salas disponíveis. O ORDER BY id garante que os dados sejam inseridos na tabela temporária na ordem de seus identificadores.

## Utilidades e Importâncias

- **Isolamento de Alterações:** Utilizar uma tabela temporária dentro de um procedimento como este isola as alterações e manipulações dos dados sem afetar a tabela permanente diretamente.
- **Desempenho:** Dependendo da quantidade de dados e da complexidade das operações, trabalhar com tabelas temporárias pode ser mais rápido do que operar diretamente em tabelas grandes e permanentes, especialmente se as operações requerem reordenação frequente ou recálculo.
- **Segurança:** Manipular dados em uma tabela temporária reduz o risco de alterar ou danificar os dados originais durante o processamento.

Cada um desses elementos do banco de dados serve a propósitos distintos e são ferramentas poderosas para manipular, acessar e proteger os dados de forma eficiente e segura em sistemas de informação.

# Aprendizados que vou levar para os próximos projetos



Nunca tinha feito um projeto tão grande assim, foram muitos aprendizados, e eu construí principalmente uma boa noção de como preparar melhor os meus próximos projetos. Vou listar aqui:

1. **Versionar no Git:** Eu não tinha noção do tamanho que o projeto poderia chegar, e eu tenho costume de só postar no Github quando finalizo tudo, sem fazer versionamento. Mas confesso que nesse projeto, várias vezes quando eu implementei uma função nova, eu acabei fazendo funções antigas não funcionarem, e eu não lembrava mais como estava. Então, acho importante fazer um push sempre que uma nova função é concluída, é um tempo que vale a pena ser gasto.

No nono dia de projeto, quando tudo estava concluído, por muito pouco não perdi todo o projeto, eu não sabia do limite de 10.000 itens do Github e tentei enviar a aplicação React completa, que por base já tem 46.000 itens, e deu pau, perdi o código, tive que ajustar tudo de novo. Foi mais um aprendizado que nunca será esquecido, quase chorei.

2. **Esboço do design:** Confesso que, devido ao meu mal prefeito, eu sempre tinha que fazer mudanças, sempre que surgia uma nova ideia, percebia que ia ter que mudar tudo

para implementa-la, então, levo como aprendizado, definir bem os requisitos e fazer um esboço de baixa fidelidade mesmo, antes de começar a desenvolver, teria facilitado bastante as coisas, é um tempo que merece ser gasto também.

3. **Diagrama entidade-relacionamento:** Pois é amigo, eu sei que é bem chato fazer esse tipo de coisa, sempre vou muito afoito logo codar o projeto, mas assim como o esboço do design, é necessário também o esboço do banco de dados.
4. **Definir os requisitos:** Como já deu para perceber, basicamente o meu maior erro foi querer fazer a medida que as ideias vinham, percebi que isso faz as coisas ficarem extremamente trabalhosas e mal organizadas, então é preciso gastar um bom tempo antes, para preparar toda a idealização do projeto, e seguir fiel a ele.
5. **Docker:** Nunca fiz um projeto com docker, mas sei que ele funciona como um ambiente totalmente controlável, onde você pode baixar todas as suas dependências, acho que isso facilitaria a transição de uma máquina para outra, fica como tentativa para um próximo projeto (pretendo testar isso em breve).
6. **Guardar os comandos SQL:** Não sabia que os comandos no banco de dados sumiam, e acabei me esquecendo do que tinha feito.