

**ESCOLA DE ARTES CIÊNCIAS E HUMANIDADES DA UNIVERSIDADE DE
SÃO PAULO**

Amanda Gimenes Perellon NºUSP 12608730

Caio Vinícius Melo da Silva NºUSP: 12542859

Gabriel Dos Santos Nascimento NºUSP 12732792

Guilherme Faria do Nascimento NºUSP 12745282

EXERCICIO DE PROGRAMAÇÃO 3

São Paulo

2024

No presente relatório iremos descrever os detalhes da implementação da solução requisitada na 3 parte do trabalho de Banco de Dados ||.

Sumário

1. Apresentação:	3
2. Estrutura do banco de dados	4
Relacionamentos:	4
Normalização	4
3. Programas Desenvolvidos	6
• Aplicação FRONTEND:	6
• Aplicação Backend:	7
4. Arquitetura da solução.	9
5. Resolução dos problemas de desempenho	11
Processo de Tuning	11
Implementação do Índice B-tree	11
Resultados da Melhoria	11

1. Apresentação:

Pela impossibilidade de realizar upload do vídeo de apresentação do trabalho, fizemos o upload do vídeo no YouTube para que não houvesse perda de qualidade e seja de fácil acesso.

Link para acesso a apresentação: <https://youtu.be/liME1S-PFOw>

2. Estrutura do banco de dados

O banco de dados desenvolvido ao longo da disciplina compõe um conjunto de tabelas que satisfazem as necessidades de uma clínica médica. Para tanto, temos 7 tabelas principais:

- **Paciente:** Armazena informações sobre pacientes, como nome, CPF, endereço, idade, sexo e telefone.
- **Médico:** Armazena informações sobre médicos, como nome, CRM, telefone.
- **Agenda:** Armazena informações sobre agenda de seus respectivos médicos, como dia da semana, hora de início e fim, médico.
- **Especialidade:** Armazena informações sobre uma especialidade médica como nome da especialidade.
- **Consulta:** Armazena informações sobre consultas agendadas, como data, paciente a ser atendido, médico a atender, horário de início e fim.
- **Diagnostico:** Armazena informações sobre diagnósticos como tratamento recomendado e remédios receitados.
- **Doença:** Armazena informações sobre doenças, como nome da doença e código da doença.

Além dessas, o banco possui tabelas de relacionamento, sendo elas:

Exerceesp: Relaciona a tabela **médico** com a tabela **especialidade**.

Diagnostica: Relaciona a tabela **doença** com a tabela **diagnostico**.

Relacionamentos:

Um paciente pode ter várias consultas.

Uma consulta é realizada por um médico.

Um médico tem diversas consultas.

Um médico tem diversas especialidades

Um médico tem diversas agendas.

Um diagnostico tem diversas doenças.

Uma doença tem vários diagnósticos.

Normalização

O banco de dados está normalizado até o terceiro nível normal. Portanto, as tabelas não contêm redundâncias de dados e os relacionamentos entre as tabelas são bem definidos.

3. Programas Desenvolvidos

Para solução do problema proposto no exercício de programação construímos uma aplicação dividida em duas partes conforme diretrizes de sua arquitetura. Essas partes podem ser nomeadas como **FrontEnd** e **BackEnd**, cada uma tendo uma responsabilidade específica e complementar a outra.

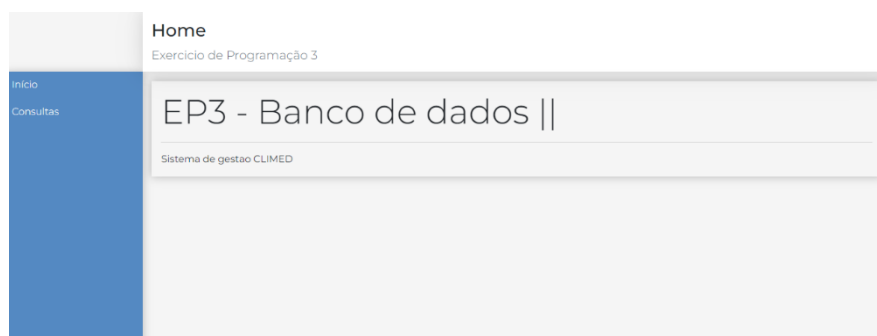
Para a solução do problema proposto no exercício de programação, desenvolvemos uma aplicação dividida em duas partes, conforme as diretrizes de sua arquitetura: Frontend e Backend, cada uma com responsabilidades específicas e complementares.

- Aplicação **FRONTEND**:

Execução: Para iniciar o sistema, basta digitar o comando `npm start` no terminal da aplicação.

Navegação: A aplicação possui um menu de navegação à esquerda para facilitar a migração entre as páginas. As principais telas são: Todos os sistemas são acompanhados de um menu de

- **Tela inicial:** Apresenta o sistema de forma simples e é exibida assim que a aplicação é iniciada.



- **Tela de agendamento:** Permite agendar consultas em duas etapas:
 1. Escolha da data e do médico.
 2. Preenchimento de dados adicionais como especialidade, paciente, horário, forma de pagamento, entre outros. Os horários são preenchidos dinamicamente conforme a disponibilidade do médico escolhido. Por exemplo, se o médico tem consultas das 08:00 às 12:00, somente horários posteriores estarão disponíveis. Para finalizar o agendamento, basta clicar no botão **“Agendar Consulta”**.

- **Tela de gestão de consultas:** Permite a edição dos dados de uma consulta. O usuário deve inserir o ID da consulta no campo de busca para exibir os dados, que podem ser alterados conforme necessário. Após as alterações, é preciso clicar no botão “**Editar Dados da Consulta**” para confirmar.

- **Aplicação Backend:**

Construída com a tecnologia assíncrona baseada em eventos Node.js, permite a execução de JavaScript fora de um navegador web.

Execução: Para iniciar o sistema, basta digitar o comando `npm start` no terminal da aplicação.

Funcionalidades: A aplicação gerencia a conexão com o banco de dados e executa comandos SQL. Além disso, cria um servidor local no endereço “**http://localhost:4040**”, onde expõe endpoints responsáveis pelas requisições dos métodos da aplicação. As rotas do sistema são:

```

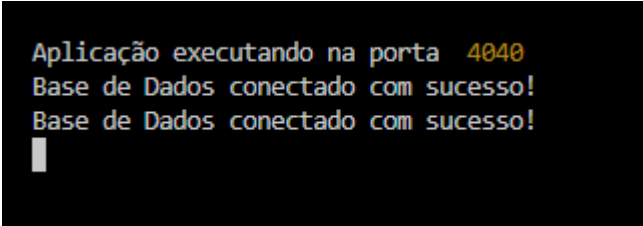
4
5 // ROTAS CLIMED
6 router.get('/medicos', controller.showMedicos);
7 router.post('/agendaMedico', controller.showAgendaPorMedicos);
8 router.post('/agendaCodigo', controller.showAgendaPorCodigo);
9 router.post('/agendarConsulta', controller.agendar);
10 router.post('/atualizarDados', controller.atualizarAgenda);
11

```

- **Médicos:** retorna uma lista com todos os médicos disponíveis para agendamento de consultas.

- **agendaMedico:** retorna a agenda de um médico para validar horários disponíveis.
- **agendaCodigo:** retorna uma consulta a partir de seu id.
- **AgendarConsulta:** Realiza a inclusão de uma consulta.
- **Atualizardados:** realiza a atualização dos dados de uma consulta.

A aplicação registra logs no terminal de execução para monitorar os acessos feitos pela aplicação frontend.

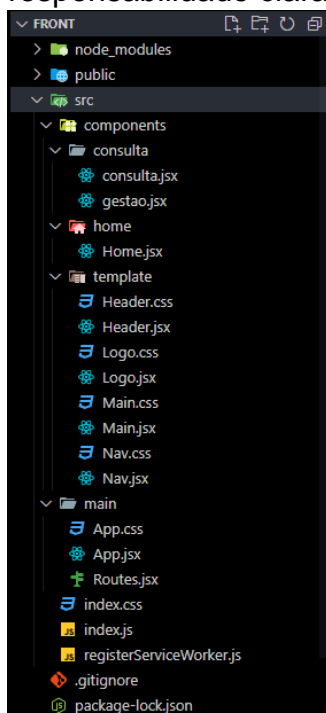
A screenshot of a terminal window with a black background and white text. The text displays three lines of logs: 'Aplicação executando na porta 4040', 'Base de Dados conectado com sucesso!', and 'Base de Dados conectado com sucesso!'. A white cursor is visible on the line following the second success message.

```
Aplicação executando na porta 4040  
Base de Dados conectado com sucesso!  
Base de Dados conectado com sucesso!  
█
```


4. Arquitetura da solução.

As aplicações seguem os princípios da arquitetura **SOLID**, que visam criar sistemas com classes bem definidas e com responsabilidades únicas, facilitando a manutenção e a escalabilidade do código.

A aplicação **frontend** mantém uma estrutura organizada seguindo os padrões da arquitetura **SOLID**, realizando a divisão de componentes atômicos que possuem repetição, páginas, estilização, rotas e infraestrutura do próprio sistema. A organização permite que cada componente tenha uma responsabilidade clara, conforme ilustrado na imagem abaixo:



Definindo a composição:

Componentes Atômicos: Pequenos componentes reutilizáveis que podem ser combinados para formar interfaces mais complexas.

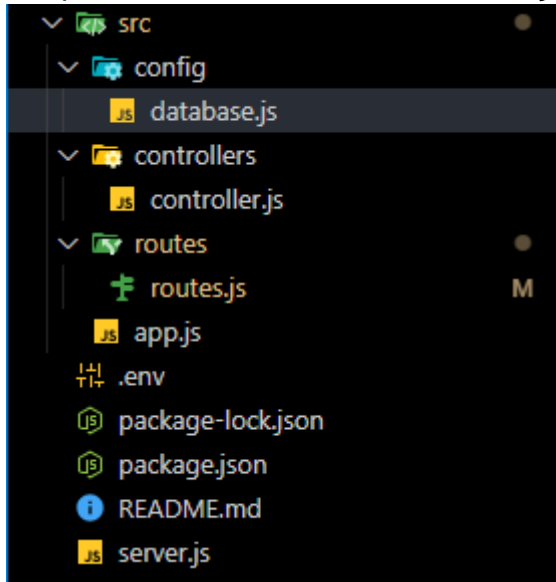
Páginas: Conjuntos de componentes que formam as telas do aplicativo.

Estilização: Gerenciamento de estilos para garantir uma interface consistente e responsiva.

Rotas: Definição das rotas para navegação entre as diferentes páginas da aplicação.

Infraestrutura: Configurações e serviços que suportam o funcionamento da aplicação, como a gestão do estado e a comunicação com o backend.

A aplicação **backend** também segue os princípios **SOLID**, com classes bem divididas e responsabilidades únicas. Por exemplo, a classe “Routes.js” é responsável apenas por declarar os endpoints, enquanto a classe “Controller” contém as funcionalidades específicas para cada endpoint. A divisão clara de responsabilidades facilita a manutenção e a escalabilidade do sistema.



Destarte, a arquitetura foi escolhida pois a

organização clara e a aplicação dos princípios SOLID garantem que a solução seja robusta, flexível e fácil de manter.

5. Resolução dos problemas de desempenho

Identificamos uma oportunidade de melhorar a eficiência na função de agendamento de consultas. O problema consistia em que, para escolher um horário disponível para um médico, o sistema realizava uma consulta para obter todos os horários ocupados e, em seguida, os excluía deixando somente horários vagos disponíveis para seleção. Este processo era ineficiente, considerando que todos os médicos possuem muitas consultas agendadas.

Processo de Tuning

Conforme a documentação de tuning para PostgreSQL disponível no edisciplinas, para notar uma melhoria na eficiência de uma consulta, é necessário um mínimo de 100 mil registros, dependendo do hardware. Com isso em mente, inserimos 750 mil registros na tabela de CONSULTA via código que será entregue junto a este relatório, respeitando a consistência dos dados do banco.

Implementação do Índice B-tree

Para melhorar a eficiência, utilizamos a função B-tree, que é recomendada para a maioria das aplicações conforme a documentação. Criamos um índice no campo idmedico usando o seguinte comando:

```
CREATE INDEX cl_medico ON consulta USING btree(idmedico);
```

Resultados da Melhoria

Durante a apresentação deste trabalho, realizamos a criação do índice em tempo real e executamos uma consulta para mostrar a quantidade de registros cadastrados, evidenciando o sucesso da melhoria de eficiência.

Comparando os logs da consulta exibidos pelo comando EXPLAIN antes e depois da criação do índice, verificamos os seguintes tempos de execução:

- **Antes da criação do índice:** 17 milissegundos
- **Depois da criação do índice:** 12 milissegundos

Isso demonstra uma melhoria de 5 milissegundos na eficiência da consulta. Embora a consulta tenha sido realizada em uma base de dados local, consideramos a melhoria significativa, sabendo que em um ambiente de produção essa otimização poderia resultar em ganhos de desempenho ainda maiores.