\*\*README.md\*\*

---

## ToDo App Full-Stack

### Descrição Geral

Este projeto implementa um sistema de gerenciamento de tarefas (ToDo List) com interface web em React e backend em Node.js/Express integrado a um banco de dados MySQL.

A interface apresenta uma lista de cards coloridos no estilo Kanban, filtragem por texto e status, criação/edição via modal e um dashboard com gráfico de pizza mostrando a distribuição de tarefas por status.

---

## Funcionalidades

1. \*\*CRUD de Tarefas\*\*

\* \*\*Listar\*\*: Exibe todas as tarefas ordenadas por data de criação.

\* \*\*Criar\*\*: Modal para cadastrar título, descrição e status.

\* \*\*Editar\*\*: Modal pré-preenchido para alterar título, descrição ou status.

\* \*\*Excluir\*\*: Confirmação “Tem certeza?” antes de remover.

2. \*\*Filtro de Tarefas\*\*

\* Campo de texto para busca em título/descrição.

\* Select para filtrar por status (Pendente, Concluída, Em Progresso, Cancelada, Aguardando).

3. \*\*Interface Kanban-Light\*\*

\* Cada tarefa é exibida num card colorido conforme seu status (cores pastéis).

\* Cards mostram título, descrição, status e data/hora de criação.

4. \*\*Dashboard de Status\*\*

\* Gráfico de pizza (Recharts) ao lado da lista exibe, em tempo real, a contagem de tarefas por status.

---

## Tecnologias Utilizadas

\* \*\*Frontend\*\*

\* React (Create-React-App)

\* Recharts

\* Axios

\* CSS puro (estrutura de classes e variáveis CSS para cores)

\* \*\*Backend\*\*

\* Node.js + Express

\* MySQL (mysql2/promise)

\* dotenv (variáveis de ambiente)

\* \*\*Banco de Dados\*\*

\* MySQL 5.7 ou superior

\* Tabela `tarefas` com colunas:

```sql

CREATE TABLE IF NOT EXISTS tarefas (

id INT NOT NULL AUTO\_INCREMENT,

titulo VARCHAR(255) NOT NULL,

descricao TEXT NULL,

status VARCHAR(100) NOT NULL,

datacriacao DATETIME NOT NULL,

PRIMARY KEY (id)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

---

## Pré-requisitos

\* \*\*Node.js\*\* (v14+ recomendado)

\* \*\*npm\*\* (v6+ ou v8+)

\* \*\*MySQL\*\* (rodando localmente ou em container; credenciais configuradas no `.env`)

\* (Opcional) \*\*npx\*\* para criar o frontend

---

## Estrutura de Pastas

```

/

├── backend/ ← API Node.js

│ ├── db.js ← Conexão MySQL via mysql2/promise

│ ├── index.js ← Servidor Express, rotas CRUD

│ └── .env ← Variáveis de ambiente (não versionar)

├── todo-front/ ← Aplicação React

│ ├── public/

│ ├── src/

│ │ ├── components/

│ │ │ ├── TaskItem.js

│ │ │ ├── TaskList.js

│ │ │ ├── TaskModal.js

│ │ │ └── StatusChart.js

│ │ ├── App.js

│ │ ├── App.css

│ │ └── index.js

│ ├── package.json

│ └── package-lock.json

└── README.md

```

## Configuração do Backend (Node.js)

1. \*\*Acesse a pasta\*\*

```bash

cd backend

```

2. \*\*Crie o arquivo `.env`\*\*

No diretório `backend`, crie um `.env` com estas linhas (ajuste conforme seu MySQL):

```

DB\_HOST=127.0.0.1

DB\_USER=root

DB\_PASS=Sc00b12$

DB\_NAME=todoapp

DB\_PORT=3306

```

3. \*\*Instale dependências\*\*

```bash

npm install

```

\* `express`

\* `mysql2`

\* `dotenv`

\* `cors` (para liberar requisições Cross-Origin)

\* `nodemon` (dependência de desenvolvimento)

4. \*\*Scripts disponíveis\*\* (em `backend/package.json`):

```json

"scripts": {

"start": "node index.js",

"dev": "nodemon index.js"

}

```

\* `npm run dev`: inicia o servidor em modo desenvolvimento.

5. \*\*Inicialize o banco de dados\*\*

\* Crie o banco `todoapp` e a tabela `tarefas` (veja o script na seção “Banco de Dados” acima).

\* Exemplo via MariaDB CLI:

```sql

CREATE DATABASE IF NOT EXISTS todoapp;

USE todoapp;

CREATE TABLE IF NOT EXISTS tarefas (...);

```

6. \*\*Execute o servidor\*\*

```bash

npm run dev

```

\* Se tudo estiver correto, verá no console:

```

→ DB\_USER: root

…

Server rodando na porta 3001

```

7. \*\*Endpoints REST\*\*

\* \*\*GET `/ping`\*\* → retorna `pong` (health check)

\* \*\*GET `/tarefas`\*\* → lista todas as tarefas (ordenadas por data de criação)

\* \*\*POST `/tarefas`\*\* → cria tarefa (body: `{ titulo, descricao, status }`)

\* \*\*PUT `/tarefas/:id`\*\* → edita tarefa (body: `{ titulo, descricao, status }`)

\* \*\*DELETE `/tarefas/:id`\*\* → remove tarefa

---

## Configuração do Frontend (React)

1. \*\*Acesse a pasta\*\*

```bash

cd todo-front

```

2. \*\*Instale dependências\*\*

```bash

npm install

```

\* `axios`

\* `recharts`

3. \*\*Ajuste o proxy\*\*

Em `package.json`, confirme que existe:

```json

"proxy": "http://localhost:3001"

```

Isso faz com que toda requisição a `/tarefas` seja redirecionada ao backend na porta 3001.

4. \*\*Estrutura de Componentes\*\*

\* \*\*`TaskItem.js`\*\*: card individual de tarefa, cor de fundo baseada no status, botões “Editar” e “Excluir (confirmação)”.

\* \*\*`TaskList.js`\*\*: renderiza lista de `TaskItem` a partir de um array de tarefas.

\* \*\*`TaskModal.js`\*\*: modal para criar ou editar tarefa (pré-preenchido em edição).

\* \*\*`StatusChart.js`\*\*: gráfico de pizza (PieChart via Recharts) que contabiliza tarefas por status.

5. \*\*Inicie o React\*\*

```bash

npm start

```

\* O app abrirá em `http://localhost:3000`.

\* A interface consome automaticamente o backend via proxy.

---

## Uso

1. \*\*Visualizar Tarefas\*\*

\* Ao acessar `http://localhost:3000`, a lista de tarefas é carregada.

\* Cada card mostra título, descrição, status e data de criação.

2. \*\*Filtrar Tarefas\*\*

\* Use o campo “Buscar por texto” para procurar em título ou descrição.

\* Use o select de status para ver apenas tarefas de um status específico.

\* A lista é atualizada em tempo real conforme você digita ou seleciona um status.

3. \*\*Criar Nova Tarefa\*\*

\* Clique em `+ Nova Tarefa`.

\* Preencha “Título”, “Descrição” e escolha “Status”.

\* Clique em \*\*Salvar\*\*; a tarefa aparecerá imediatamente na lista.

4. \*\*Editar Tarefa\*\*

\* Clique em \*\*Editar\*\* no card desejado.

\* O modal abrirá com os dados atuais.

\* Faça alterações e clique em \*\*Salvar Alterações\*\*.

5. \*\*Excluir Tarefa\*\*

\* Clique em \*\*Excluir\*\* no card desejado.

\* Será exibido um `window.confirm("Tem certeza que deseja excluir esta tarefa?")`.

\* Se confirmar, a tarefa será removida do banco e da lista.

6. \*\*Dashboard de Status\*\*

\* Ao lado da lista, há um gráfico de pizza mostrando o número de tarefas em cada status.

\* Ele se atualiza automaticamente toda vez que você criar, editar ou excluir uma tarefa.

---

## Padrões e Boas Práticas

\* \*\*Componentização\*\*

\* Cada responsabilidade em arquivo separado (cards, lista, modal, gráfico).

\* Facilita manutenção e testes futuros.

\* \*\*Estilização via CSS\*\*

\* Uso de classes semânticas (`.task-card`, `.status-<nome>`, `.task-container`, etc.).

\* Variáveis CSS (`:root`) para cores principais e textuais.

\* \*\*Rotas REST Simples\*\*

\* Seguir convenções HTTP (GET, POST, PUT, DELETE).

\* Respostas com status code adequados:

\* 200 (OK) para GET e PUT,

\* 201 (Created) para POST,

\* 204 (No Content) para DELETE,

\* 400 (Bad Request) para dados inválidos,

\* 404 (Not Found) se `id` inexistente,

\* 500 (Internal Server Error) para falhas não tratadas.

\* \*\*Tratamento de Erros\*\*

\* Backend: logger (`console.error`) e JSON padronizado `{ erro: "mensagem" }`.

\* Frontend: `console.error` em caso de falha nas requisições.

---

## Considerações Finais e Próximos Passos

\* \*\*Testes Automatizados\*\*

\* Criar testes unitários no backend (JUnit) e testes de componente no frontend (React Testing Library).

\* \*\*Dockerização\*\*

\* \*\*Dockerfile (backend)\*\*: empacotar o serviço Node ou, na migração, o Spring Boot.

\* \*\*Dockerfile (frontend)\*\*: build estático da React App.

\* \*\*docker-compose.yml\*\*: orquestrar banco MySQL, backend e frontend para facilitar setup.

\* \*\*CI/CD\*\*

\* Configurar pipeline (GitHub Actions, GitLab CI, etc.) para: testes, build e deploy automático.

---

#### Autor

\* Desenvolvido por Reinaldo “Gepeto” Fernandes, com foco em clean code, usabilidade e arquitetura modular.

---

\*\*Obrigado por conferir!\*\*

Nenhuma dúvida, desafio ou sugestão? Abra uma issue ou entre em contato diretamente.