PLANO DE EXECUÇÃO
PROJETO BACK-END FRAMEWORKS

Sistema de Gestão de Eventos e Inscrições

MEMBROS:

MARCOS EMANUEL CELESTINO TAVARES – 01718690
PEDRO ANTÔNIO DA SILVA IZÍDIO – 01705873
ALFRED MANOEL VICENTE SILVA DA ROCHA – 01695802
VINICIUS REGIS DA SILVA – 01738125
RICHARD BRYAN LYRA DA SILVA – 01763796

Visão Geral do Projeto

O **Sistema de Gestão de Eventos e Inscrições** tem como objetivo desenvolver uma **aplicação back-end funcional em Node.js com Express**, conectada a um **banco de dados relacional PostgreSQL**, capaz de gerenciar **usuários, eventos, palestras, inscrições e pagamentos**.

O projeto será testado via **Postman ou Thunder Client**, sem interface gráfica, e servirá como um exercício prático dos conceitos de **arquitetura MVC**, **autenticação com JWT**, **criptografia de senha** e **operações CRUD** completas.

Estrutura do Projeto

A estrutura de pastas já definida segue o modelo **MVC simplificado**, com base na divisão entre configuração, rotas, controladores e lógica principal.

```
src/
├── config/
│   └── db.js              # Configuração e conexão com banco de dados
(Supabase / PostgreSQL)
├── controllers/
│   └── usuarioController.js  # Regras de negócio dos usuários (login,
registro etc.)
├── routes/
│   └── usuarioRoutes.js      # Definição das rotas de usuários
(Express)
└── server.js                 # Arquivo principal: inicializa servidor
e registra rotas
```

Cada integrante será responsável por partes específicas da aplicação, respeitando as dependências entre módulos.

Etapas Gerais do Desenvolvimento

1. **Configuração do ambiente e banco de dados**
2. **Criação das entidades e relacionamentos SQL**
3. **Desenvolvimento dos modelos e controladores**
4. **Criação das rotas (endpoints Express)**
5. **Implementação da autenticação JWT e criptografia de senhas**
6. **Testes de requisições e ajustes finais**
7. **Documentação e entrega**

Descrição e Divisão das Funções

A seguir, estão detalhadas as responsabilidades individuais de cada membro do grupo, incluindo as dependências de trabalho e orientações específicas.

MARCOS (CT) – Líder Técnico / Configuração e Integração

Responsabilidades Principais:

1. **Configurar o banco de dados (PostgreSQL via Supabase):**
   - Criar o banco e tabelas principais com as entidades definidas no DER.
   - Garantir relacionamentos (chaves primárias e estrangeiras).
   - Escrever o **script SQL** de criação das tabelas.
2. **Arquivo `/src/config/db.js`:**
   - Configurar a conexão com o banco via `pg` (ou `mysql2`, se for usado MySQL).
   - Criar e exportar uma função de consulta genérica (`query`) para uso nos controladores.
   - Tratar erros e reconexões automáticas ao banco.
3. **Arquivo `/src/server.js`:**
   - Configurar o servidor Express.
   - Registrar rotas principais com `app.use()`.
   - Definir middlewares básicos (ex: `express.json()` e `cors()`).
   - Criar mensagem de inicialização no console ("Servidor rodando na porta 3000").
4. **Integração final entre módulos:**
   - Garantir que o servidor reconhece e carrega todas as rotas criadas.
   - Auxiliar os colegas com importações e testes de endpoints.
   - Validar se as respostas JSON estão no formato correto.

Dependência:
A finalização do `db.js` e do `server.js` é **prioritária** — os demais membros devem aguardar Marcos concluir a conexão com o banco **antes de testar controladores e rotas**.

IZIDIO – Desenvolvedor de Rotas (Routes Layer)

Responsabilidades Principais:

1. **Criar e organizar os arquivos de rotas:**
   - Basear-se na estrutura já existente em `/routes/usuarioRoutes.js`.
   - Criar novas rotas para cada entidade:

        - `/eventosRoutes.js`
        - `/palestrasRoutes.js`
        - `/inscricoesRoutes.js`
        - `/pagamentosRoutes.js`
        - `/categoriasRoutes.js`
        - `/locaisRoutes.js`
    - Utilizar `Router()` do Express, definindo métodos:
        - `GET` (listar todos e por ID)
        - `POST` (criar)
        - `PUT` (atualizar)
        - `DELETE` (excluir)
2. **Conectar rotas aos controladores correspondentes:**
    - Importar funções do respectivo controller (por exemplo, `usuarioController.js`).
    - Exemplo:
    - `import { listarUsuarios, criarUsuario } from '../controllers/usuarioController.js';`
    - `router.get('/', listarUsuarios);`
    - `router.post('/', criarUsuario);`
3. **Registrar rotas no servidor (após Marcos concluir `server.js`):**
    - Adicionar rotas no `server.js` com `app.use('/usuarios', usuarioRoutes).`

Dependência:
Aguardar **Marcos** finalizar a conexão com o banco e a configuração do servidor.
Após isso, trabalhar em conjunto com **Richards**, pois cada rota depende de uma função controller pronta.

RICHARDS (RICK) – Controladores e Lógica de Negócio

Responsabilidades Principais:

1. **Criar e organizar os controladores de cada entidade:**
    - Criar arquivos:
        - `/controllers/eventoController.js`
        - `/controllers/palestraController.js`
        - `/controllers/inscricaoController.js`
        - `/controllers/pagamentoController.js`
        - `/controllers/categoriaController.js`
        - `/controllers/localController.js`
2. **Funções CRUD:**
   Cada controller deve exportar as funções padrão:

```
export const listar = async (req, res) => { ... }
export const buscarPorId = async (req, res) => { ... }
export const criar = async (req, res) => { ... }
export const atualizar = async (req, res) => { ... }
export const excluir = async (req, res) => { ... }
```

8. **Integração com o banco de dados:**
    - Utilizar a função `db.query()` exportada de `db.js` (feita por Marcos).
    - Exemplo:
    - `const result = await db.query('SELECT * FROM eventos');`

- `res.json(result.rows);`

9. **Tratar erros e validações simples (campos obrigatórios).**

Dependência:
Deve aguardar **Marcos** concluir a conexão no `db.js` para realizar testes.
Trabalhar em conjunto com **Izidio**, pois cada controller será chamado dentro de uma rota.

PAULISTA – Middleware e Autenticação (JWT + Criptografia)

Responsabilidades Principais:

1. **Criar o diretório `/src/middleware/`:**
   - `authMiddleware.js` → valida o token JWT.
   - `validateFields.js` → middleware genérico de validação (opcional).
2. **Implementar autenticação:**
   - **Login e Registro** no `usuarioController.js`.
   - Usar **bcryptjs** para criptografar senhas.
   - Usar **jsonwebtoken** para gerar e verificar tokens.
   - Estrutura esperada:
   - `const token = jwt.sign({ id_usuario }, process.env.JWT_SECRET, { expiresIn: '1h' });`
3. **Proteger rotas sensíveis:**
   - Exemplo:
   - `router.post('/eventos', authMiddleware, criarEvento);`
4. **Manter variáveis de ambiente seguras** no arquivo `.env`:
   - `JWT_SECRET`, `DB_USER`, `DB_PASSWORD`, `DB_HOST` etc.

Dependência:
Deve aguardar **Richards** implementar o `usuarioController.js` e **Izidio** criar as rotas de usuário antes de testar a autenticação.
Após isso, poderá aplicar o middleware nas demais rotas.

ALFRED (INDIANO) – Banco de Dados, Script SQL e Documentação Final

Responsabilidades Principais:

1. **Modelagem e script SQL:**
   - Criar um arquivo `/src/database.sql` contendo todas as tabelas e relacionamentos.
   - Garantir a presença das 10 entidades principais.
   - Incluir comandos `CREATE TABLE`, `PRIMARY KEY`, `FOREIGN KEY` e `REFERENCES`.
2. **Testar a estrutura do banco com Supabase / pgAdmin.**
3. **Documentação e entrega:**

- Criar o documento `endpoints.pdf` ou `collection Postman`.
- Organizar os endpoints testados (método, rota e exemplo de corpo JSON).
- Auxiliar Marcos na criação do **README.md final** com instruções de execução.

Dependência:
O script SQL precisa ser feito **antes dos testes do Richards e Izidio**, pois os controladores e rotas dependem das tabelas.
A documentação deve ser feita **após todos os endpoints estarem funcionando**.

Funções de Cada Membro (Resumo Final)

Integrante	Função	Responsabilidades
Marcos (CT)	Líder técnico	Banco de dados (`db.js`), servidor (`server.js`), integração geral e suporte técnico
Izidio	Rotas (Express)	Criação e organização das rotas de cada entidade, conexão com controladores
Richards (Rick)	Controladores	Implementação das funções CRUD e integração com o banco
Paulista	Autenticação e Middleware	Criação de middlewares, login/registro com JWT e bcrypt
Alfred (Indiano)	Banco e Documentação	Criação do script SQL, testes de tabelas e documentação final

---

Fluxo de Trabalho Recomendado (Ordem de Execução)

1. **Marcos (CT)** – Finalizar `db.js` e `server.js`.
2. **Alfred (Indiano)** – Criar `database.sql` e testar no Supabase.
3. **Richards (Rick)** – Implementar controladores com base no banco.
4. **Izidio** – Criar e conectar as rotas aos controladores.
5. **Paulista** – Implementar autenticação e aplicar middleware JWT.
6. **Todos** – Testar no Postman e documentar resultados.