Visão geral da arquitetura

O backend está organizado em camadas:

1. **server.js**
    - Ponto de entrada da aplicação Express.
    - Configura CORS, JSON, autenticação e registra todas as rotas.
2. **/config**
    - Conexão com o Supabase (supabase.js).
    - Configuração do Mercado Pago (mercadoPago.js).
3. **/middlewares**
    - auth.js: cuida da autenticação via **JWT** e controle de acesso por **role**.
4. **/controllers**
    - Cada arquivo representa um "módulo" de negócio (eventos, equipes, pessoas, etc.).
    - Recebem req e res, conversam com o Supabase e devolvem JSON.
5. **/routes**
    - Definem as URLs (ex: /eventos) e qual função do controller atende cada rota.

🚀 server.js

Arquivo: server.js
Função: ponto de entrada do backend.

O que ele faz:

1. **Carrega variáveis de ambiente**

   ```
   import dotenv from "dotenv";
   dotenv.config();
   ```

2. **Cria a aplicação Express**

   ```
   const app = express();
   app.use(cors());
   app.use(express.json());
   ```

3. **Importa as rotas**
   - authRoutes → /auth
   - adminRoutes → /admin
   - pessoasRoutes → /pessoas
   - eventosRoutes → /eventos
   - equipeRoutes → /equipes
   - equipesEventoRoutes → /equipes-evento
   - teamroleRoutes → /teamrole
   - momentosRoutes → /momentos
   - inscricoesRoutes → /inscricoes
   - coordenadoresRoutes → /coordenadores
   - encontristaInscricaoRoutes → /encontrista-inscricao
   - pagamentoRoutes → /pagamento
   - webhookRoutes → /webhook
   - devRoutes → /dev
4. **Define rotas públicas**
   - /auth (login / register)
   - /webhook (Mercado Pago manda notificações)
   - /dev (utilidades de desenvolvimento, se usadas)
5. **Define rotas protegidas com authMiddleware**
   - Todas as outras: /pessoas, /eventos, /equipes, /teamrole, etc.
   - Isso significa que você **precisa de JWT** nessas rotas.
6. **Rota raiz**

   ```
   app.get("/", (req, res) => {
     res.send("🚀 Backend ECC funcionando!");
   });
   ```

7. **Sobe o servidor**

   ```
   app.listen(PORT, () => console.log(`Servidor rodando...`));
   ```

⚙ Pasta /config

config/supabase.js

Função: cria um client do Supabase para o projeto inteiro.

- Lê SUPABASE_URL e SUPABASE_ANON_KEY do .env
- Cria:

  ```
  const supabase = createClient(supabaseUrl, supabaseKey);
  export default supabase;
  ```

- Todos os controllers fazem consultas assim:

  ```
  const { data, error } = await supabase.from("pessoas").select("*");
  ```

---

config/mercadoPago.js ***(nome que usamos na prática)***

Função: centralizar a configuração do Mercado Pago.

- Seta o access_token do Mercado Pago com a env MP_ACCESS_TOKEN.
- Costuma exportar algo como:

  ```
  export const mpPayment = mercadopago.payment;
  ```

- Os controllers de pagamento chamam:

  ```
  const payment = await mpPayment.create({ body });
  ```

📦 **Pasta /middlewares**

middlewares/auth.js

Função: autenticação e autorização via JWT.

Provavelmente exporta:

1. **authMiddleware**
    - Pega o header Authorization: Bearer <token>.
    - Valida o JWT com JWT_SECRET.
    - Se ok, coloca o usuário em req.user:

      req.user = { id, email, role }

    - Se falhar → 401 Unauthorized.
2. **requireRole(role)**
    - Middleware que só deixa passar se req.user.role combinar com o role exigido.
    - Ex: requireRole(Roles.ADMIN).
3. **Roles**
    - Enumzinho com as roles:

      export const Roles = { ADMIN: "admin", USER: "user" };

Resumo:
Toda vez que você vê authMiddleware ou requireRole(Roles.ADMIN) nas rotas, é esse arquivo que está cuidando de conferir o token e o tipo de usuário.

🧠 Controllers (regra de negócio)

Vou agrupar por domínio.

🔐 controllers/auth.controller.js

Função: login e registro de usuários.

Principais funções:

1. **register(req, res)**
   - Valida nome, email, telefone, password com Zod.
   - Verifica se email já existe em pessoas.
   - Cria password_hash com bcrypt.hash.
   - Insere na tabela pessoas com role: "user".
   - Gera um JWT com id, email, role.
   - Retorna:
     { "pessoa": { ... }, "token": "xxx" }
2. **login(req, res)**
   - Valida email e password.
   - Busca a pessoa pelo email (incluindo password_hash).
   - Compara senha com bcrypt.compare.
   - Se ok → gera JWT.
   - Retorna pessoa (sem hash) + token.
   - Se não ok → 401 { error: "Credenciais inválidas" }.

🛠️ controllers/admin.controller.js

Função: permitir que um admin crie usuários direto pelo painel.

- **createUserByAdmin(req, res)**
  - Valida presença de email e password.
  - Confere se email já existe.
  - Gera password_hash.
  - Insere pessoa em pessoas com role (padrão "user" ou "admin" se vier no body).
  - Retorna 201 com:
    { "message": "Usuário criado com sucesso", "pessoa": { ... } }

👤 controllers/pessoas.controller.js

Função: CRUD de pessoas (dados básicos).

1. **listarPessoas**
   - GET /pessoas
   - Retorna lista com id, nome, email, telefone, role.

2. **buscarPessoa**
    - GET /pessoas/:id
    - Busca pessoa pelo id.
    - 404 se não existir.
3. **criarPessoa**
    - POST /pessoas
    - Valida com Zod (nome, email, etc.).
    - Verifica se o email já existe.
    - Insere na tabela.
    - Retorna message + data.
4. **atualizarPessoa**
    - PUT /pessoas/:id
    - Valida campos com schema parcial.
    - Se trocar email, verifica duplicidade.
    - Atualiza a pessoa.
5. **atualizarSenha**
    - PATCH /pessoas/:id/senha
    - Valida nova senha.
    - Gera password_hash com bcrypt.
    - Atualiza apenas a senha.
6. **deletarPessoa**
    - DELETE /pessoas/:id
    - Deleta a pessoa (somente admin pela rota).

---

controllers/eventos.controller.js

Função: gerir eventos ECC.

1. **listarEventos**
    - Retorna todos os eventos ordenados por start_date.
2. **buscarEvento**
    - Busca evento por ID.
3. **criarEvento**
    - Valida com Zod: nome, local, start_date, etc.
    - Garante que nome é único.
    - Insere na tabela eventos.
4. **atualizarEvento**
    - Atualização parcial do evento.
5. **deletarEvento**
    - Remove evento (cascateia nas FKs por causa dos ON DELETE CASCADE).

👥 controllers/equipe.controller.js

Função: gerir as equipes do encontro.

1. **listarEquipes** – todas as equipes.
2. **buscarEquipe** – equipe por ID.
3. **criarEquipe**
    - Valida nome.
    - Garante que nome é único.
    - Insere em equipes.
4. **atualizarEquipe** – atualiza nome/descrição.
5. **deletarEquipe** – remove equipe (se não tiver FKs ativas).

🔗 controllers/equipesEvento.controller.js

Função: vincular equipes a eventos.

1. **listarVinculos**
    - Lista tudo de equipes_evento com equipe + evento populados (JOIN via Supabase).
2. **listarPorEvento**
    - Só vínculos de um evento.
3. **listarPorEquipe**
    - Só vínculos de uma equipe.
4. **criarVinculo**
    - Valida equipe_id e evento_id.
    - Verifica se já existe vínculo igual.
    - Insere.
5. **deletarVinculo**
    - Remove o registro da tabela equipes_evento.

🖇️ controllers/teamrole.controller.js

Função: vínculo pessoa ⇄ equipe ⇄ evento (quem está em qual equipe, e se é líder).

1. **listarTeamRoles**
    - Lista geral com:
        - pessoa (id, nome, email)
        - equipe (id, nome)
        - evento (id, nome)
2. **listarPorPessoa**
    - Equipes e eventos de uma pessoa.
3. **listarPorEquipe**
    - Pessoas e eventos de uma equipe.
4. **listarPorEvento**
    - Pessoas e equipes de um evento.

5. **criarTeamRole**
    - Valida pessoa_id, equipe_id, evento_id.
    - Evita duplicidade: mesma pessoa + equipe + evento.
    - Insere.
6. **atualizarTeamRole**
    - Atualiza, por exemplo, is_leader ou pagou.
7. **deletarTeamRole**
    - Remove o vínculo por ID.

📜 **controllers/inscricoes.controller.js**

Função: gerenciar inscrições de pessoas em eventos.

1. **listarInscricoes**
    - Lista global com pessoa + evento.
2. **listarPorEvento**
    - Todas as inscrições de um evento.
3. **listarPorPessoa**
    - Todas as inscrições de uma pessoa.
4. **buscarInscricao**
    - Inscrição por ID.
5. **criarInscricao**
    - Valida com Zod (pessoa_id, evento_id, tipo, valor).
    - Garante que a pessoa **não tem outra inscrição** nesse mesmo evento.
    - Cria com status = 'pending'.
6. **atualizarInscricao**
    - Pode mudar status, tipo, valor, paid_by_pessoa_id.
7. **cancelarInscricao**
    - Atualiza status para cancelled.
8. **deletarInscricao**
    - Remove inscrição (somente admin).

📋 **controllers/encontristaInscricao.controller.js**

Função: formulário grande do casal encontrista (não é a inscrição "técnica", é a ficha de informações).

1. **listar**
    - Lista todas as fichas de encontrista_inscricao.
2. **buscar**
    - Busca por ID.
3. **criar**
    - Valida com Zod todos os campos (nome, endereço, dados do casal, filhos, etc.).
    - Insere na tabela.

4. **atualizar**
    - Atualiza a ficha.
5. **deletar**
    - Remove ficha (admin).

⏱️ **controllers/momentos.controller.js**

Função: cronograma (momentos do evento).

1. **listarMomentos**
    - Lista todos os momentos do banco.
2. **listarPorEvento**
    - Só momentos de um evento, ordenados por ordem.
3. **buscarMomento**
    - Momento específico.
4. **criarMomento**
    - Valida dados com Zod.
    - Se ordem não for enviada, calcula a próxima ordem disponível dentro daquele evento.
    - Insere.
5. **atualizarMomento**
    - Atualização parcial.
6. **deletarMomento**
    - Remove o momento.

🧑‍🏫 **controllers/coordenadores.controller.js**

Função: gerenciar papel de coordenador.

1. **listarCoordenadores**
    - Lista pessoas com role = 'admin'.
2. **promoverCoordenador**
    - Recebe pessoa_id.
    - Atualiza role da pessoa para admin.
3. **removerCoordenador**
    - Volta o role para user.
4. **listarLideres**
    - Para um determinado evento_id, lista teamrole onde is_leader = true.

💳 **controllers/pagamento.controller.js**

Função: criar pagamento (PIX / cartão) ligado a uma inscrição.

1. **criarPagamentoPix**
    - Recebe inscricaoId na URL.
    - Busca a inscrição e seu valor.

- Cria um pagamento Mercado Pago com payment_method_id: "pix".
- Salva em pagamentos_inscricao (com mp_payment_id, status, etc.).
- Devolve os dados do PIX (qr_code, etc.)

2. **criarPagamentoCartao**
    - Parecido com o PIX, mas com cartão:
        - recebe token, installments, etc.
    - Cria o pagamento no Mercado Pago.
    - Salva os dados no banco.
    - Retorna status.

Obs.: A confirmação final (aprovado / recusado) é tratada no **webhook**, abaixo.

🌐 **controllers/webhook.controller.js**

Função: receber notificações do Mercado Pago.

- **mpWebhook(req, res)**
    - Mercado Pago chama essa rota com { type, data: { id } }.
    - Busca detalhes do pagamento via SDK do Mercado Pago.
    - Atualiza:
        - pagamentos_inscricao com mp_status, mp_status_detail, raw_payload.
    - Se status === "approved":
        - Atualiza inscricoes.status = 'confirmed'.
    - Responde 200 OK pro Mercado Pago.

🧪 **controllers/dev.controller.js**

Função: utilidades de desenvolvimento (debug, teste).

Como a gente não mexeu ele em detalhe aqui, costuma ser:

- testezinhos de conexão
- rotas de "health check"
- talvez algo pra ver env, etc.

Eu recomendo:

- manter só em ambiente de desenvolvimento
- não expor em produção

📂 Pasta /routes

Cada arquivo de rotas só faz o "mapa":

- URL + método HTTP → função do controller
- Aplica middlewares (authMiddleware, requireRole) onde necessário.

Exemplos:

routes/auth.routes.js

- POST /auth/register → register
- POST /auth/login → login

routes/admin.routes.js

- POST /admin/create-user → createUserByAdmin
  (protegida com authMiddleware lá no server.js)

routes/pessoas.routes.js

- GET /pessoas → listarPessoas
- GET /pessoas/:id → buscarPessoa
- POST /pessoas → criarPessoa
- PUT /pessoas/:id → atualizarPessoa
- PATCH /pessoas/:id/senha → atualizarSenha
- DELETE /pessoas/:id → deletarPessoa

E assim por diante para:

- eventos.routes.js
- equipe.routes.js
- equipesEvento.routes.js
- teamrole.routes.js
- inscricoes.routes.js
- momentos.routes.js
- coordenadores.routes.js
- encontristaInscricao.routes.js
- pagamento.routes.js
- webhook.routes.js
- dev.routes.js