MePaga API - Documentação de Arquitetura

API REST para divisão de despesas em grupo, construída com **Clean Architecture** e **Clean Code**.

Tecnologias

Tecnologia	Função
Node.js + TypeScript	Runtime e linguagem
Fastify	Framework HTTP
Prisma 7	ORM e migrações
PostgreSQL	Banco de dados relacional
bcrypt	Hash de senhas

Estrutura do Projeto

```
src/
├── domain/                    # Regras de negócio puras (sem dependências
externas)
│   ├── entities/              # Interfaces das entidades do domínio
│   │   ├── User.ts
│   │   ├── Group.ts
│   │   ├── Expense.ts
│   │   └── Balance.ts
│   └── repositories/          # Contratos (interfaces) dos repositórios
│       ├── IUserRepository.ts
│       ├── IGroupRepository.ts
│       └── IExpenseRepository.ts
│
├── application/               # Casos de uso (orquestram o domínio)
│   └── usecases/
│       ├── RegisterUser.ts
│       ├── CreateGroup.ts
│       ├── JoinGroup.ts
│       ├── LinkAccount.ts
│       ├── CreateExpense.ts
│       └── GetGroupBalances.ts
│
├── infrastructure/            # Implementações concretas (banco, serviços
externos)
│   ├── database/
```

```
│   │   └── prismaClient.ts
│   └── repositories/
│       ├── PrismaUserRepository.ts
│       ├── PrismaGroupRepository.ts
│       └── PrismaExpenseRepository.ts
│
├── presentation/              # Camada HTTP (entrada/saída)
│   ├── controllers/
│   │   ├── GroupController.ts
│   │   ├── UserController.ts
│   │   ├── ExpenseController.ts
│   │   └── BalanceController.ts
│   └── routes/
│       ├── groupRoutes.ts
│       ├── userRoutes.ts
│       ├── expenseRoutes.ts
│       └── balanceRoutes.ts
│
├── shared/                    # Código compartilhado entre camadas
│   └── errors/
│       └── AppError.ts
│
├── app.ts                     # Composition Root (injeção de dependências)
└── server.ts                  # Ponto de entrada da aplicação
```

Princípios Aplicados

Clean Architecture

O fluxo de dependência é sempre de fora para dentro:

```
Presentation → Application → Domain
                    ↑
Infrastructure ─────┘
```

- **Domain** não conhece nenhuma outra camada. Define entidades e contratos.
- **Application** conhece apenas o Domain. Implementa casos de uso consumindo interfaces de repositório.
- **Infrastructure** implementa os contratos do Domain usando Prisma/PostgreSQL.
- **Presentation** recebe HTTP, converte para o formato do caso de uso e retorna a resposta.

Inversão de Dependência (DIP)

Os casos de uso dependem de **interfaces** (`IUserRepository`, `IGroupRepository`, `IExpenseRepository`), nunca de implementações concretas.

A injeção é feita manualmente no `app.ts` (**Composition Root**).

Single Responsibility Principle (SRP)

Cada classe possui uma única responsabilidade:

- **Controller** → converte request/response HTTP
- **Use Case** → orquestra regra de negócio
- **Repository** → acessa dados

Modelagem do Banco de Dados

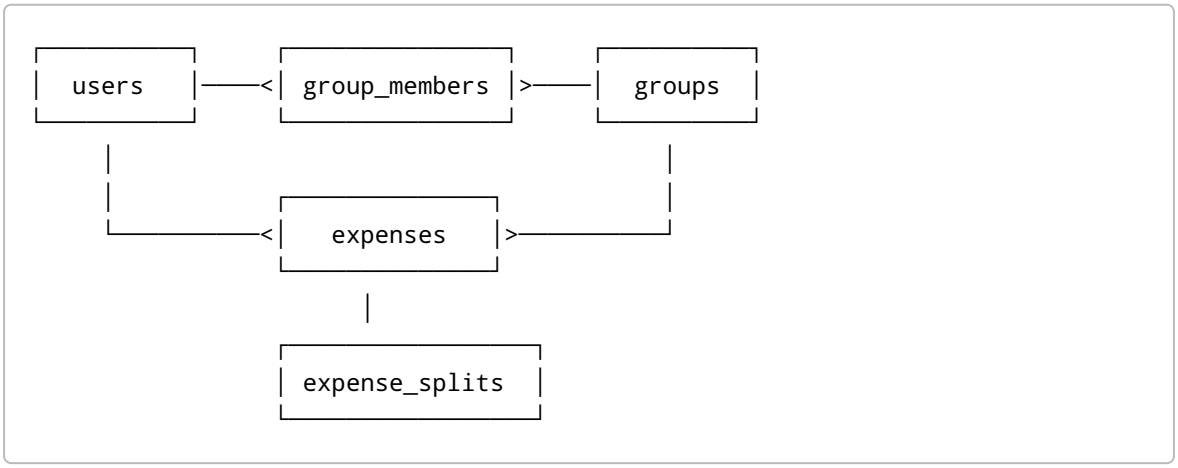

Tabela	Descrição
`users`	Usuários temporários ou cadastrados
`groups`	Grupos de despesas
`group_members`	Relação N:N entre usuários e grupos
`expenses`	Despesas registradas
`expense_splits`	Divisão individual das despesas

Tabela `users`

Coluna	Tipo	Obrigatório	Descrição
`id`	Int	Sim	Chave primária
`name`	VARCHAR(100)	Sim	Nome completo
`email`	VARCHAR(255)	Não	E-mail único
`pix_key`	VARCHAR(100)	Não	Chave Pix
`password_hash`	VARCHAR(255)	Não	Hash bcrypt

Coluna	Tipo	Obrigatório	Descrição
`created_at`	TIMESTAMP	Sim	Data de criação

Ciclo de Vida do Usuário

```
[Usuário temporário] → [Cadastro completo]
 criado ao entrar       POST /api/users/register
 em grupo               (nome, email, senha)
       ↓
[Vincular conta]
PATCH /api/users/link-account
```

Usuários temporários são criados automaticamente pelos fluxos `CreateGroup` e `JoinGroup` quando nenhum `user_id` é informado.

Endpoints da API

Usuários

POST `/api/users/register`

Cadastra um novo usuário.

Request

```
{
  "name": "Lorena Silva",
  "email": "lorena@email.com",
  "password": "minha_senha",
  "pix_key": "lorena@pix.com"
}
```

Response (201)

```
{
  "id": 1,
  "name": "Lorena Silva",
  "email": "lorena@email.com",
  "pix_key": "lorena@pix.com"
}
```

Erros

- `409` → e-mail já utilizado

PATCH `/api/users/link-account`

Vincula conta a usuário temporário.

Request

```
{
  "user_id": 1,
  "email": "carlos@email.com"
}
```

Response (200)

```
{
  "id": 1,
  "name": "Carlos",
  "email": "carlos@email.com",
  "is_temporary": false
}
```

Erros

- `404` → usuário não encontrado
- `409` → conta já vinculada ou e-mail em uso

Grupos

POST `/api/groups`

Cria grupo e usuário criador temporário.

Request

```
{
  "name": "Viagem SP",
  "category": "Viagem",
  "creator_name": "Lorena"
}
```

Response (201)

```
{
  "group": {
    "id": 1,
    "name": "Viagem SP",
    "category": "Viagem",
    "invite_url": "http://localhost:3000/invite/abc123xyz"
  },
  "creator": {
    "id": 1,
    "name": "Lorena",
    "is_temporary": true
  }
}
```

POST `/api/groups/join`

Entra em grupo por convite.

Request

```
{
  "invite_token": "abc123xyz",
  "name": "Carlos",
  "user_id": null
}
```

Response (200)

```
{
  "group_id": 1,
  "user": {
    "id": 2,
    "name": "Carlos",
    "is_temporary": true
  }
}
```

Despesas

POST `/api/groups/:group_id/expenses`

Registra despesa.

Request

```
{
  "description": "Jantar",
  "amount": 150,
  "paid_by": 1,
  "receipt_url": "https://storage.com/recibo.jpg",
  "splits": [
    {
      "user_id": 1,
      "amount_owed": 50
    },
    {
      "user_id": 2,
      "amount_owed": 100
    }
  ]
}
```

Response (201)

```
{
  "id": 1,
  "group_id": 1,
  "paid_by": 1,
  "amount": 150,
  "description": "Jantar",
  "receipt_url": "https://storage.com/recibo.jpg",
  "splits": [
    {
      "user_id": 1,
      "amount_owed": 50
    },
    {
      "user_id": 2,
      "amount_owed": 100
    }
  ]
}
```

Validação

A soma de `amount_owed` deve ser igual ao valor total com tolerância de **R$0,01**.

Balanços

GET `/api/groups/:group_id/balances`

Retorna saldos e dívidas simplificadas.

Response (200)


```
{
  "group_id": 1,
  "who_pays_next": 2,
  "individual_balances": [
    {
      "user_id": 1,
      "name": "Lorena",
      "net_balance": 100
    },
    {
      "user_id": 2,
      "name": "Carlos",
      "net_balance": -100
    }
  ],
  "simplified_debts": [
    {
      "from": {
        "id": 2,
        "name": "Carlos"
      },
      "to": {
        "id": 1,
        "name": "Lorena",
        "pix_key": "lorena@pix.com"
      },
      "amount": 100
    }
  ]
}
```

Motor de Liquidação (Greedy Algorithm)

O cálculo de `simplified_debts` minimiza transações.

Etapas

1. Calcula saldo líquido:

```
saldo = total_pago - total_devido
```

1. Separa:
2. **Devedores** → saldo negativo
3. **Credores** → saldo positivo
4. Ordena por maior valor.
5. Executa casamento ganancioso:
6. maior devedor
7. maior credor
8. transfere menor valor possível
9. atualiza saldos
10. repete

Exemplo:

```
A → B → C
↓
A → C
```

Implementação:

```
src/application/usecases/GetGroupBalances.ts
```

Tratamento de Erros

Erros de negócio usam `AppError`.

Código	Situação
404	Usuário/grupo não encontrado
409	Conflito de negócio
422	Soma inválida da divisão
500	Erro interno

Como Executar

1. Instalar dependências

```
npm install
```

2. Configurar ambiente

```
cp .env.example .env
```

Editar `.env` com PostgreSQL.

3. Gerar Prisma Client

```
npx prisma generate
```

4. Rodar migrações

```
npx prisma migrate dev --name init
```

5. Executar projeto

```
npm run dev
```

Scripts Disponíveis

Script	Comando
Desenvolvimento	`npm run dev`
Build	`npm run build`
Produção	`npm start`
Prisma Generate	`npm run prisma:generate`
Prisma Migrate	`npm run prisma:migrate`
Prisma Studio	`npm run prisma:studio`