🗺️ Mapa Hierárquico do Projeto

```
server.js
 ├── db.js (conexão com banco)
 ├── middlewares/
 │    ├── authMiddleware.js (JWT)
 │    └── adminMiddleware.js (acesso admin)
 ├── routes/
 │    ├── authRoutes.js ──> authController.js
 │    ├── driverRoutes.js ──> driverController.js
 │    ├── passengerRoutes.js ──> passengerController.js
 │    ├── rideRoutes.js ──> rideController.js
 │    ├── paymentRoutes.js ──> paymentController.js
 │    ├── paymentWebhookRoutes.js ──> paymentWebhookController.js
 │    ├── withdrawRoutes.js ──> withdrawController.js
 │    ├── walletRoutes.js ──> walletController.js
 │    ├── transactionRoutes.js ──> transactionController.js
 │    ├── notificationRoutes.js ──> notificationController.js
 │    ├── chatRoutes.js ──> chatController.js
 │    ├── supportRoutes.js ──> supportController.js
 │    ├── ratingRoutes.js ──> ratingController.js
 │    ├── locationRoutes.js ──> locationController.js
 │    ├── vehicleRoutes.js ──> vehicleController.js
 │    ├── webhookRoutes.js ──> webhookController.js
 │    └── adminRoutes.js ──> adminController.js
 ├── services/
 │    ├── ledgerService.js
 │    ├── payoutService.js
 │    └── picpayService.js
 └── jobs/
      ├── payoutJob.js
      └── picpayPollingJob.js
```

📖 Explicação em Lista (A, B, C...)

A) server.js

- Ponto de entrada da aplicação.
- Configura o Express, carrega middlewares e conecta todas as rotas.
- Importa `db.js` para garantir conexão com o banco.

B) db.js

- Faz a conexão com o banco de dados (MySQL/PostgreSQL).
- Centraliza o acesso, para que todos os controllers usem a mesma conexão.

C) middlewares/

1. **authMiddleware.js**
    a. Verifica se o token JWT é válido.
    b. Define qual usuário está logado e se pode acessar a rota.
2. **adminMiddleware.js**
    a. Restringe acesso a certas rotas apenas para administradores.

D) routes/

Cada arquivo de rota define endpoints da API e encaminha para um **controller**:

1. **authRoutes.js → authController.js**
    a. Login, cadastro, recuperação de senha, tokens.
2. **driverRoutes.js → driverController.js**
    a. Cadastro de motoristas, upload de documentos, ativar/desativar motorista.
3. **passengerRoutes.js → passengerController.js**
    a. Cadastro e atualização de passageiros.
4. **rideRoutes.js → rideController.js**
    a. Criar corrida, aceitar corrida, finalizar corrida.
    b. Comunicação entre passageiro e motorista.
5. **paymentRoutes.js → paymentController.js**
    a. Inicia pagamentos pelo PicPay.
    b. Calcula valores de corrida e taxas.
6. **paymentWebhookRoutes.js → paymentWebhookController.js**
    a. Recebe notificações do PicPay.
    b. Atualiza o status de pagamento (aprovado, recusado).

7. **withdrawRoutes.js → withdrawController.js**
    a. Solicitação de saque do motorista (vale-pix).
    b. Aprovação/rejeição do saque.
8. **walletRoutes.js → walletController.js**
    a. Gerencia saldo do usuário.
    b. Exibe histórico de créditos/débitos.
9. **transactionRoutes.js → transactionController.js**
    a. Registra todas as movimentações financeiras.
    b. Garante rastreabilidade no sistema.
10. **notificationRoutes.js → notificationController.js**
    a. Envia notificações push ou internas (ex: corrida aceita).
11. **chatRoutes.js → chatController.js**
    a. Mensagens entre motorista e passageiro durante a corrida.
12. **supportRoutes.js → supportController.js**
    a. Abertura e gerenciamento de chamados de suporte.
13. **ratingRoutes.js → ratingController.js**
    a. Avaliações e notas entre motorista e passageiro.
14. **locationRoutes.js → locationController.js**
    a. Atualização da localização em tempo real.
    b. Suporte ao tracking do passageiro e motorista.
15. **vehicleRoutes.js → vehicleController.js**
    a. Cadastro e edição de veículos vinculados a motoristas.
16. **webhookRoutes.js → webhookController.js**
    a. Webhooks gerais (para integrações futuras além do PicPay).
17. **adminRoutes.js → adminController.js**
    a. Funções administrativas: listar motoristas, passageiros, corridas.

E) services/

1. **ledgerService.js**
    a. Registra entradas e saídas no "livro-caixa" interno da plataforma.
2. **payoutService.js**
    a. Executa transferências de valores da carteira para motoristas.
3. **picpayService.js**
    a. Abstrai a comunicação com a API do PicPay (criar cobrança, consultar status).

F) jobs/

1. **payoutJob.js**
    a. Executa pagamentos programados automaticamente.

2. **picpayPollingJob.js**
    a. Consulta periodicamente o PicPay para atualizar status de pagamentos pendentes.