Express with Prisma Project Structure

Express integrated with Prisma ORM for type-safe database operations with auto-generated types.

#express #nodejs #typescript #prisma #database #postgresql

PNG PDF Copy Prompt

Project Directory

```
myproject/
  src/ Application sou…
    index.ts Server startup
    app.ts Express app set…
    config/
      index.ts
      database.ts Prisma client i…
    routes/
      index.ts
      user.routes.ts
      post.routes.ts
    controllers/
      user.controller.ts
      post.controller.ts
    services/ Business logic
      user.service.ts Uses Prisma cli…
      post.service.ts
    middleware/
      error-handler.ts Prisma errors
      validation.ts
    types/
      index.ts
  prisma/ Prisma configur…
    schema.prisma Database schema
    seed.ts Seed script
    migrations/
      .gitkeep
  tests/
    setup.ts
    user.test.ts
  package.json
  tsconfig.json
  .env.example DATABASE_URL he…
  .gitignore
```

Why This Structure?

Prisma generates TypeScript types from `schema.prisma`—no manual interface definitions. Services inject the Prisma client and get full autocompletion. Migrations version-control your schema changes.

Key Directories

`prisma/schema.prisma` - Single source of truth for models

`prisma/migrations/` - Database version history

`src/config/database.ts` - Prisma client singleton

`src/services/` - Business logic uses Prisma directly

Getting Started

1. `npm install`
2. `cp .env.example .env` (set DATABASE_URL)
3. `npx prisma migrate dev`
4. `npx prisma db seed`
5. `npm run dev`

Best Practices

- Export a singleton Prisma client from `config/database.ts`
- Use `select` and `include` to avoid over-fetching
- Handle `PrismaClientKnownRequestError` in error middleware
- Run `prisma generate` in CI after migrations
- Use transactions for multi-table writes

Trade-offs

`No repository abstraction` - Services use Prisma directly—simpler but coupled

`Prisma schema lock-in` - Schema syntax specific to Prisma

`Complex queries` - May need `$queryRaw` for advanced SQL