Express MVC TypeScript Project Structure

TypeScript MVC with controllers, services, and models. Production-ready Express architecture.

#express #nodejs #typescript #mvc #rest #api

PNG PDF Copy Prompt

Project Directory

```
myproject/
  src/ All TypeScript …
    index.ts Entry point
    app.ts Express app set…
    config/ Configuration
      index.ts Config loader
      database.ts
      logger.ts
    controllers/ Route handlers
      index.ts Export all cont…
      user.controller.ts
      auth.controller.ts
    services/ Business logic
      index.ts
      user.service.ts
      auth.service.ts
    models/ Data models
      index.ts
      user.model.ts
    routes/ Route definitio…
      index.ts Route aggregator
      user.routes.ts
      auth.routes.ts
    middleware/ Express middlew…
      auth.middleware.ts
      validation.middleware.ts
      error.middleware.ts
    types/ TypeScript defi…
      index.ts
      express.d.ts Extend Express …
    utils/
      async-handler.ts Wrap async rout…
      api-error.ts Custom error cl…
  tests/
    setup.ts Test configurat…
    user.test.ts
    auth.test.ts
  package.json
  tsconfig.json
  .env
  .env.example
  .gitignore
```

Why This Structure?

MVC separates concerns cleanly: Controllers handle HTTP, Services contain business logic, Models define data shapes. TypeScript adds type safety across layers. This pattern scales well for teams and makes testing straightforward.

Key Directories

controllers/ - Parse requests, call services, format responses

services/ - Business logic, database operations, external APIs

models/ - TypeScript interfaces and database schemas

middleware/ - Auth, validation, error handling, logging

routes/ - Route definitions wired to controllers

Controller Pattern

```typescript
// src/controllers/user.controller.ts
export class UserController {
  constructor(private userService: UserService) {}

  getUser = async (req: Request, res: Response) => {
    const user = await this.userService.findById(req.param
    res.json(user);
  };
}
```

Getting Started

1. `npm init -y`
2. `npm install express dotenv`
3. `npm install -D typescript @types/node @types/express ts-node`
4. `npx tsc --init`
5. `cp .env.example .env`
6. `npm run dev` # Add: "dev": "ts-node src/index.ts"

When To Use This

- Production APIs with multiple endpoints
- Teams of 2+ developers
- Need type safety and IDE support
- APIs with complex business logic
- Projects requiring long-term maintenance

Trade-offs

Build step - Must compile TypeScript before deploy

More boilerplate - Each layer requires wiring up

Learning curve - Need to understand TypeScript + patterns

Naming Conventions

Controllers - `{resource}.controller.ts` (user.controller.ts)

Services - `{resource}.service.ts` (user.service.ts)

Routes - `{resource}.routes.ts` (user.routes.ts)

Middleware - `{purpose}.middleware.ts` (auth.middleware.ts)