Nuxt 3 Full-Stack Project Structure

Full-stack Nuxt with Nitro server, API routes, and Prisma. Ready for real applications.

#nuxt #vue #typescript #fullstack #prisma #nitro

PNG PDF Copy Prompt

Project Directory

```
my-fullstack-app/
  app/  Application lay…
    app.vue  Root component
    app.config.ts
  server/  Nitro server (a…
    api/  API routes
      auth/
        login.post.ts  POST /api/auth/…
        logout.post.ts
        me.get.ts  GET /api/auth/me
      users/
        index.get.ts  GET /api/users
        index.post.ts  POST /api/users
        [id].get.ts  GET /api/users/…
        [id].put.ts
        [id].delete.ts
    middleware/  Server middlewa…
      auth.ts  Auth verificati…
    utils/  Server-only uti…
      db.ts  Prisma client i…
      auth.ts  Auth helpers
  pages/  File-based rout…
    index.vue
    dashboard/
      index.vue  Protected route
    auth/
      login.vue
      register.vue
  components/  Auto-imported c…
    ui/  Base UI compone…
      Button.vue
      Input.vue
      Card.vue
    layout/
      Header.vue
      Footer.vue
      Sidebar.vue
  composables/  Shared composab…
    useAuth.ts  Auth state and …
    useApi.ts  Typed API fetch…
  layouts/
    default.vue
    auth.vue  Minimal layout …
    dashboard.vue  With sidebar
  middleware/  Route middlewar…
    auth.ts  Protect routes
  prisma/  Database schema
    schema.prisma  Models and rela…
    migrations/
  types/  Shared TypeScri…
    index.d.ts
    api.d.ts  API request/res…
  public/
    favicon.ico
  nuxt.config.ts
  package.json
  tsconfig.json
  .env  DATABASE_URL, s…
  .env.example
  .gitignore
```

Why This Structure?

This structure is designed for real-world full-stack applications. `Nitro` provides a powerful server layer with `file-based API routes`. `Prisma` handles database access with type safety. The separation between `server/` and client code is clear—server utilities never leak to the browser.

Key Directories

server/api/ - `API routes` —filename becomes HTTP method (`login.post.ts`)

server/utils/ - `Server-only code` (Prisma client, auth helpers)

server/middleware/ - Runs before every API request

middleware/ - `Client-side route guards` (auth protection)

prisma/ - Database schema and migrations

types/ - `Shared TypeScript types` for API contracts

API Route Naming

users/index.get.ts - `GET /api/users`

users/index.post.ts - `POST /api/users`

users/[id].get.ts - `GET /api/users/:id`

auth/login.post.ts - `POST /api/auth/login`

Getting Started

```
npx nuxi@latest init my-fullstack-app
cd my-fullstack-app
npm install prisma @prisma/client
npx prisma init
Configure DATABASE_URL in .env
npx prisma migrate dev --name init
npm run dev
```

When To Use This

- Building a SaaS or dashboard application
- Need authentication and user management
- Projects with database requirements
- Full-stack teams using Vue.js
- Applications requiring API routes

Trade-offs

More complexity - Many folders to understand initially

Prisma lock-in - Can swap for Drizzle or kysely if preferred

Server-side auth - Must handle session/JWT yourself or use nuxt-auth

Best Practices

- Keep `server/utils/` for database and auth only—no business logic
- Use `types/` for API contracts shared between client and server
- Protect routes with `middleware/`, not in page components
- Use `useFetch()` with typed responses from API routes
- Run `prisma generate` after schema changes