Axum REST API Project Structure

Full REST API with SQLx, tower middleware, and structured handlers.

#rust #axum #rest #api #sqlx #tower #backend

PNG PDF Copy Prompt

Project Directory

```
myproject/
    Cargo.toml
    Cargo.lock
    .gitignore
    README.md
    .env  DB connection, …
    .env.example
    sqlx-data.json  Offline query c…
  migrations/  SQLx migrations
    001_create_users.sql
  src/
    main.rs  Bootstrap and s…
    lib.rs  Library root fo…
    config/
      mod.rs
      settings.rs  Typed config wi…
    routes/  Route modules
      mod.rs  Combines all ro…
      users.rs
      health.rs
    handlers/  Request handlers
      mod.rs
      users.rs
      health.rs
    models/  Domain types
      mod.rs
      user.rs
    db/  Database layer
      mod.rs
      pool.rs  Connection pool…
      users.rs  User queries
    middleware/
      mod.rs
      auth.rs  JWT or session …
      logging.rs  Request logging
    error/
      mod.rs  AppError and re…
    extractors/  Custom extracto…
      mod.rs
      json.rs  Validated JSON …
```

Why This Structure?

This structure separates HTTP concerns from database access using Axum's extractor pattern. SQLx provides compile-time checked queries, while tower middleware handles cross-cutting concerns. Each layer has a clear responsibility—routes define paths, handlers process requests, db executes queries.

Key Directories

src/routes/ - Route definitions, maps paths to handlers

src/handlers/ - Request processing, calls db layer

src/db/ - SQLx queries and connection pool

src/middleware/ - Tower layers for auth, logging, CORS

src/extractors/ - Custom request extractors with validation

Handler with State

```rust
// src/handlers/users.rs
use axum::{extract::State, Json};
use crate::{db, error::AppError, models::User};

pub async fn list_users(
    State(pool): State,
) -> Result>, AppError> {
    let users = db::users::find_all(&pool).await?;
    Ok(Json(users))
}
```

Getting Started

1. `cargo new myproject && cd myproject`
2. Add `axum`, `sqlx`, `tokio`, `tower-http` to `Cargo.toml`
3. `sqlx database create && sqlx migrate run`
4. `cargo run`

When To Use This

- Building REST APIs with database access
- Need compile-time query verification
- Authentication and middleware requirements
- APIs with 10+ endpoints
- Team projects with clear layer separation

Trade-offs

More boilerplate - Rust requires explicit error handling

Build times - SQLx compile-time checks add to build time

Module ceremony - mod.rs files needed for each folder

Naming Conventions

Modules - lowercase, singular (handler, model, db)

Files - Lowercase with underscores (user_handler.rs)

Types - PascalCase (User, CreateUserRequest)

Functions - snake_case (find_by_id, create_user)