SECTION 6 — Project Folder Structure Explained

Understanding config, middlewares, modules, utils, scripts, and .env

1. Why Folder Structure Matters

In small projects, developers often place all files in one folder.

In enterprise applications, this approach fails because:

- Code becomes unmanageable.
- Responsibilities are mixed.
- Debugging becomes difficult.
- Scalability becomes impossible.

A clean folder structure ensures:

- Separation of concerns
- Clear ownership of logic
- Maintainability
- Scalability
- Microservice readiness

Monsta follows a structured, modular architecture.

2. High-Level Project Structure

```
src/
 ├── config/
 ├── middlewares/
 ├── modules/
 │     ├── identity/
 │     ├── governance/
 ├── utils/
 ├── scripts/
 ├── server.js
 ├── .env
```

Each folder has a clearly defined purpose.

3. The config Folder

```
config/
 ├── db.js
 ├── env.js
 ├── permission.matrix.js
```

Purpose

The config folder contains system-level configuration.

Configuration means:

- Settings required for the system to run
- Not business logic
- Not request handling

3.1 db.js

Responsible for:

- Connecting to MongoDB
- Handling database connection errors

Keeps database setup separate from server logic.

3.2 env.js (if used)

Responsible for:

- Centralizing environment variable access
- Validating required environment variables

Prevents hardcoding secrets.

3.3 permission.matrix.js

Defines:

- What permissions exist
- Which roles are allowed to perform which actions

This is system configuration, not business logic.

4. The middlewares Folder

```
middlewares/
 ├── auth.middleware.js
 ├── permission.middleware.js
 ├── error.middleware.js
 ├── policyGuard.middleware.js
```

Purpose

Middleware handles cross-cutting concerns.

These are tasks that apply to multiple routes.

Examples:

- Authentication
- Authorization

- Error handling
- Policy validation

What Is Middleware?

Middleware is a function that runs before the controller.

Flow:

```
Request → Middleware → Controller → Response
```

Middleware can:

- Allow request
- Block request
- Modify request
- Attach data to request

Common Middleware in Monsta

auth.middleware.js (protect)

- Verifies JWT token
- Attaches user to request

permission.middleware.js (authorize)

- Checks if user has required permission

error.middleware.js

- Handles all application errors
- Sends standardized error responses

policyGuard.middleware.js

- Ensures active platform policy exists

5. The modules Folder

```
modules/
 ├── identity/
 ├── governance/
```

Purpose

The modules folder contains domain-specific business logic.

Each module represents a bounded domain.

What Is a Module?

A module is:

- A self-contained domain
- With its own routes
- Controllers
- Services
- Models

Inside a Module

Example: identity module

```
identity/
 ├── models/
 │     └── user.model.js
 ├── auth.routes.js
 ├── auth.controller.js
 ├── auth.service.js
```

Responsibilities of Each Layer

Routes

- Define API endpoints
- Attach middleware
- Connect route to controller

Controller

- Handles request and response
- Calls service
- Does not contain heavy business logic

Service

- Contains business logic
- Handles validation
- Interacts with database models

Model

- Defines database schema
- Defines structure of stored data

Why Separate Controller and Service?

Bad practice:

- Writing business logic directly in controller.

Good practice:

- Controller → orchestrates.
- Service → contains logic.

This makes code:

- Testable
- Maintainable
- Reusable

6. The utils Folder

```
utils/
 ├── jwt.util.js
 ├── password.util.js
 ├── response.util.js
 ├── appError.util.js
 ├── asyncHandler.util.js
```

Purpose

Utilities contain reusable helper functions.

These are not tied to a specific module.

They support multiple parts of the system.

7. The scripts Folder

scripts/

Purpose:

- Contains scripts for automation.
- Used for:
  - Seeding database
  - Creating admin user
  - Migration scripts

This folder is optional but useful in production systems.

8. The .env File

The .env file contains environment variables.

Example:

PORT=5000

MONGO_URI=your_mongo_connection

JWT_SECRET=your_secret

Important rules:

- Never commit .env to GitHub.
- Never hardcode secrets in code.
- Use different values for development and production.

9. How All Folders Work Together

Example: Creating Platform Policy

Route → Middleware → Controller → Service → Model → Database

Folders involved:

- modules/governance
- middlewares
- utils
- config
- server.js

Everything has a defined role.

10. Architectural Principles in This Structure

This structure enforces:

1. Separation of concerns
2. Single responsibility principle
3. Modular isolation
4. Scalable architecture
5. Microservice readiness

No folder contains mixed responsibilities.

11. What Students Must Understand

- server.js boots the application.
- config contains system configuration.
- middlewares handle cross-cutting concerns.
- modules contain domain logic.
- utils contain reusable helpers.
- .env stores secrets.
- scripts automate repetitive tasks.

Understanding structure is more important than writing endpoints.

12. Summary

By the end of this section, students should understand:

- Why folder structure matters
- What each folder does
- How modules are structured internally
- Why controller and service are separated
- How middleware fits into request lifecycle
- Why configuration is separated

This section prepares students to understand the implementation layer clearly.