In Node.js development, choosing the right **folder structure** helps maintain scalability, readability, and maintainability of your codebase. Here are the **common types of folder structures** used in Node.js projects:

**🔹 1. Basic Folder Structure (For Small Projects)**

bash

CopyEdit

my-app/

├── node\_modules/

├── public/

├── src/

│ ├── index.js

├── package.json

* **Use case**: Simple projects or quick prototypes.
* Minimal setup, everything handled in one or two files.

**🔹 2. MVC Structure (Model-View-Controller)**

bash

CopyEdit

my-app/

├── controllers/

├── models/

├── routes/

├── views/ # If using templating engine like EJS, Pug

├── public/

├── config/

├── index.js

├── package.json

* **Use case**: Clean separation of concerns.
* **Controller**: Handles logic.
* **Model**: Handles database interactions.
* **View**: Handles UI (optional, for web apps).

**🔹 3. Feature-Based Structure**

bash

CopyEdit

my-app/

├── features/

│ ├── auth/

│ │ ├── auth.controller.js

│ │ ├── auth.routes.js

│ │ └── auth.service.js

│ ├── user/

│ │ ├── user.controller.js

│ │ ├── user.routes.js

│ │ └── user.service.js

├── config/

├── utils/

├── middlewares/

├── index.js

├── package.json

* **Use case**: Mid-to-large applications.
* Organizes files **by feature/domain**, not by type.
* Promotes **modularity** and easier scalability.

**🔹 4. Service-Oriented / Layered Architecture**

bash

CopyEdit

my-app/

├── controllers/

├── services/

├── repositories/ # Data access logic

├── models/

├── routes/

├── middlewares/

├── utils/

├── config/

├── index.js

* **Use case**: When business logic is complex.
* Clear layers: **Controller → Service → Repository**.

**🔹 5. Modular Monolith Structure**

bash

CopyEdit

my-app/

├── modules/

│ ├── product/

│ │ ├── product.controller.js

│ │ ├── product.service.js

│ │ ├── product.model.js

│ │ └── product.routes.js

│ ├── order/

│ └── user/

├── shared/ # Common utils, helpers, middlewares

├── config/

├── app.js

├── package.json

* **Use case**: Preparing for **microservices** in the future.
* Keeps modules self-contained and decoupled.

**🔹 6. Microservices-Oriented Structure**

Each microservice is in its **own folder/repo**, with its own:

bash

CopyEdit

user-service/

order-service/

payment-service/

* Each has:

bash

CopyEdit

├── src/

├── tests/

├── Dockerfile

├── package.json

**Which One Should You Use?**

* ✅ **Beginner / Small App** → Basic or MVC
* ✅ **Growing App** → Feature-based or Layered
* ✅ **Team Development** → Modular or Feature-based
* ✅ **Future Microservices** → Modular Monolith