**📂 test/ Directory — Purpose & Best Practices**

The test/ folder is the central place to organize all testing-related files. NestJS scaffolds this by default, but we enhance it for **structured, scalable** testing support.

**✅ Recommended Folder Structure for test/**

test/

├── app.e2e-spec.ts # Sample e2e test for app module

├── jest-e2e.json # Jest config for E2E

├── unit/

│ ├── user.service.spec.ts # Unit test example

│ └── auth.service.spec.ts # Unit test example

├── integration/

│ └── user-auth.integration.spec.ts # Auth integration test

├── mocks/

│ ├── user.mock.ts # Sample mock user data

│ └── auth.mock.ts # Sample mock auth token

└── utils/

└── setup-test-app.ts # Bootstrap Nest test app

**🔍 Folder/Files Breakdown & Example**

**📁 unit/**

* **Purpose**: Isolated testing of services, utils, and classes without DB or external APIs.

**Example: user.service.spec.ts**

import { Test, TestingModule } from '@nestjs/testing';

import { UserService } from '../../src/modules/user/user.service';

describe('UserService', () => {

let service: UserService;

beforeEach(async () => {

const module: TestingModule = await Test.createTestingModule({

providers: [UserService],

}).compile();

service = module.get<UserService>(UserService);

});

it('should be defined', () => {

expect(service).toBeDefined();

});

it('should return user', async () => {

const result = await service.getUserById(1);

expect(result).toBeTruthy();

});

});

**📁 integration/**

* **Purpose**: Tests combining multiple services/modules with real logic (may include test DB).

**Example: user-auth.integration.spec.ts**

import \* as request from 'supertest';

import { INestApplication } from '@nestjs/common';

import { Test } from '@nestjs/testing';

import { AppModule } from '../../src/app.module';

describe('AuthController (integration)', () => {

let app: INestApplication;

beforeAll(async () => {

const moduleFixture = await Test.createTestingModule({

imports: [AppModule],

}).compile();

app = moduleFixture.createNestApplication();

await app.init();

});

it('/auth/login (POST)', async () => {

return request(app.getHttpServer())

.post('/auth/login')

.send({ email: 'admin@example.com', password: 'admin' })

.expect(201);

});

afterAll(async () => {

await app.close();

});

});

**📁 mocks/**

* **Purpose**: Store mock objects/functions/stubs for reuse in tests.

**Example: user.mock.ts**

export const mockUser = {

id: 1,

email: 'admin@example.com',

password: 'hashedPassword',

};

**📁 utils/**

* **Purpose**: Bootstrap or teardown logic for testing

**Example: setup-test-app.ts**

import { Test } from '@nestjs/testing';

import { AppModule } from '../../src/app.module';

import { INestApplication } from '@nestjs/common';

export async function createTestApp(): Promise<INestApplication> {

const moduleRef = await Test.createTestingModule({

imports: [AppModule],

}).compile();

const app = moduleRef.createNestApplication();

await app.init();

return app;

}

**📄 app.e2e-spec.ts**

* NestJS scaffolded E2E test
* Can be replaced by your own grouped tests in integration/

**📄 jest-e2e.json**

**Sample jest-e2e.json:**

{

"moduleFileExtensions": ["js", "json", "ts"],

"rootDir": "../",

"testRegex": ".e2e-spec.ts$",

"transform": {

"^.+\\.(t|j)s$": "ts-jest"

},

"setupFiles": ["dotenv/config"],

"testEnvironment": "node"

}

**🧪 Run Your Tests**

**Run all tests:**

npm run test

**Run only unit tests:**

npm run test:unit

**Run only integration/e2e:**

npm run test:e2e

You can configure these in package.json like:

"scripts": {

"test": "jest",

"test:unit": "jest --config test/jest-unit.json",

"test:e2e": "jest --config test/jest-e2e.json"

}

**🛡 Best Practices for Production Testing**

| **🟢 Best Practice** | **✅ Benefit** |
| --- | --- |
| Use test DB for integration | Prevent production data mutation |
| Separate test config (.env.test) | Isolate environments |
| Use mocking where possible | Speed + isolation |
| Automate in CI (GitHub Actions) | Reliable pre-deployment checks |
| Use coverage reports | Track quality and gaps |

**✅ Goal**

1. Add **test coverage reporting** using jest.
2. Enforce **minimum coverage thresholds** (fail the build if coverage is low).
3. Upload coverage report as a GitHub Actions artifact.

**🧪 Step 1: Configure Jest for Coverage**

Update jest.config.ts (or jest.config.js) with coverage settings.

**📄 jest.config.ts (production-ready)**

export default {

moduleFileExtensions: ['js', 'json', 'ts'],

rootDir: 'src',

testRegex: '.\*\\.spec\\.ts$',

transform: {

'^.+\\.(t|j)s$': 'ts-jest',

},

collectCoverage: true, // Enables coverage

coverageDirectory: '../coverage', // Output directory

coverageReporters: ['json', 'lcov', 'text', 'clover'], // Format options

coveragePathIgnorePatterns: ['/node\_modules/', '/test/'],

coverageThreshold: {

global: {

branches: 80,

functions: 85,

lines: 85,

statements: 85,

},

},

testEnvironment: 'node',

};

💡 Adjust coverage thresholds (branches, functions, lines, statements) based on your requirements. When coverage is below threshold, **CI will fail.**

**🧰 Step 2: Add Coverage Commands in package.json**

"scripts": {

"test": "jest",

"test:cov": "jest --coverage",

"test:watch": "jest --watch",

"test:e2e": "jest --config test/jest-e2e.json",

"test:unit": "jest --config test/jest-unit.json"

}

**⚙️ Step 3: Update GitHub Actions CI Workflow**

Update ci.yml to include a test job with coverage enforcement and upload coverage as artifact.

**📄 .github/workflows/ci.yml**

name: CI

on:

push:

branches: [main]

pull\_request:

branches: [main]

jobs:

test:

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [18.x]

steps:

- name: ⬇️ Checkout code

uses: actions/checkout@v3

- name: 🟢 Setup Node.js

uses: actions/setup-node@v3

with:

node-version: ${{ matrix.node-version }}

- name: 📦 Install dependencies

run: npm ci

- name: 🧪 Run Tests with Coverage

run: npm run test:cov

- name: 📤 Upload coverage report

uses: actions/upload-artifact@v3

with:

name: coverage-report

path: coverage/

- name: ✅ Coverage Summary

run: cat coverage/lcov-report/index.html || echo "View HTML report in artifact"

✅ If the coverage is **below threshold**, the npm run test:cov step **will fail** the pipeline automatically due to the coverageThreshold config.

**📈 Optional: Add Coverage Badge**

Use a service like [Coveralls](https://coveralls.io/) or [Codecov](https://about.codecov.io/) to:

* Track coverage over time
* Show a badge on your README
* Compare PRs with baseline

Want help integrating Coveralls or Codecov?

**✅ Summary**

| **Step** | **Description** |
| --- | --- |
| 🧪 Jest config | Enabled coverage collection and thresholds |
| 📁 CI Workflow | Runs tests, checks thresholds, uploads report |
| 📊 Optional | Add Codecov/Coveralls for dashboards & badges |