TestingModuleBuilder

August 17, 2026

TestingModuleBuilder

Kind: Class

Source: `packages/testing/testing-module.builder.ts`

Part of: Testing

`TestingModuleBuilder` configures and compiles a NestJS testing module before it is used in tests. It supports overriding providers, modules, guards, pipes, filters, and interceptors, configuring a custom logger, and supplying a mocker for unresolved dependencies. Calling `compile()` produces a `TestingModule` that can resolve and exercise application components in isolation.

Methods

Method	Signature	Returns
`setLogger`	`setLogger(testingLogger: LoggerService)`	`void`
`overridePipe`	`overridePipe(typeOrToken: T)`	`OverrideBy`
`useMocker`	`useMocker(mocker: MockFactory)`	`TestingModuleBuilder`
`overrideFilter`	`overrideFilter(typeOrToken: T)`	`OverrideBy`
`overrideGuard`	`overrideGuard(typeOrToken: T)`	`OverrideBy`
`overrideInterceptor`	`overrideInterceptor(typeOrToken: T)`	`OverrideBy`
`overrideProvider`	`overrideProvider(typeOrToken: T)`	`OverrideBy`
`overrideModule`	`overrideModule(moduleToOverride: ModuleDefinition)`	`OverrideModule`
`compile`	`compile(options: Pick<NestApplicationContextOptions, 'snapshot'	'preview'>)`

Diagram

```
graph LR
  A[Test.createTestingModule metadata] --> B[TestingModuleBuilder]
  B --> C[Configure logger or mocker]
  B --> D[Override providers/modules]
  B --> E[Override guards/pipes/filters/interceptors]
  C --> F[compile()]
  D --> F
  E --> F
  F --> G[TestingModule]
  G --> H[module.get()]
  G --> I[Application or unit tests]
```

Usage

```
import { Test } from '@nestjs/testing';
import { UsersService } from './users.service';
import { UsersRepository } from './users.repository';

describe('UsersService', () => {
  let usersService: UsersService;

  beforeEach(async () => {
    const moduleRef = await Test.createTestingModule({
      providers: [UsersService, UsersRepository],
    })
      .overrideProvider(UsersRepository)
      .useValue({
        findById: jest.fn().mockResolvedValue({
          id: 'user-1',
          email: 'user@example.com',
        }),
      })
      .useMocker((token) => {
        if (token === 'EMAIL_CLIENT') {
          return { send: jest.fn() };
        }
      })
      .compile();

    usersService = moduleRef.get(UsersService);
  });

  it('returns a user', async () => {
    await expect(usersService.findById('user-1')).resolves.toEqual({
      id: 'user-1',
      email: 'user@example.com',
    });
```

```
  });
});
```

AI Coding Instructions

- Create the builder through `Test.createTestingModule()` and call `compile()` only after all module configuration and overrides are complete.
- Use `overrideProvider()`, `overrideGuard()`, `overridePipe()`, `overrideFilter()`, or `overrideInterceptor()` to replace production dependencies with deterministic test doubles.
- Use `useMocker()` for automatic fallback mocks, but explicitly override dependencies whose behavior is important to the test scenario.
- Keep overrides scoped to the testing module; do not modify production module metadata or global application configuration for test-specific behavior.
- Retrieve compiled dependencies from `TestingModule` with `moduleRef.get()` and reset Jest mocks between tests when mock state can leak.

How it works

`TestingModuleBuilder` is the mutable builder returned by `Test.createTestingModule(metadata, options)`. It accepts module metadata and optional `moduleIdGeneratorAlgorithm` configuration, creates a `NestContainer`, and turns the metadata into a dynamically decorated `RootTestModule`. packages/testing/test.ts:11-16 packages/testing/testing-module.builder.ts:29-32 packages/testing/testing-module.builder.ts:49-56 packages/testing/testing-module.builder.ts:195-199

- `setLogger(logger)` stores a `LoggerService` and returns the same builder. During `compile()`, it globally overrides Nest's logger with that service; absent an explicitly set logger, it installs `TestingLogger`. `TestingLogger` suppresses `log`, `warn`, `debug`, and `verbose`, while forwarding `error` to `ConsoleLogger`. packages/testing/testing-module.builder.ts:58-61 packages/testing/testing-module.builder.ts:100-100 packages/testing/testing-module.builder.ts:201-203 packages/testing/services/testing-logger.service.ts:6-17
- `overridePipe`, `overrideFilter`, `overrideGuard`, and `overrideInterceptor` register a non-provider replacement for a supplied type or token. `overrideProvider` registers a provider replacement instead. Each returns an `OverrideBy` object whose `useValue`, `useClass`, and `useFactory` methods record the replacement and return the builder; factory options accept a required `factory` callback and optional `inject` array, and

the builder records the callback under `useFactory`. A later override for the same type or token replaces the earlier map entry. packages/testing/testing-module.builder.ts:63-86 packages/testing/testing-module.builder.ts:134-153 packages/testing/interfaces/override-by.interface.ts:7-10 packages/testing/interfaces/override-by-factory-options.interface.ts:4-7

- `overrideModule(moduleToOverride).useModule(newModule)` records a module substitution and returns the builder. `compile()` passes all recorded substitutions to dependency scanning, then applies recorded type/token replacements through `container.replace`. packages/testing/testing-module.builder.ts:88-95 packages/testing/testing-module.builder.ts:117-123 packages/testing/testing-module.builder.ts:156-169

- `useMocker(mocker)` stores a callback of type `(token?: InjectionToken) => any` and returns the builder. While creating dependency instances, the testing injector first tries normal resolution; if that throws, it calls the mocker with the unresolved name/token. A falsy mock result, or no mocker, rethrows the original resolution error. For a truthy result, it creates a resolved wrapper and adds/exports a `useValue` provider from the internal core module; if that module is absent, it throws `Expected to have internal core module reference at this point.` packages/testing/testing-module.builder.ts:67-70 packages/testing/testing-module.builder.ts:176-193 packages/testing/interfaces/mock-factory.ts:1-4 packages/testing/testing-instance-loader.ts:7-14 packages/testing/testing-injector.ts:35-55 packages/testing/testing-injector.ts:80-118

- `compile({ snapshot, preview })` is asynchronous and returns a `TestingModule`. It scans the generated root module, applies overrides, creates dependency instances, and applies application providers. With `snapshot: true`, it constructs a `GraphInspector` and switches the global `UuidFactory.mode` to deterministic; otherwise it uses `NoopGraphInspector` and switches that global mode to random. It passes both `preview` and `snapshot`, defaulting each to `false`, to `TestingInjector`. packages/testing/testing-module.builder.ts:97-132 packages/testing/testing-module.builder.ts:176-193

- The returned `TestingModule` is a `NestApplicationContext` constructed with the builder's container, graph inspector, first module in the container as context module, and application configuration. packages/testing/testing-module.builder.ts:125-131 packages/testing/testing-module.builder.ts:171-174 packages/testing/testing-module.ts:26-40

- The builder has no explicit runtime validation for constructor metadata, override inputs, logger, mocker, or `compile` options; errors from scanning and dependency creation are not caught by `compile()`. packages/testing/testing-module.builder.ts:49-56 packages/testing/testing-module.builder.ts:97-123