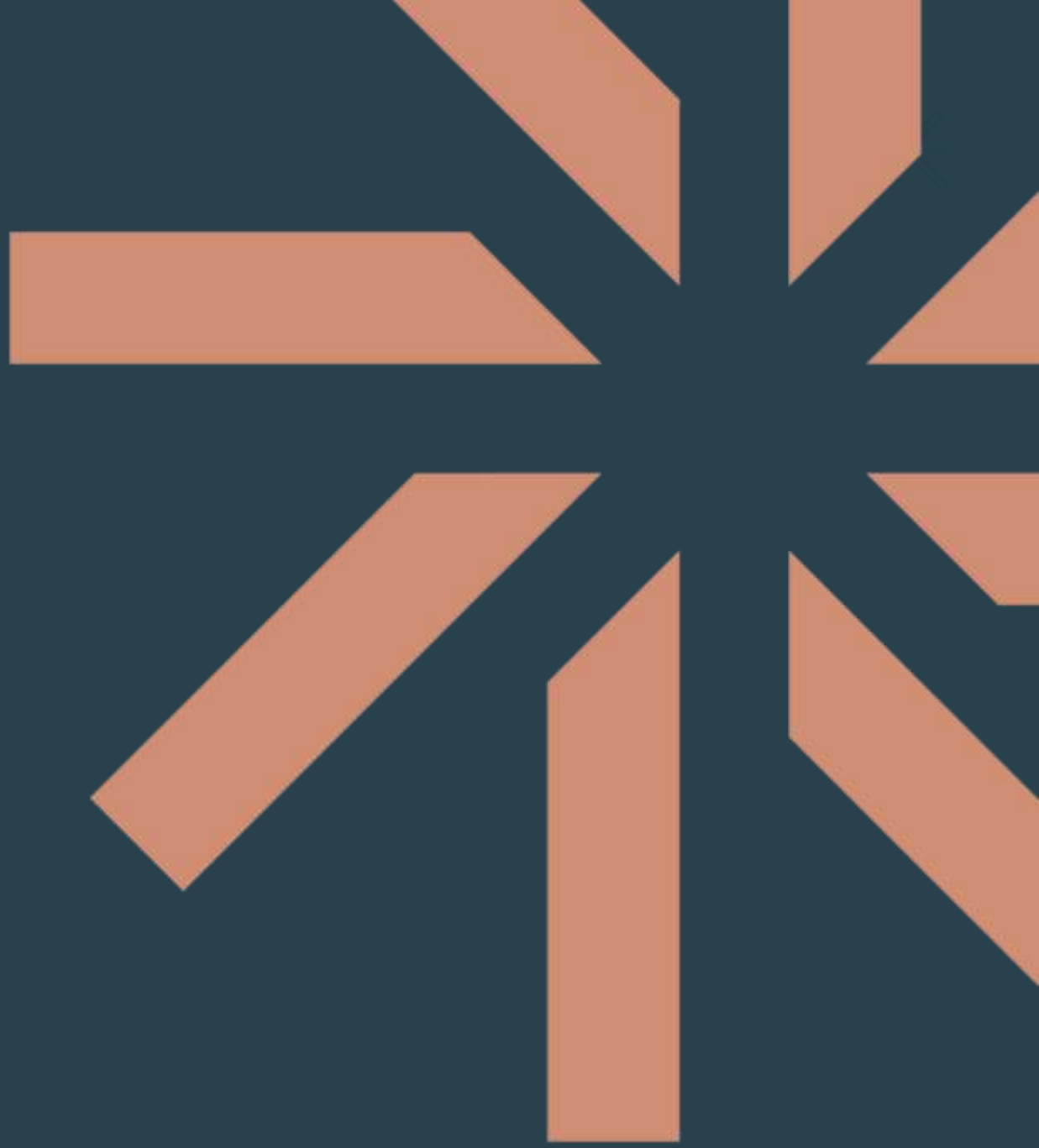
Modules & Dependency Injection

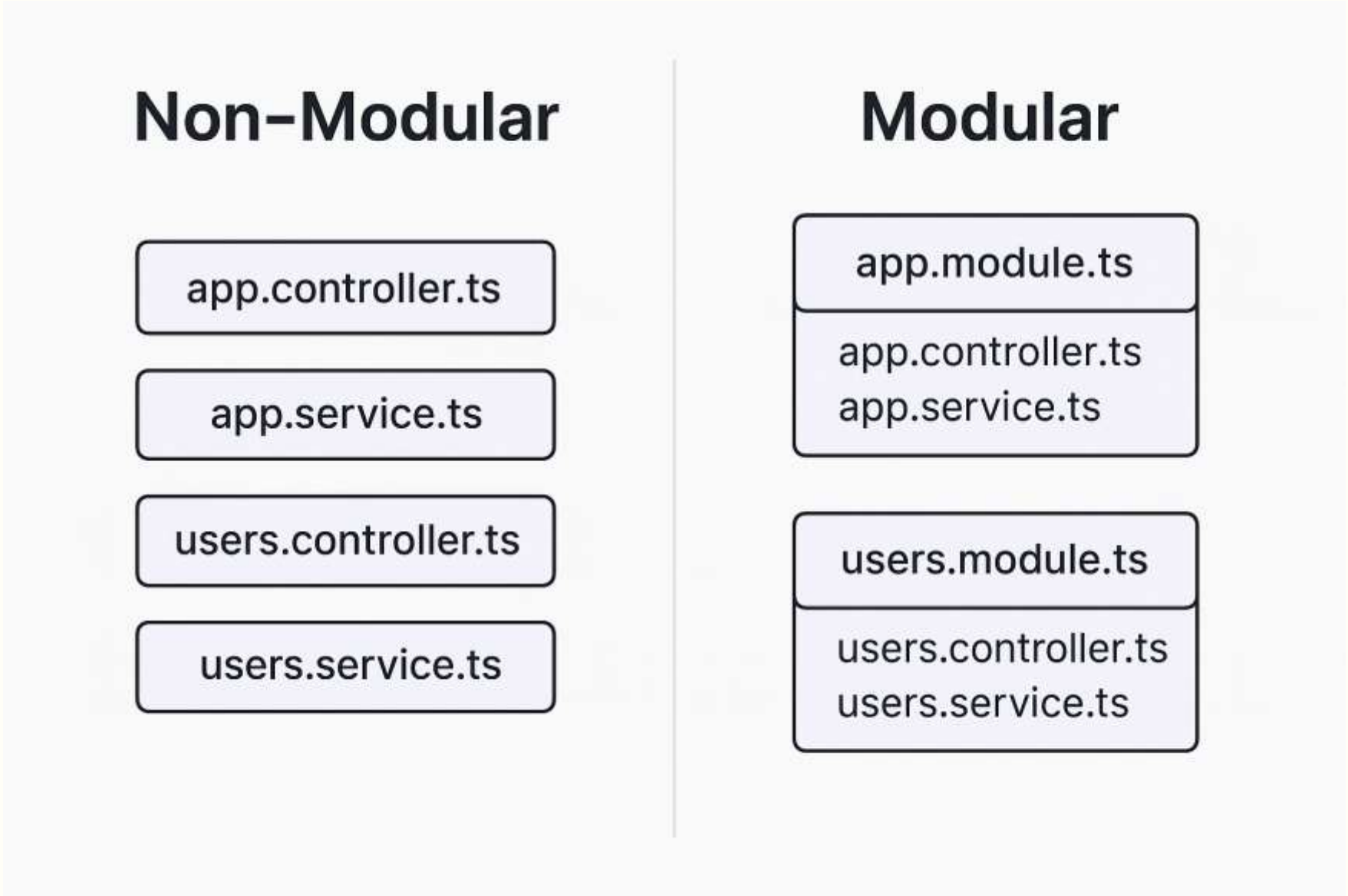
Qinshift Academy

What is a Module in NestJS?

A module in NestJS is a class decorated with @Module() that organizes related components like controllers, services, and providers into a cohesive unit. It helps structure the application into feature-based or domain-based sections, enabling better modularity, scalability, and maintainability.



Non-Modular VS Modular structure



Non-Modular Structure

Files are flat and ungrouped.

Example files:

- app.controller.ts
- app.service.ts
- users.controller.ts
- users.service.ts

Issue: No clear separation of features/domains. Harder to maintain and scale.

Modular Structure

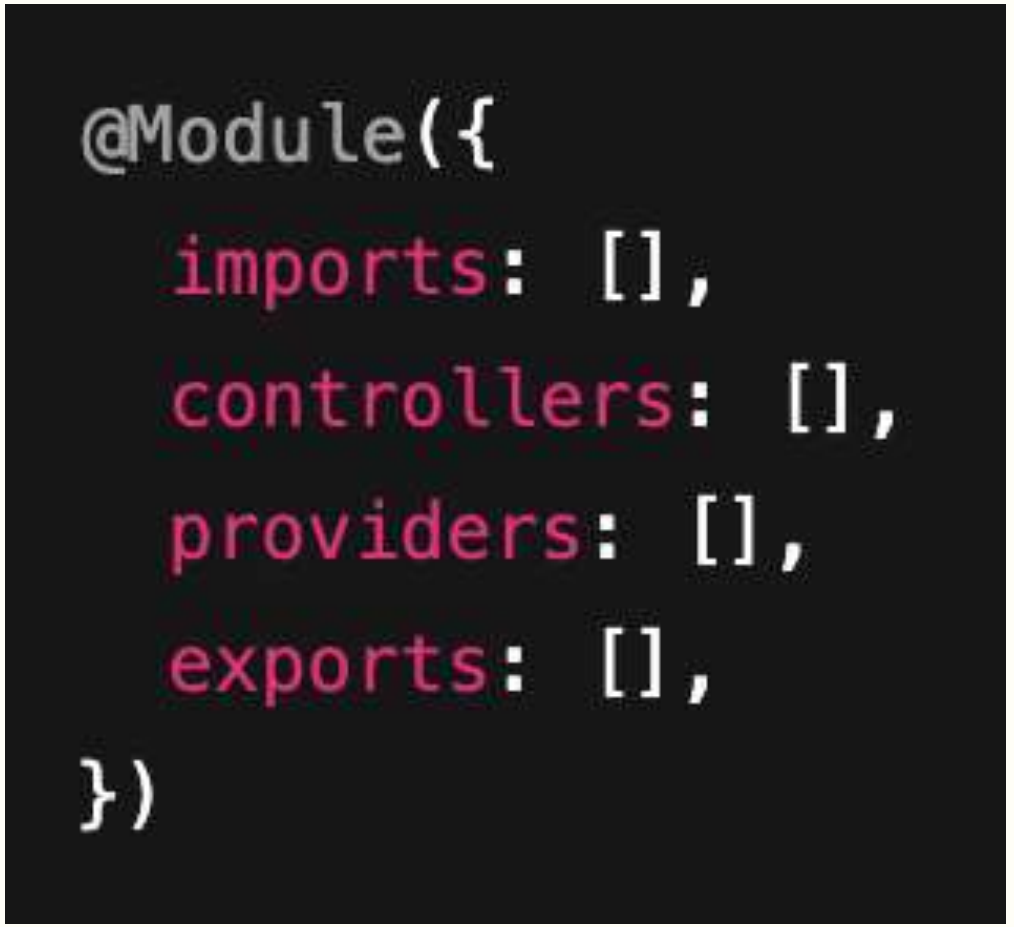
Files are grouped into modules based on features.

Example:

- AppModule contains app.controller.ts and app.service.ts
- UsersModule contains users.controller.ts, users.service.ts, and a users.module.ts

Benefit: Each module encapsulates its own logic, making the application more maintainable, testable, and scalable.

@Module() Decorator



Imports

- Used to bring in other modules that this module depends on.
- Enables module composition.

Imports

- Used to bring in other modules that this module depends on.
- Enables module composition.

```
imports: [UsersModule, AuthModule]
```

Controllers

- Lists all controllers managed by this module.
- These controllers define the routes and handle requests.

Providers

- Registers services, factories, or other providers that contain business logic or dependencies.
- These are the parts that can be injected via Dependency Injection.

Exports

- Specifies which providers (like services) should be made available to other modules that import this one.
- Without exporting, a service is only available internally.

```
exports: [UsersService]
```

Shared Module

- A common module that exports reusable services, pipes, guards, etc., used in many other modules.
- Should not register global providers (unless you want them shared across the entire app).

Feature Module

- A module dedicated to a specific domain or feature, like UsersModule, AuthModule, ProductsModule.

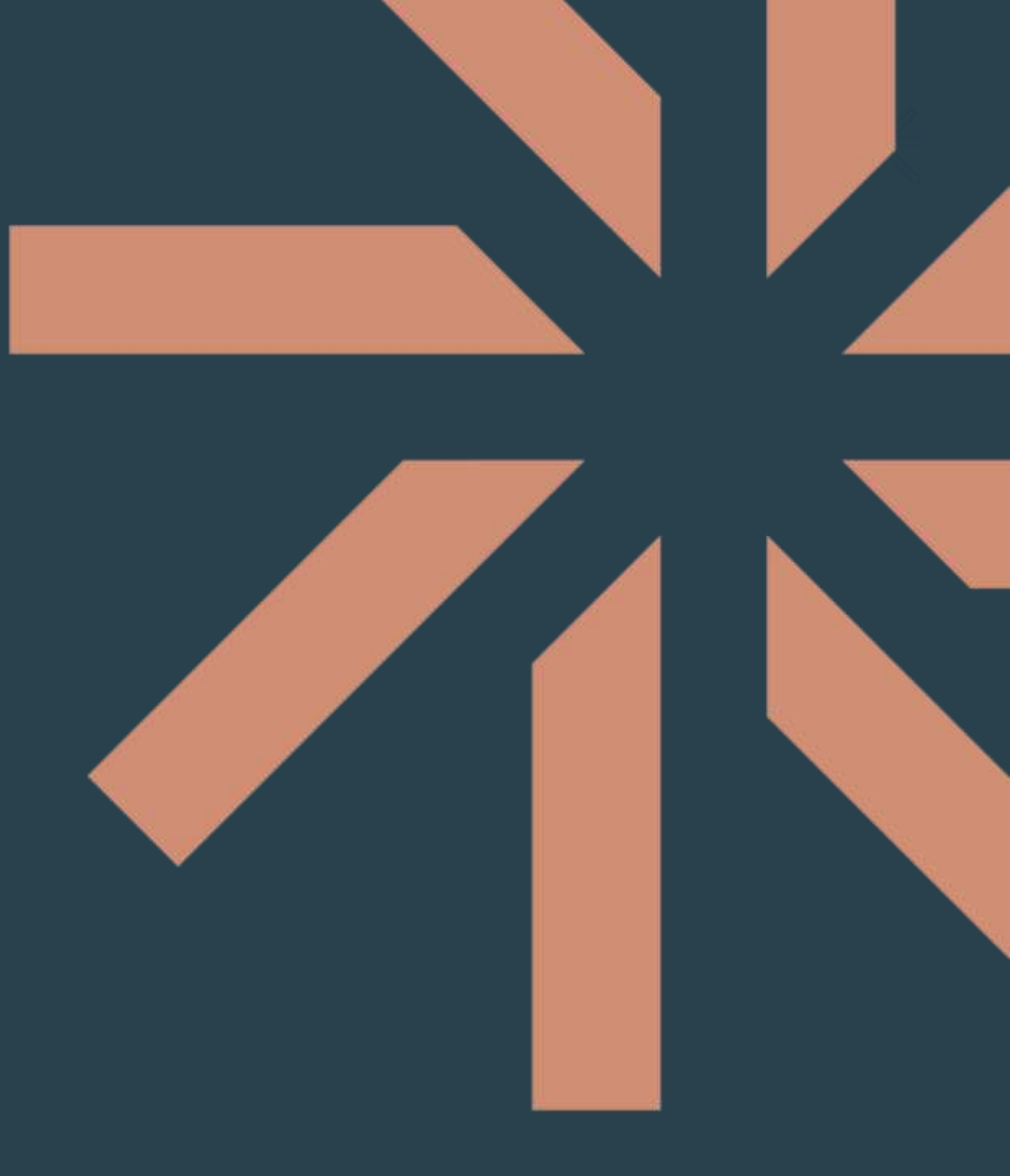
Global Module

- A module marked with @Global() makes its providers available application-wide without importing it manually elsewhere.

```
@Global()
@Module({ ... })
export class ConfigModule {}
```

What is Dependency Injection?

Dependency Injection (DI) is a design pattern where classes receive their dependencies (like services or repositories) from the framework, instead of creating them manually. It promotes loose coupling, easier testing, and better code organization by letting NestJS automatically manage and inject required instances where needed.



Why Use DI in NestJS?

- ✓ Promotes loose coupling
- ✓ Makes code more testable
- ✓ Encourages single responsibility
- ✓ Simplifies dependency management

- NestJS has a built-in DI container that automatically handles class dependencies and their lifecycles.

Provider

A provider is any class or value that can be injected as a dependency.

```
@Injectable()
export class UsersService {
  getUsers() {
    return ['Alice', 'Bob'];
  }
}
```

Marked with @Injectable() so Nest knows it can be injected.

Injecting Dependencies

You can inject providers using the constructor of a class:

```
@Controller('users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

  @Get()
  findAll() {
    return this.usersService.getUsers();
  }
}
```

Nest automatically creates an instance of UsersService and provides it to UsersController.

Questions?

Trainer Name

Trainer

trainer@mail.com

Assistant Name

Assistant

asistant@mail.com

Qinshift Academy