NestJS — The Enterprise-Ready Node.js Framework

NestJS (Nest) is a progressive Node.js framework for building efficient, reliable, and scalable server-side applications. Built with TypeScript, it uses Express (or Fastify) under the hood but adds a powerful architectural layer inspired by Angular.

Core Concepts

1. Modules
Every application has at least one module — the root module. Modules encapsulate providers, controllers, and imports.

```ts
@Module({
  imports: [UsersModule, AuthModule],
  controllers: [AppController],
  providers: [AppService, PrismaService],
})
export class AppModule {}
```

2. Controllers & Routing

Controllers handle incoming HTTP requests and return responses.

```ts
@Controller('users')
export class UsersController {
  @Get(':id')
  findOne(@Param('id') id: string) {
    return this.usersService.findOne(+id);
  }

  @Post()
  @UseGuards(JwtAuthGuard)
  create(@Body() createUserDto: CreateUserDto) {
    return this.usersService.create(createUserDto);
  }
}
```

3. Providers & Dependency Injection

Services, repositories, helpers — all injectable.

```ts
@Injectable()
export class UsersService {
  constructor(private prisma: PrismaService) {}
  async findOne(id: number) {
    return this.prisma.user.findUnique({ where: { id } });
  }
}
```

4. Middleware, Guards, Interceptors, Pipes

- **Guards** → Authorization (CanActivate)
- **Interceptors** → Transform responses, logging, caching
- **Pipes** → Validation & transformation (class-validator + class-transformer)
- **Exception Filters** → Centralized error handling

5. Microservices & Hybrid Apps

Nest supports TCP, Redis, MQTT, gRPC, and Kafka out of the box.

```ts
// microservice.ts
async function bootstrap() {
  const app = await NestFactory.createMicroservice(AppModule, {
    transport: Transport.REDIS,
    options: { host: 'localhost', port: 6379 },
  });
  await app.listen();
}
```

6. GraphQL, WebSockets, and More

Built-in support for:

- GraphQL (code-first or schema-first)
- WebSockets (Gateways)
- CQRS & Event Sourcing
- OpenAPI (Swagger) auto-generation

Why Top Companies Choose NestJS in 2025

- TypeScript-first → fewer runtime errors
- Angular-style architecture → easy for large teams
- Modular & testable by design
- Excellent for monorepos and microservices
- Huge ecosystem (Prisma, TypeORM, Mongoose, etc.)

NestJS is no longer "just another framework" — it's the **standard for enterprise Node.js backends** in 2025.