```
# A Nova Esperança do Backend (NestJS)

## □ Introdução

NestJS é um framework progressivo para construir aplicações Node.js com arquitetura modular, tipagem forte e suporte nativo ao TypeS

Este ebook resume as principais funcionalidades do NestJS de forma simples, prática e com exemplos reais.

---

## □ Página 1 — "O Caminho Jedi do NestJS"

### Arquitetura Modular

A base do NestJS é a modularidade. Cada módulo agrupa funcionalidades e facilita a manutenção. Exemplo real: módulo de usuários.

```typescript
// users.module.ts
import { Module } from '@nestjs/common';
import { UsersService } from './users.service';
import { UsersController } from './users.controller';

@Module({
  controllers: [UsersController],
  providers: [UsersService],
})
export class UsersModule {}
```

□ Página 2 — "Força e Tipagem"

Injeção de Dependências (DI)

Com DI, o NestJS cria e gerencia objetos automaticamente.

```
// users.service.ts
@Injectable()
export class UsersService {
  findAll() {
    return [{ id: 1, name: 'Leia Organa' }];
  }
}

// users.controller.ts
@Controller('users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

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

□ Página 3 — "Rotas no Hiperspaço"

Controllers e Rotas

Controllers definem as rotas da aplicação.
```

```
@Controller('planets')
export class PlanetsController {
  @Get('tatooine')
  getPlanet() {
    return { name: 'Tatooine', climate: 'arid' };
  }
}
```

Página 4 — "Domine a Força das Interfaces"

DTOs e Validação

Os DTOs garantem entradas seguras e validadas.

```
// create-user.dto.ts
export class CreateUserDto {
  @IsString()
  name: string;
}

@Post()
create(@Body() dto: CreateUserDto) {
  return { message: `Usuário ${dto.name} criado` };
}
```

Página 5 — "Ordem dos Dados"

Integração com Banco de Dados

NestJS funciona bem com ORMs como TypeORM e Prisma.

```
// user.entity.ts
@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;
}
```

Página 6 — "Guardiões da Galáxia API"

Middlewares

Ideais para autenticação, logs e regras globais.

```
export function Logger(req, res, next) {
  console.log(`Request → ${req.method} ${req.url}`);
  next();
}
```

Página 7 — "O Lado Seguro da Força"

Guards (Autorização)

Guards controlam acesso às rotas.

```
@Injectable()
export class AdminGuard implements CanActivate {
  canActivate(context: ExecutionContext) {
    const req = context.switchToHttp().getRequest();
    return req.headers['x-admin'] === 'true';
  }
}

@UseGuards(AdminGuard)
@Get('secure')
secureData() {
  return 'Área restrita para Jedis';
}
```

Página 8 — "Aprendendo com o Conselho Jedi"

Pipes (Transformação e Validação)

Pipes ajustam os dados antes de chegar ao controller.

```
@Pipe()
export class ParseIntPipe implements PipeTransform {
  transform(value: string) {
    const val = Number(value);
    if (isNaN(val)) throw new BadRequestException('ID inválido');
    return val;
  }
}
```

Página 9 — "Mensageiros da República"

Interceptors

Perfeitos para logs, manipular respostas e métricas.

```
@Injectable()
export class LogInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler) {
    console.log('Antes da resposta...');
    return next.handle().pipe(
      tap(() => console.log('Depois da resposta...')),
    );
  }
}
```

Página 10 — "Expanda seu Império com Microservices"

Microservices e Mensageria

NestJS suporta Redis, MQTT, Kafka e mais.

```
// microservice.ts
NestFactory.createMicroservice(AppModule, {
  transport: Transport.TCP,
});
```

Conclusão

Você agora domina as principais forças do NestJS. Continue treinando como um verdadeiro Jedi do backend e expanda suas habilidades construindo APIs escaláveis, seguras e bem estruturadas.

Que a Força esteja com você! ```