NestJS is a **progressive Node.js framework** for building **efficient, scalable, and maintainable server-side applications**. It's built with **TypeScript** (but supports JavaScript too) and heavily inspired by **Angular**, making it particularly attractive for developers coming from a frontend Angular background.

□ Key Features of NestJS

1. **TypeScript First**
   - Written in TypeScript and supports full OOP, FP, and FRP paradigms.
2. **Modular Architecture**
   - Code is organized into **modules**, making it easier to scale and maintain.
3. **Dependency Injection**
   - Built-in DI system (inspired by Angular), promotes clean and testable code.
4. **Support for Express and Fastify**
   - By default, Nest uses **Express.js**, but you can switch to **Fastify** for better performance.
5. **Built-in Support for WebSockets, GraphQL, Microservices, and REST APIs**
6. **CLI Tooling**
   - The Nest CLI helps with generating code, running tests, and building your project efficiently.
7. **Extensive Decorator Usage**
   - Uses decorators like `@Module()`, `@Controller()`, `@Injectable()`, etc., to define structure.
8. **Middleware, Pipes, Guards, Interceptors**
   - Support for advanced patterns to handle requests and responses in a clean way.

□ Example Structure

```
src/
├── app.controller.ts
├── app.module.ts
├── app.service.ts
main.ts
// app.controller.ts
@Controller()
export class AppController {
  @Get()
  getHello(): string {
    return 'Hello World!';
  }
}
```

□ Use Cases

- RESTful APIs
- GraphQL APIs
- Microservices architecture
- Real-time applications (with WebSockets)
- Monolithic or distributed backend systems

□ Summary

Feature	Description
Language	TypeScript
Framework Base	Built on top of Express/Fastify
Paradigm	Modular, Decorator-based, DI
Inspired by	Angular
Ideal For	Scalable, testable server-side apps

Sure! Here's A **Quick And Practical Example** Of How To Build A **5-6 Page Website** (Multi-Page Restful Website) Using **Nestjs**.

□ Prerequisites

- Node.js installed
- NestJS CLI installed:

```
npm i -g @nestjs/cli
```

```
D:\>mkdir nestjs

D:\>cd nestjs

D:\nestjs>npm i -g @nestjs/cli

added 245 packages in 1m

46 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New minor version of npm available! 11.3.0 -> 11
npm notice Changelog: https://github.com/npm/cli/releases/t
npm notice To update run: npm install -g npm@11.5.2
npm notice
```

□ Step-by-Step Setup

---

1. Create a NestJS Project

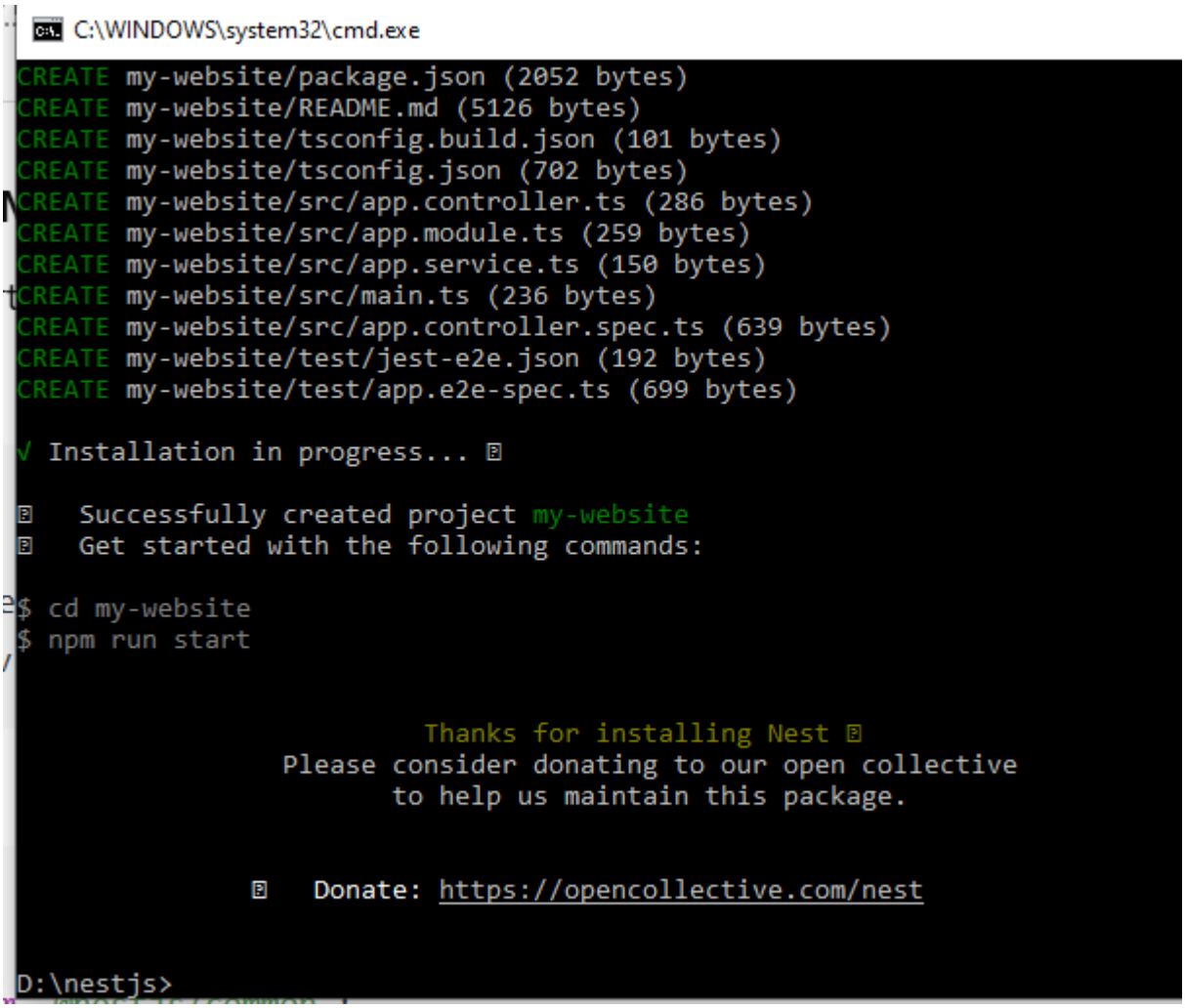
```
nest new my-website
cd my-website
```

```
D:\nestjs>
D:\nestjs>nest new my-website
□  We will scaffold your app in a few seconds..

√ Which package manager would you □□  to use? npm
CREATE my-website/.prettierrc (54 bytes)
CREATE my-website/eslint.config.mjs (869 bytes)
CREATE my-website/nest-cli.json (179 bytes)
CREATE my-website/package.json (2052 bytes)
CREATE my-website/README.md (5126 bytes)
CREATE my-website/tsconfig.build.json (101 bytes)
CREATE my-website/tsconfig.json (702 bytes)
CREATE my-website/src/app.controller.ts (286 bytes)
CREATE my-website/src/app.module.ts (259 bytes)
CREATE my-website/src/app.service.ts (150 bytes)
CREATE my-website/src/main.ts (236 bytes)
CREATE my-website/src/app.controller.spec.ts (639 bytes)
CREATE my-website/test/jest-e2e.json (192 bytes)
CREATE my-website/test/app.e2e-spec.ts (699 bytes)

▸□□□□ Installation in progress... □
```

Then

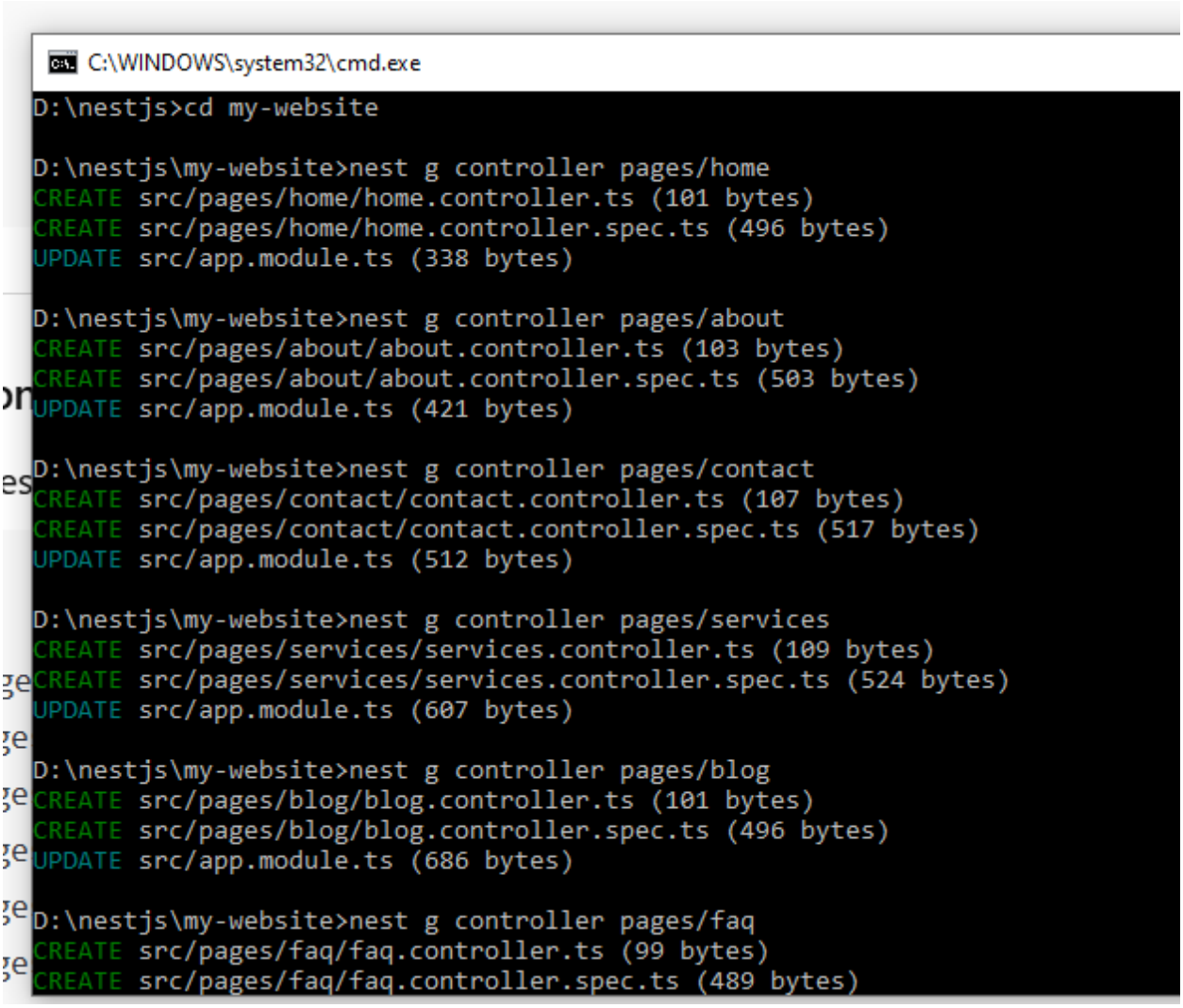


---

2. Generate Pages (Controllers + Services)

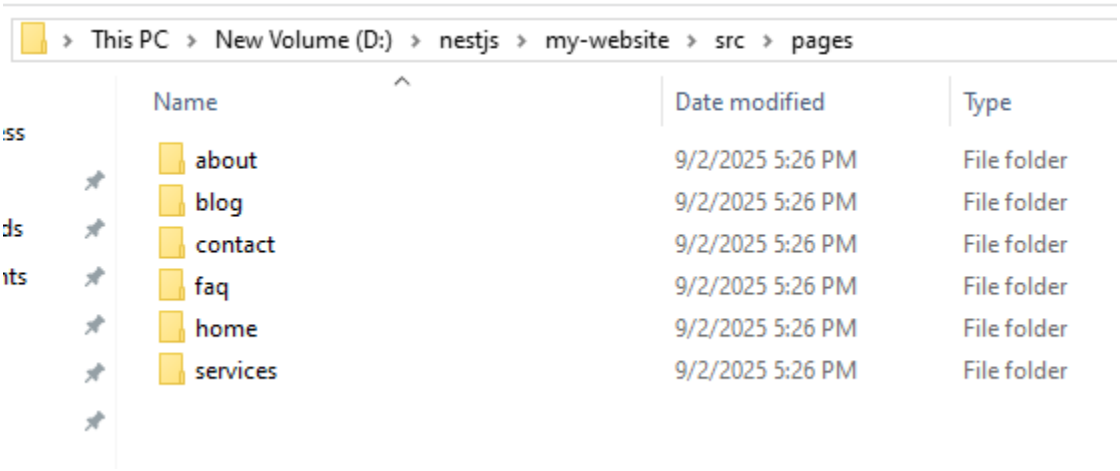
Let's say you want 6 pages: **Home**, **About**, **Contact**, **Services**, **Blog**, and **FAQ**.

```
nest g controller pages/home
nest g controller pages/about
nest g controller pages/contact
nest g controller pages/services
nest g controller pages/blog
nest g controller pages/faq
```



(You don't need services unless doing logic, so we skip `nest g service`.)

After it open your project folder you will see like this :-



---

3. Define Routes in Controllers

Home page: `pages/home/home.controller.ts`

```
import { Controller, Get } from '@nestjs/common';

@Controller()
export class HomeController {
  @Get()
  getHome(): string {
    return 'Welcome to the Home Page!';
  }
}
```

About page: `pages/about/about.controller.ts`

```
import { Controller, Get } from '@nestjs/common';

@Controller('about')
export class AboutController {
  @Get()
  getAbout(): string {
    return 'About Us Page';
  }
}
```

📁 Folder Structure:-

Assuming we're placing all these under:

```
src/
└── pages/
    ├── contact/
    │   └── contact.controller.ts
    ├── services/
    │   └── services.controller.ts
    ├── blog/
    │   └── blog.controller.ts
    └── faq/
        └── faq.controller.ts
```

📄 1 contact page. `pages/contact/contact.controller.ts`

```
import { Controller, Get } from '@nestjs/common';

@Controller('contact')
export class ContactController {
  @Get()
  getContact(): string {
    return 'This is the Contact Page.';
  }
}
```

📄 2 services page. `pages/services/services.controller.ts`

```
import { Controller, Get } from '@nestjs/common';

@Controller('services')
export class ServicesController {
  @Get()
  getServices(): string {
    return 'This is the Services Page.';
  }
}
```

📄 3 blog page. `pages/blog/blog.controller.ts`

```
import { Controller, Get } from '@nestjs/common';

@Controller('blog')
export class BlogController {
  @Get()
  getBlog(): string {
    return 'This is the Blog Page.';
  }
}
```

□ 4 faq page. `pages/faq/faq.controller.ts`

```
import { Controller, Get } from '@nestjs/common';

@Controller('faq')
export class FaqController {
  @Get()
  getFaq(): string {
    return 'This is the FAQ Page.';
  }
}
```

Add Controllers to `src/app.module.ts:-`

Edit `src/app.module.ts` and register the controllers:-

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

import { AppController } from './app.controller';

import { AppService } from './app.service';

import { HomeController } from './pages/home/home.controller';

import { AboutController } from './pages/about/about.controller';

import { ContactController } from './pages/contact/contact.controller';

import { ServicesController } from './pages/services/services.controller';

import { BlogController } from './pages/blog/blog.controller';

import { FaqController } from './pages/faq/faq.controller';

```
@Module({

  imports: [],

  controllers: [HomeController, AboutController, ContactController, ServicesController, BlogController, FaqController],

  providers: [AppService],

})

export class AppModule {}
```

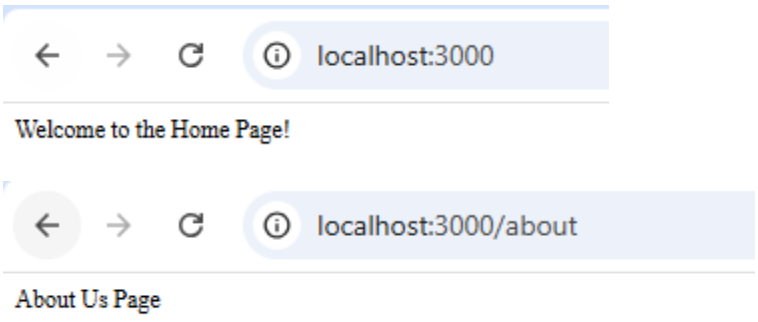
Run Your App

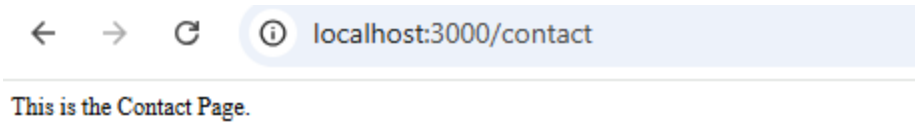
```
npm run start
```

Open in browser:

- http://localhost:3000/contact
- http://localhost:3000/services
- http://localhost:3000/blog
- http://localhost:3000/faq

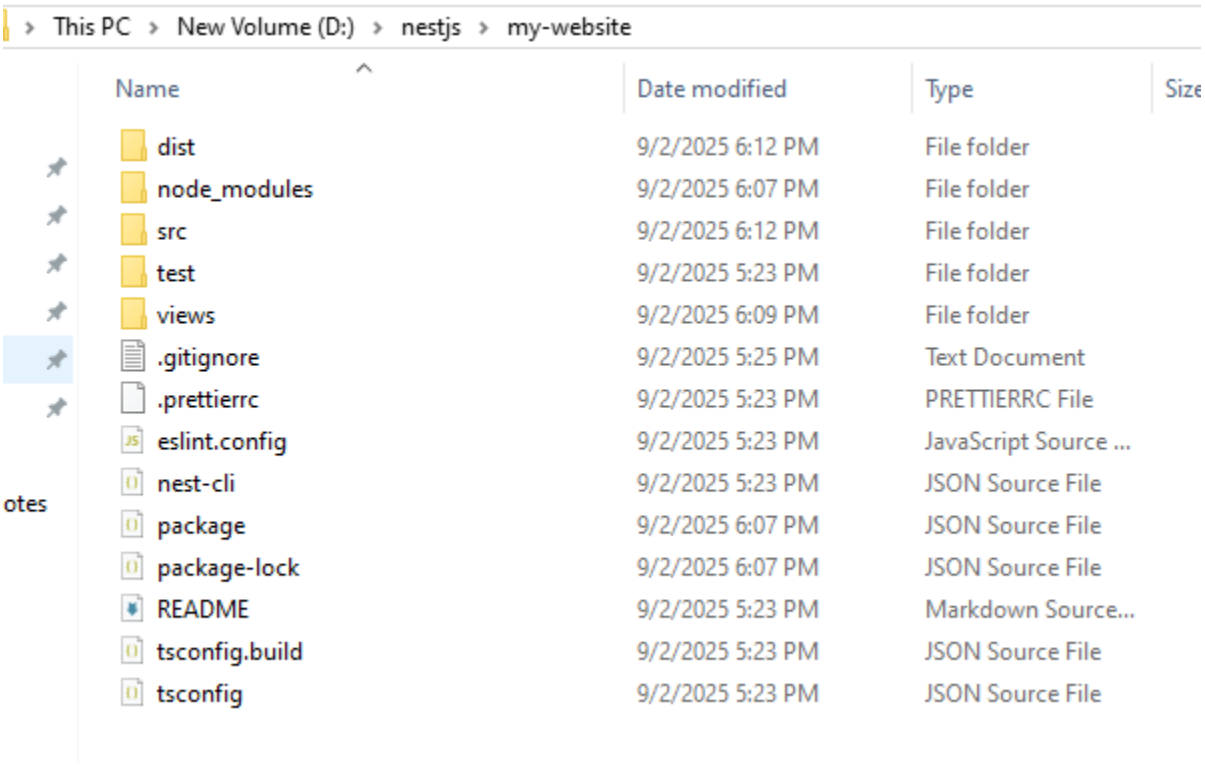
```
output :-
```





This is the Contact Page.

Now to display content with ejs format :-



Using **EJS** with NestJS allows you to serve **dynamic HTML pages** — perfect for building a real website with templates.

□ Goal

- Set up **EJS** as the view engine.

- Render a dynamic **Home Page** at `/`.
- Use EJS for future pages like `/contact`, `/services`, etc.

□ Step-by-Step: Setup EJS with NestJS

□ 1. Install EJS

In your NestJS project:

```
npm install ejs
```

□ 2. Enable View Engine in `main.ts`

Open `src/main.ts` and modify it:

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { NestExpressApplication } from '@nestjs/platform-express';
import { join } from 'path';

async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(AppModule);

  // Set EJS as the view engine
  app.setBaseViewsDir(join(__dirname, '..', 'views'));
  app.setViewEngine('ejs');

  await app.listen(3000);
}
bootstrap();
```

□ 3. Create `my-website/views` Folder & `home.ejs` File

In your root directory, create:

```
/views/
└── home.ejs
```

Add some HTML content to `home.ejs`:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Home</title>
</head>
<body>
  <h1>Welcome to the Home Page</h1>
  <p>This page is rendered using <strong>EJS</strong>.</p>
</body>
</html>
```

□ 4. Update `home.controller.ts` to Render the View

```
import { Controller, Get, Render } from '@nestjs/common';

@Controller()
export class HomeController {
  @Get()
  @Render('home') // Looks for views/home.ejs
  getHome() {
    return {}; // Pass data here if needed
  }
}
```

□ 5. Run the Server

```
npm run start
```

□ 6. Open in Browser

Go to:

```
http://localhost:3000/
```

Optional: Pass Data to EJS Template

You can pass dynamic data like this pages/home/home.controller.ts :

```
import { Controller, Get, Render } from '@nestjs/common';


@Controller()

export class HomeController {

  @Get()

  @Render('home') // Looks for views/home.ejs

  getHome() {

    return {title: 'My NestJS Website'}; // Pass data here if needed

  }

}
```

Update my-website/views/`home.ejs` to use the data:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Home</title>
</head>
<body>
  <h1><%= title %></h1>

  <p>This page is rendered using <strong>EJS</strong>.</p>
</body>
</html>
```