Model-View-Controller

Nest, by default, makes use of **express** library under the hood. Hence, every tutorial about MVC (Model-View-Controller) pattern in express concerns Nest as well. Firstly, let's scaffold a simple Nest application using **CLI** tool:

```
$ npm i -g @nestjs/cli
$ nest new project
```

In order to create a MVC app, we have to install a **template engine**:

```
$ npm install --save hbs
```

We decided to use a `hbs` engine, though, you can use whatever fits your requirements. Once the installation process is completed, we need to configure the express instance using following code:

main.ts JS

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

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

  app.useStaticAssets(join(__dirname, '..', 'public'));
  app.setBaseViewsDir(join(__dirname, '..', 'views'));
  app.setViewEngine('hbs');

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

We told **express** that the `public` directory will be used for storing static assets, `views` will contain templates, and a

`hbs` template engine should be used to render an HTML output.

Now, let's create a `views` directory and `index.hbs` template inside it. In the template, we are gonna print a `message` passed from the controller:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>App</title>
  </head>
  <body>
    {{ message }}
  </body>
</html>
```

Afterward, open the `app.controller` file and replace the `root()` method with the following code:

app.controller.ts JS

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

@Controller()
export class AppController {
  @Get()
  @Render('index')
  root() {
    return { message: 'Hello world!' };
  }
}
```

> **HINT**
> In fact, when Nest detects `@Res()` decorator, it injects library-specific `response` object. We can use such an object to dynamically render the template. Learn more about its abilities **here**.

While the application is running, open your browser and navigate to `http://localhost:3000/`. You should see the `Hello world!` message.

Dynamic template rendering

If the application logic must dynamically decide which template to render, then we should use `@Res()` decorator:

app.controller.ts JS

```
import { Get, Controller, Render } from '@nestjs/common';
import { Response } from 'express';
import { AppService } from './app.service';

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  root(@Res() res: Response) {
    return res.render(
      this.appService.getViewName(),
      { message: 'Hello world!' },
    );
  }
}
```

Example

A working example is available **here**.

Fastify

As mentioned in this **chapter**, we are able to use any compatible HTTP provider together with Nest. One of them is a **fastify** library. In order to create a MVC application with fastify, we have to install following packages:

```
$ npm i --save fastify point-of-view handlebars
```

The next steps cover almost the same stuff as in case of express library (with small differences). Once the installation process is completed, we need to open `main.ts` file and update its contents:

main.ts JS

```
import { NestFactory } from '@nestjs/core';
import { NestFastifyApplication, FastifyAdapter } from '@nestjs/platform-fastify';
import { ApplicationModule } from './app.module';
import { join } from 'path';
```

```
async function bootstrap() {
  const app = await NestFactory.create<NestFastifyApplication>(
    ApplicationModule,
    new FastifyAdapter(),
  );
  app.useStaticAssets({
    root: join(__dirname, '..', 'public'),
    prefix: '/public/',
  });
  app.setViewEngine({
    engine: {
      handlebars: require('handlebars'),
    },
    templates: join(__dirname, '..', 'views'),
  });
  await app.listen(3000);
}
bootstrap();
```

The API is different a little but the idea that sits behind those methods calls remains the same. Also, we have to ensure that the template name passed into the `@Render()` decorators include a file extension.

app.controller.ts JS

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

@Controller()
export class AppController {
  @Get()
  @Render('index.hbs')
  root() {
    return { message: 'Hello world!' };
  }
}
```

While the application is running, open your browser and navigate to `http://localhost:3000/`. You should see the `Hello world!` message.

Example

A working example is available **here**.

Support us

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to join them, please read more here.

Principal Sponsor

Sponsors / Partners