

Tóm tắt:

Giới thiệu NestJS	2
1. Các thành phần cơ bản của NestJS	2
1.1. Modules	2
1.2. Controllers	3
1.3. Services.....	3
1.4. Dependency Injection	4
2. Các công nghệ chính được cung cấp sẵn trong NestJS	4
2.1. GraphQL.....	4
2.2. WebSockets	4
2.3. Microservices.....	4
2.4. Validation	4
2.5. Authentication	4
2.6. Swagger	5
3. Triển khai giải pháp theo chuẩn MVC trong NestJS.....	5
3.1. Mô hình MVC trong NestJS	5
Cài đặt thực tế	7
1. Cài đặt môi trường.....	7
2. Generate User Module, Controller và Service	8
3. Cấu hình MySQL với TypeORM.....	9
4. Cài đặt Swagger trong NestJS để kiểm tra API.....	10
Kết luận.....	12

Giới thiệu NestJS

NestJS là một framework dành cho backend được xây dựng trên nền tảng Node.js và sử dụng TypeScript. Framework này lấy cảm hứng từ Angular và giúp xây dựng các ứng dụng server-side dễ dàng và mạnh mẽ, hỗ trợ nhiều kỹ thuật và mô hình kiến trúc hiện đại.

1. Các thành phần cơ bản của NestJS

NestJS sử dụng một cấu trúc ứng dụng modular (module-based), với các thành phần chính sau:

1.1. Modules

- Module trong NestJS giúp nhóm các thành phần lại với nhau, như controller và service.
- Mỗi ứng dụng NestJS ít nhất phải có một AppModule để khởi tạo ứng dụng.
- Các module có thể import vào nhau, tạo thành các phần độc lập.

Ví dụ về Module:

```
// app.module.ts

import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  imports: [], // Import các module khác
  controllers: [AppController], // Các controller
  providers: [AppService], // Các service
})
export class AppModule { }
```

1.2. Controllers

- Controller trong NestJS xử lý các HTTP request (GET, POST, PUT, DELETE).
- Mỗi route handler trong controller sẽ nhận request và trả về response.

Ví dụ về một controller:

```
// app.controller.ts
import { Controller, Get } from '@nestjs/common';

@Controller() // Route chính của controller này
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get() // Xử lý GET request tại route '/'
  getHello(): string {
    return this.appService.getHello();
  }
}
```

1.3. Services

- Service là nơi chứa các logic nghiệp vụ trong ứng dụng.
- Controllers gọi các services để xử lý dữ liệu hoặc các yêu cầu phức tạp.

Ví dụ về một service:

```
// app.service.ts
import { Injectable } from '@nestjs/common';

@Injectable() // Đánh dấu đây là một service có thể inject vào controller
```

```
export class AppService {  
  getHello(): string {  
    return 'Hello World!';  
  }  
}
```

1.4. Dependency Injection

- NestJS sử dụng Dependency Injection (DI) để dễ dàng quản lý các phụ thuộc giữa các thành phần của ứng dụng, giúp tái sử dụng code dễ dàng và đơn giản hơn.

2. Các công nghệ chính được cung cấp sẵn trong NestJS

NestJS cung cấp một số công nghệ và tính năng mạnh mẽ:

2.1. GraphQL

- NestJS hỗ trợ GraphQL để xây dựng các API có khả năng truy vấn dữ liệu linh hoạt.

2.2. WebSockets

- WebSockets được tích hợp sẵn, giúp bạn xây dựng các ứng dụng thời gian thực như chat, thông báo, v.v.

2.3. Microservices

- NestJS hỗ trợ Microservices, giúp xây dựng các ứng dụng phân tán dễ dàng.

2.4. Validation

- Validation được tích hợp sẵn với thư viện class-validator và class-transformer để kiểm tra và chuyển đổi dữ liệu đầu vào.

2.5. Authentication

- NestJS hỗ trợ việc xác thực và phân quyền thông qua các chiến lược như JWT, OAuth, v.v.

2.6. Swagger

- Swagger được tích hợp để tự động sinh tài liệu API, giúp bạn dễ dàng xây dựng và kiểm tra API.

3. Triển khai giải pháp theo chuẩn MVC trong NestJS

3.1. Mô hình MVC trong NestJS

- M (Model): Định nghĩa cấu trúc dữ liệu (thường dùng Entity hoặc DTO).
- V (View): Trong NestJS, phần này thường không được sử dụng nhiều (NestJS chủ yếu là backend, không liên quan đến giao diện người dùng).
- C (Controller): Xử lý các HTTP request và trả về response.
- S (Service): Xử lý các logic nghiệp vụ và tương tác với cơ sở dữ liệu.

Ví dụ về cách triển khai theo mô hình MVC:

Bước 1: Tạo một module mới cho User

```
nest generate module users  
nest generate controller users  
nest generate service users
```

Bước 2: Cấu trúc UserController và UserService

UserController (Xử lý các route liên quan đến người dùng)

```
// src/users/users.controller.ts  
import { Controller, Post, Body } from '@nestjs/common';  
import { UserService } from './users.service';  
import { CreateUserDto } from './create-user.dto';  
  
@Controller('users')  
export class UsersController {  
  constructor(private readonly userService: UserService) {}  
}
```

```
@Post() // Route để tạo người dùng
create(@Body() createUserDto: CreateUserDto) {
  return this.userService.create(createUserDto);
}
}
```

UserService (Chứa logic nghiệp vụ liên quan đến người dùng)

```
// src/users/users.service.ts
import { Injectable } from '@nestjs/common';
import { CreateUserDto } from './create-user.dto';

@Injectable()
export class UsersService {
  private users = [];

  create(createUserDto: CreateUserDto) {
    this.users.push(createUserDto);
    return 'User created!';
  }
}
```

CreateUserDto (Định nghĩa cấu trúc dữ liệu khi tạo người dùng)

```
// src/users/create-user.dto.ts
export class CreateUserDto {
  readonly username: string;
```

```
readonly email: string;
readonly password: string;
}
```

Bước 3: Cập nhật AppModule

```
// src/app.module.ts
import { Module } from '@nestjs/common';
import { UsersModule } from '../users/users.module';

@Module({
  imports: [UsersModule], // Import UsersModule vào AppModule
})
export class AppModule {}
```

Cài đặt thực tế

1. Cài đặt môi trường

Bước 1: Cài đặt Node.js và NestJS CLI

- Đầu tiên, Cài đặt **Node.js** (nếu chưa có). Bạn có thể tải Node.js tại nodejs.org và cài đặt theo hướng dẫn.
- Sau khi cài đặt Node.js, cài đặt **NestJS CLI** toàn cục:

```
PS D:\Test\AASCTest2> npm install -g @nestjs/cli
added 248 packages in 26s
45 packages are looking for funding
run 'npm fund' for details
```

Bước 2: Tạo dự án mới

- Sử dụng NestJS CLI để tạo một dự án mới cho ứng dụng:
- Mở cmd ở nơi muốn đặt ứng dụng:

```
PS D:\Test\AASCTest2> nest new full-solution-test2
✨ We will scaffold your app in a few seconds..

? Which package manager would you ❤️ to use? (Use arrow keys)
> npm
  yarn
  pnpm
```

Chọn npm

- Cd đến thư mục ứng dụng

Bước 3: Cài đặt các package phụ thuộc

- Để triển khai server game online ở bài 3, cần cài đặt các thư viện bổ sung như **TypeORM** (cho cơ sở dữ liệu) và **Passport** (cho xác thực người dùng).
- Cài đặt TypeORM và MySQL driver:

```
PS D:\Test\AASCTest2\full-solution-test2> npm install @nestjs/typeorm typeorm mysql2
added 28 packages, and audited 850 packages in 10s

160 packages are looking for funding
  run 'npm fund' for details
```

- Cài đặt Passport và các chiến lược xác thực:

```
PS D:\Test\AASCTest2\full-solution-test2> npm install @nestjs/passport passport passport-local
added 6 packages, and audited 856 packages in 7s

161 packages are looking for funding
  run 'npm fund' for details
```

- Cài đặt WebSocket (có thể dùng cho trò chơi trực tuyến):

```
PS D:\Test\AASCTest2\full-solution-test2> npm install @nestjs/websockets socket.io
added 23 packages, and audited 879 packages in 9s

161 packages are looking for funding
  run 'npm fund' for details
```

2. Generate User Module, Controller và Service

- Mở **Terminal** chạy:

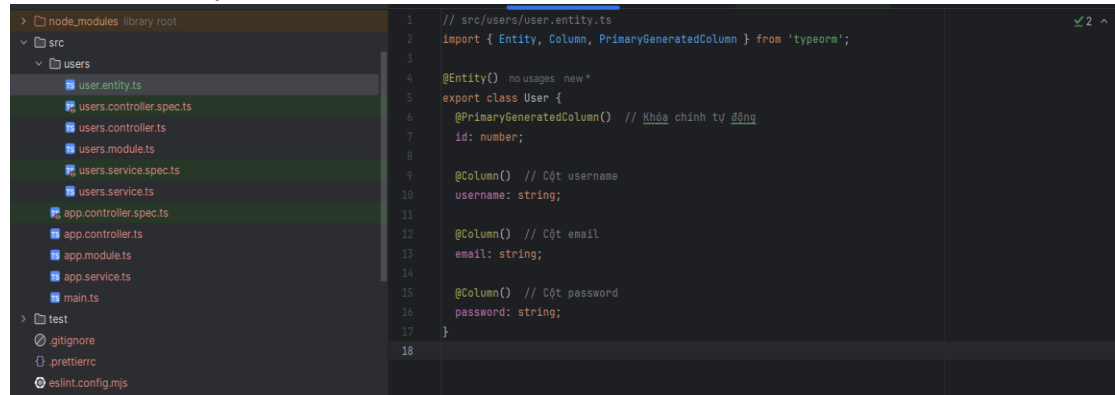
```
PS D:\Test\AASCTest2\full-solution-test2> nest generate module users
CREATE src/users/users.module.ts (86 bytes)
UPDATE src/app.module.ts (312 bytes)
PS D:\Test\AASCTest2\full-solution-test2> nest generate controller users
CREATE src/users/users.controller.ts (103 bytes)
CREATE src/users/users.controller.spec.ts (503 bytes)
UPDATE src/users/users.module.ts (174 bytes)
PS D:\Test\AASCTest2\full-solution-test2> nest generate service users
CREATE src/users/users.service.ts (93 bytes)
```


3. Cấu hình MySQL với TypeORM

- Cấu hình TypeORM trong app.module.ts, chỉnh sửa trong file
Mở file **app.module.ts** và cấu hình **TypeORM** để kết nối với cơ sở dữ liệu **MySQL**.

```
@Module({ Show usages
  imports: [
    TypeOrmModule.forRoot({
      type: 'mysql', // Loại cơ sở dữ liệu là MySQL
      host: 'localhost', // Địa chỉ của MySQL server
      port: 3306, // Cổng MySQL mặc định
      username: 'root', // Tên người dùng MySQL
      password: 'password', // Mật khẩu MySQL
      database: 'nestjs_db', // Tên cơ sở dữ liệu
      entities: [__dirname + '/*.entity{.ts,.js}'], // Các Entity
      synchronize: true, // Tự động đồng bộ hóa cơ sở dữ liệu (dành cho môi trường phát triển)
    }),
    UsersModule, // Import UsersModule vào AppModule
  ],
})
```

- Tạo User Entity:



```
1 // src/users/user.entity.ts
2 import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';
3
4 @Entity()
5 export class User {
6   @PrimaryGeneratedColumn() // Khóa chính tự động
7   id: number;
8
9   @Column() // Cột username
10  username: string;
11
12  @Column() // Cột email
13  email: string;
14
15  @Column() // Cột password
16  password: string;
17 }
18
```

- Cấu hình User Controller:

```
import { Controller, Post, Body } from '@nestjs/common';
import { UsersService } from './users.service';
import { CreateUserDto } from './create-user.dto';

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

  @Post('create') // Route để tạo người dùng mới no usages
  create(@Body() createUserDto: CreateUserDto) {
    return this.usersService.create(createUserDto);
  }
}
```

- Cập nhật User Service:

```
@Injectable() Show usages
export class UsersService {
  constructor( no usages
    @InjectRepository(User) // Inject User repository vào service
    private userRepository: Repository<User>,
  ) {}

  // Tạo người dùng mới
  async create(createUserDto: CreateUserDto): Promise<User> { Show usages
    const user = new User();
    user.username = createUserDto.username;
    user.email = createUserDto.email;
    user.password = createUserDto.password;

    return this.userRepository.save(user); // Lưu vào cơ sở dữ liệu
  }
}
```

- Tạo DTO (Data Transfer Object)

```
export class CreateUserDto { Show usages new *
  readonly username: string;
  readonly email: string;
  readonly password: string;
}
```

- Cập nhật module:

```
// src/users/users.module.ts
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UsersController } from './users.controller';
import { UsersService } from './users.service';
import { User } from './user.entity';

@Module({ Show usages
  imports: [TypeOrmModule.forFeature([User])], // Import User repository
  controllers: [UsersController],
  providers: [UsersService],
})
export class UsersModule {}
```

4. Cài đặt Swagger trong NestJS để kiểm tra API

- Chạy lệnh để cài đặt

```
PS D:\Test\AASCTest2\full-solution-test2> npm install @nestjs/swagger swagger-ui-express
added 6 packages, and audited 885 packages in 20s

161 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- Cấu hình Swagger trong main.ts

```
// src/main.ts
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';

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

  // Cấu hình Swagger
  const config = new DocumentBuilder()
    .setTitle('User API')
    .setDescription('The User API description')
    .setVersion('1.0')
    .addTag('users') // Thêm tag để phân loại các API liên quan đến users
    .build();

  // Tạo tài liệu Swagger
  const document = SwaggerModule.createDocument(app, config);

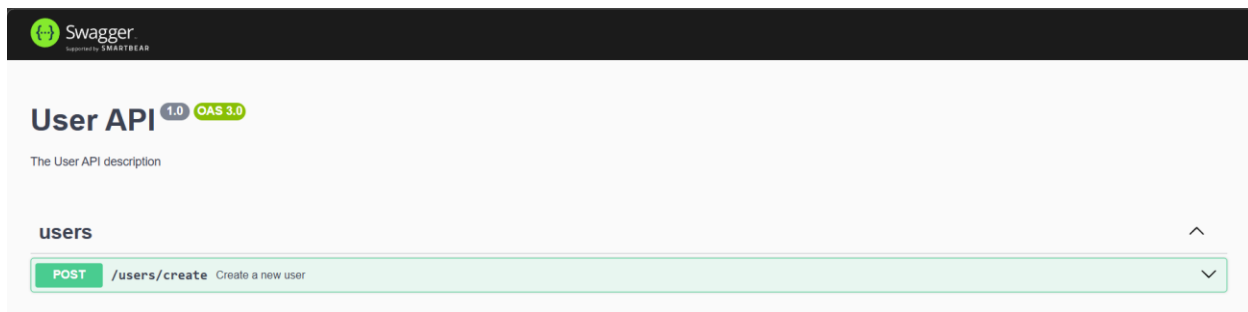
  // Cấu hình Swagger UI tại đường dẫn /api
  SwaggerModule.setup('api', app, document);

  await app.listen(3000);
}

bootstrap();
```

- Sửa lại Controller và Dto để phù hợp.
- Chạy ứng dụng trên localhost:3000/api

Kết quả:



Kết luận

Sau khi hoàn thành các bước triển khai, em đã nắm vững các thành phần cơ bản của **NestJS** và cách triển khai một ứng dụng NestJS theo chuẩn **MVC**. Ứng dụng của bạn sẽ có:

- **Controller**: Xử lý các **HTTP requests** từ người dùng, bao gồm việc nhận dữ liệu và trả về kết quả từ server.
- **Service**: Chứa **logic nghiệp vụ**, xử lý dữ liệu hoặc các yêu cầu phức tạp như tính toán, lưu trữ thông tin vào cơ sở dữ liệu.
- **Module**: Nhóm các **controller** và **service** lại với nhau trong cùng một module để dễ dàng quản lý và tổ chức mã nguồn.
- **DTO (Data Transfer Object)**: Định nghĩa **dữ liệu vào/ra**, giúp đảm bảo rằng dữ liệu nhận vào và trả ra từ API là hợp lệ và đúng định dạng.

Các bước đã thực hiện trong quá trình triển khai:

1. Cài đặt môi trường:

- Cài đặt **Node.js** và **NestJS CLI** để tạo và quản lý các dự án.
- Tạo dự án mới với lệnh `nest new`.
- Cài đặt các **package phụ thuộc** như **TypeORM** và **MySQL driver** để kết nối NestJS với cơ sở dữ liệu.

2. Tạo các thành phần của ứng dụng:

- **Generate** các module, controller và service cho **User** bằng lệnh `nest generate`.
- Cấu hình **MySQL** kết nối với **NestJS** thông qua **TypeORM**.

- Tạo **Entity** cho User và thiết lập các quy tắc lưu trữ người dùng vào cơ sở dữ liệu.
- Tạo **DTO** để định nghĩa cấu trúc dữ liệu khi nhận vào và trả về từ API.

3. Cài đặt và cấu hình Swagger:

- Tích hợp **Swagger** vào ứng dụng để tạo tài liệu API tự động và kiểm tra các endpoints trực tiếp.
- Cấu hình Swagger trong file main.ts và sử dụng decorators từ **@nestjs/swagger** để tạo tài liệu cho controller và service.

4. Kiểm tra ứng dụng:

- Sử dụng **Postman** hoặc **Swagger UI** tại <http://localhost:3000/api> để kiểm tra các API và đảm bảo chúng hoạt động như mong muốn