Day 7 Assignment

Section 1: Theory

1. What is the purpose of a module in a backend framework like NestJS?

In NestJS, a module serves as a way to organize related components such as controllers, providers (services), and other modules. It helps in encapsulating functionality and managing dependencies effectively. Modules provide a modular structure which makes the application more maintainable and scalable. They allow for logical grouping of features and promote separation of concerns within the app.

2. Explain the difference between a Controller and a Service. Why is it beneficial to separate them?

A Controller in NestJS handles incoming HTTP requests and routes them to appropriate services. A Service contains the business logic and is responsible for performing operations like fetching data or computations. Separating them improves code readability, maintainability, and makes it easier to test. It also allows reusability of logic as services can be injected into multiple controllers or other services.

3. What is a DTO and how does it help in maintaining clean architecture?

A DTO (Data Transfer Object) is a plain TypeScript class used to define the shape of data sent over the network. It ensures type safety and data validation at compile time. DTOs help in maintaining clean architecture by clearly defining the data structure and keeping data handling consistent. They separate data structure concerns from business logic, enhancing modularity.

4. Describe the flow of an HTTP request from a client to the server in a NestJS application.

When a client sends a request, it first reaches a Controller based on the route defined. The Controller then calls the appropriate method in the Service for processing. The Service contains the logic to process the request, interact with databases or perform operations. The result is sent back to the Controller, which then sends the response back to the client.

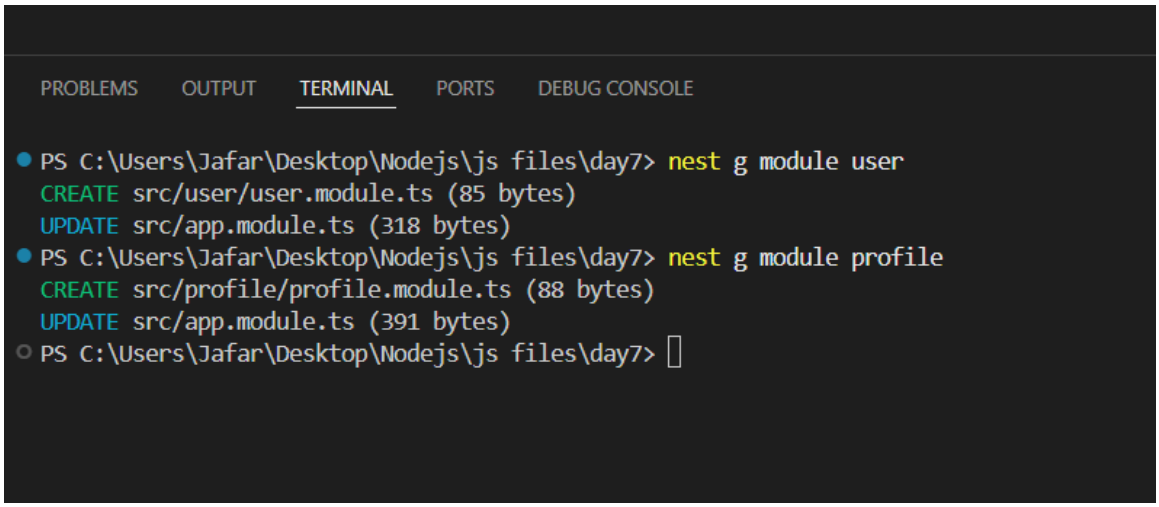
5. What decorators are used in controllers to map HTTP methods to functions? Explain briefly each one of them.

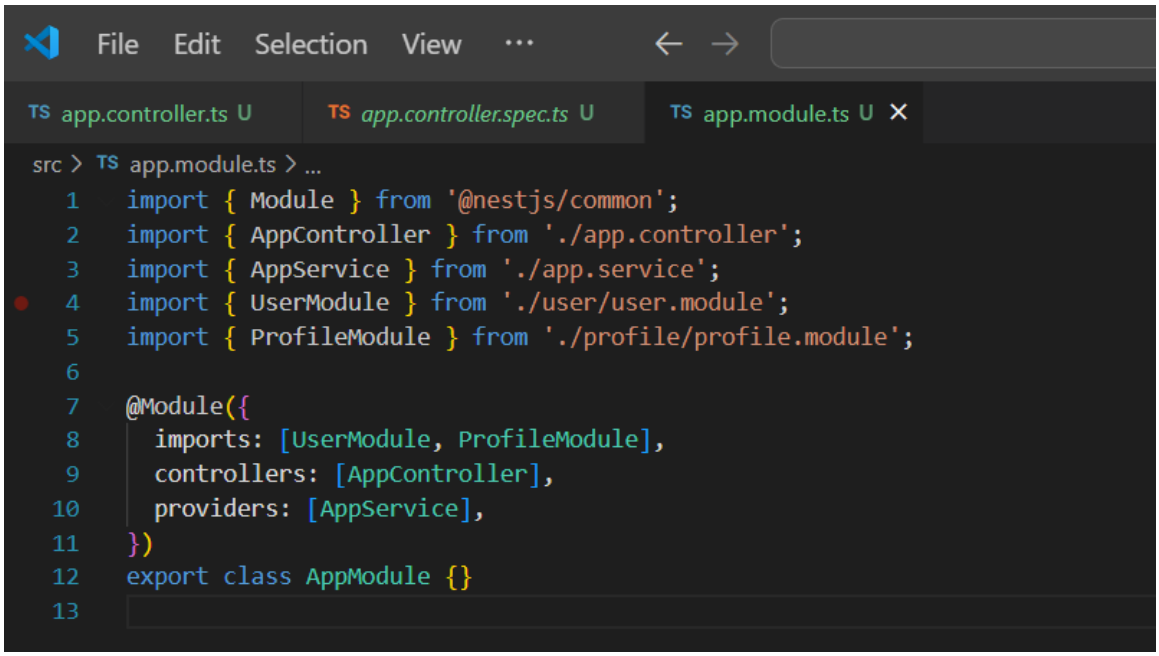
@Get() is used to handle HTTP GET requests. @Post() handles POST requests. @Put() is for updating resources using PUT, while @Delete() handles DELETE operations. Each decorator maps a specific HTTP method to a controller function, making it easy to define RESTful endpoints. These decorators enhance readability and maintainability of code by clearly indicating the request method.

☑ Section 2: Practical (Code-based)

6. Create a UserModule and a ProfileModule, and show how they can be imported into the root AppModule.

The NestJS CLI commands `nest g module user` and `nest g module profile` to generate the modules. To use them in AppModule, import them in the imports array of app.module.ts like this:





7. Write a UserService with createUser and findUserById, and a ProfileService with createProfile and getProfileForUser. Let ProfileService depend on UserService (i.e., one service uses another).

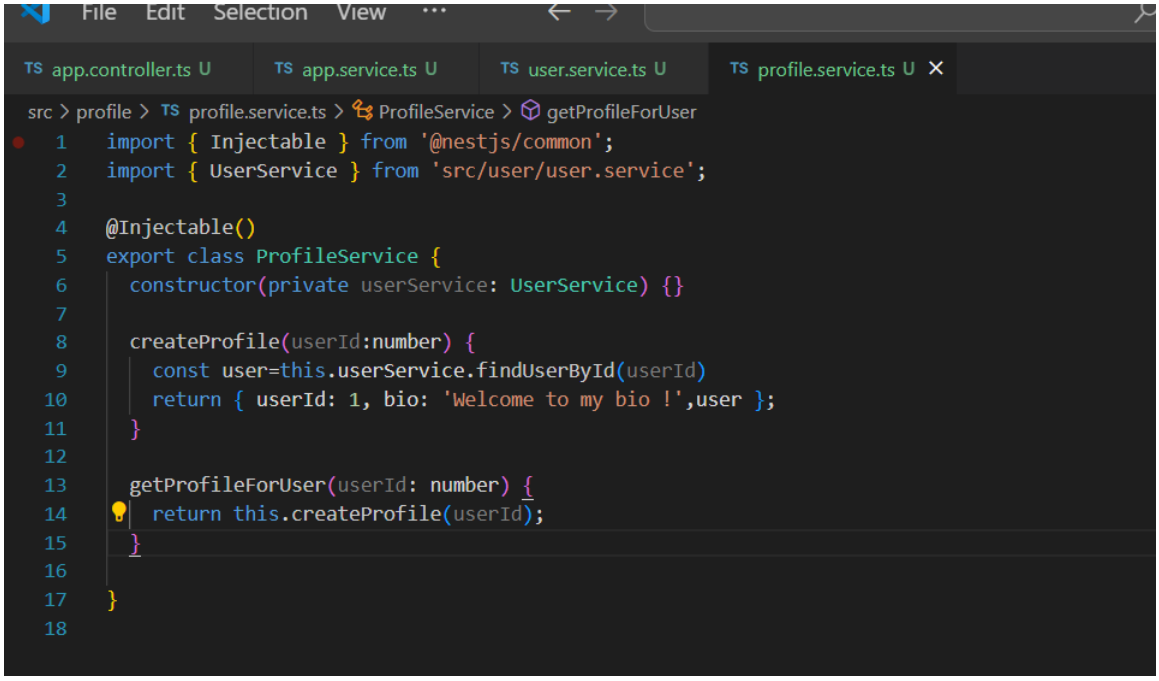
we can inject UserService into ProfileService using constructor injection. This allows ProfileService to access and use methods from UserService. It's an example of dependency injection which is a core concept in NestJS for managing service interactions. Each service handles its own responsibilities, and collaboration is done through injection.

```
PS C:\Users\Jafar\Desktop\Nodejs\js files\day7> nest g service user
>>
CREATE src/user/user.service.ts (92 bytes)
CREATE src/user/user.service.spec.ts (464 bytes)
UPDATE src/user/user.module.ts (159 bytes)
PS C:\Users\Jafar\Desktop\Nodejs\js files\day7> nest g service profile
CREATE src/profile/profile.service.ts (95 bytes)
CREATE src/profile/profile.service.spec.ts (485 bytes)
UPDATE src/profile/profile.module.ts (171 bytes)
```

app.controller.ts U | app.service.ts U | user.service.ts U | profile.service.ts U

src > user > user.service.ts > UserService > findUserById

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


@Injectable()
export class UserService {

    createUser(name: string) {
        return { id: 1, name };
    }

    findUserById(id: number) {
     return { id, name: 'Jafar Beldar' };
    }


}
```

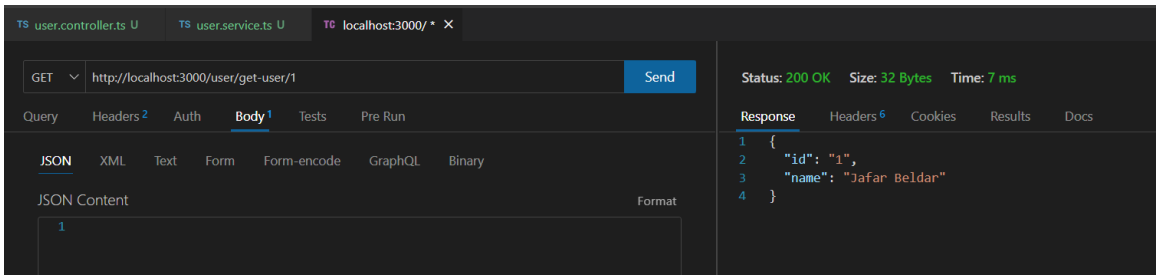


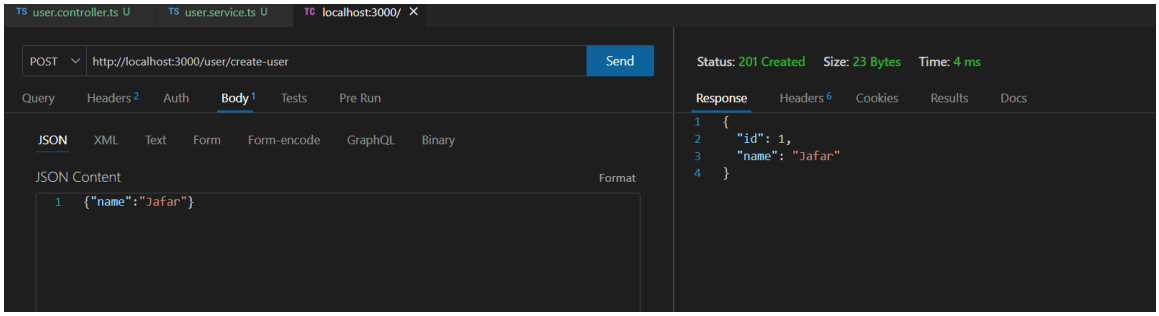
8. Create two controllers: UserController and ProfileController. Each should expose basic CRUD endpoints.

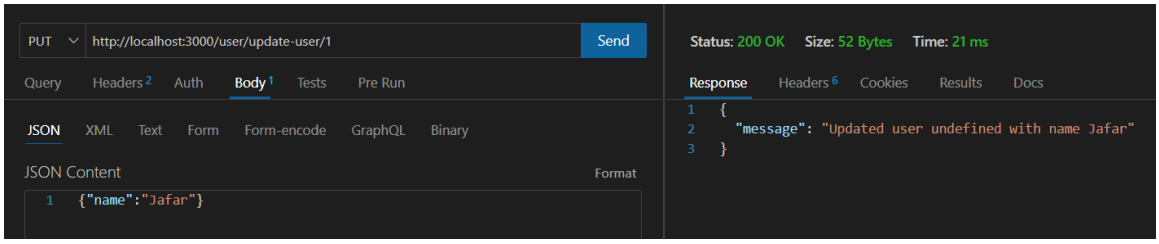
Use the CLI to create controllers with `nest g controller user` and `nest g controller profile`. Each controller defines routes for creating, reading, updating, and deleting users, profiles. The controllers use decorators like @Get(), @Post(), etc., to map routes to functions. These routes interact with services to execute the underlying logic.

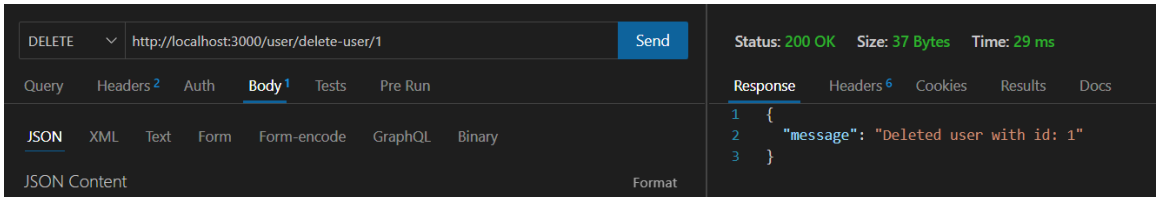
USER CONTROLLER:

```ts
import {
  Body,
  Controller,
  Delete,
  Get,
  Param,
  Post,
  Put,
} from '@nestjs/common';
import { UserService } from './user.service';

@Controller('user')
export class UserController {
  constructor(private userService: UserService) {}

  @Get('get-user/:userId')
  getUserById(@Param('userId') userId: number) {
    return this.userService.findUserById(userId);
  }

  @Post('create-user/')
  createUSer(@Body('name') name: string) {
    return this.userService.createUser(name);
  }

  @Put('update-user/:userId')
  updateUser(@Param('id') id: number, @Body('name') name: string) {
    return { message: `Updated user ${id} with name ${name}` };
  }

  @Delete('delete-user/:userId')
  deleteUser(@Param('userId') userId: number) {
    return { message: `Deleted user with id: ${userId}` };
  }
}

```


user.controller.ts U
user.service.ts U
localhost:3000/ *
GET
http://localhost:3000/user/get-user/1
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
Status: 200 OK
Size: 32 Bytes
Time: 7 ms
Response
Headers 6
Cookies
Results
Docs
{
"id": "1",
"name": "Jafar Beldar"
}

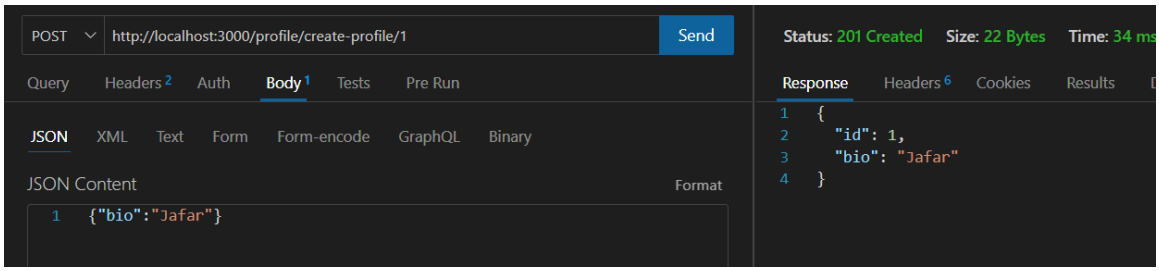

user.controller.ts U
user.service.ts U
localhost:3000/
POST
http://localhost:3000/user/create-user
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{"name":"Jafar"}
Status: 201 Created
Size: 23 Bytes
Time: 4 ms
Response
Headers 6
Cookies
Results
Docs
{
"id": 1,
"name": "Jafar"
}

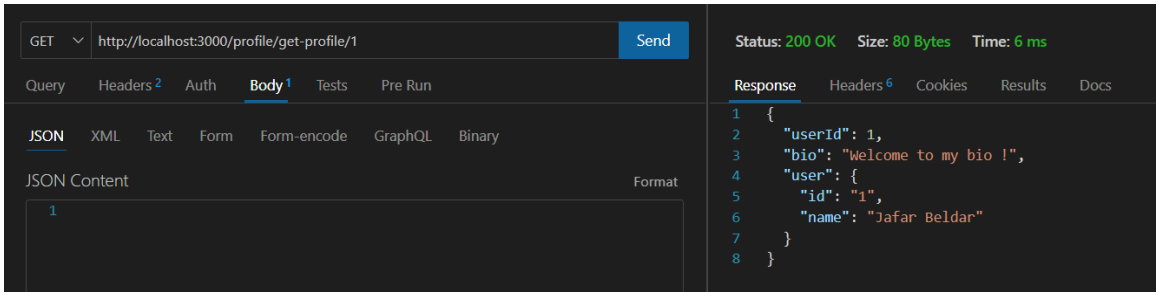

PUT
http://localhost:3000/user/update-user/1
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{"name":"Jafar"}
Status: 200 OK
Size: 52 Bytes
Time: 21 ms
Response
Headers 6
Cookies
Results
Docs
{
"message": "Updated user undefined with name Jafar"
}

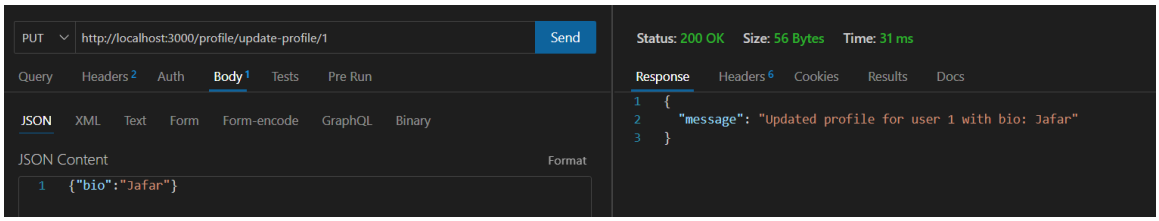

DELETE
http://localhost:3000/user/delete-user/1
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
Status: 200 OK
Size: 37 Bytes
Time: 29 ms
Response
Headers 6
Cookies
Results
Docs
{
"message": "Deleted user with id: 1"
}

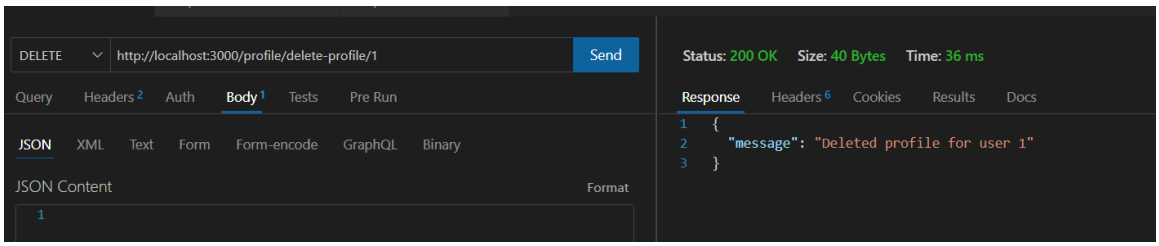
PROFILE :

```ts
import {
  Body,
  Controller,
  Delete,
  Get,
  Param,
  Post,
  Put,
} from '@nestjs/common';
import { ProfileService } from './profile.service';

@Controller('profile')
export class ProfileController {
  constructor(private profileService: ProfileService) {}

  @Post('create-profile/:userId')
  create(@Param('userId') userId: number, @Body('bio') bio: string) {
    return this.profileService.createProfile(bio);
  }

  @Get('get-profile/:userId')
  get(@Param('userId') userId: number) {
    return this.profileService.getProfileForUser(userId);
  }

  @Put('update-profile/:userId')
  update(@Param('userId') userId: number, @Body('bio') bio: string) {
    return { message: `Updated profile for user ${userId} with bio: ${bio}` };
  }

  @Delete('delete-profile/:userId')
  delete(@Param('userId') userId: number) {
    return { message: `Deleted profile for user ${userId}` };
  }
}
```


POST
http://localhost:3000/profile/create-profile/1
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{"bio":"Jafar"}
Status: 201 Created
Size: 22 Bytes
Time: 34 ms
Response
Headers 6
Cookies
Results
{
"id": 1,
"bio": "Jafar"
}

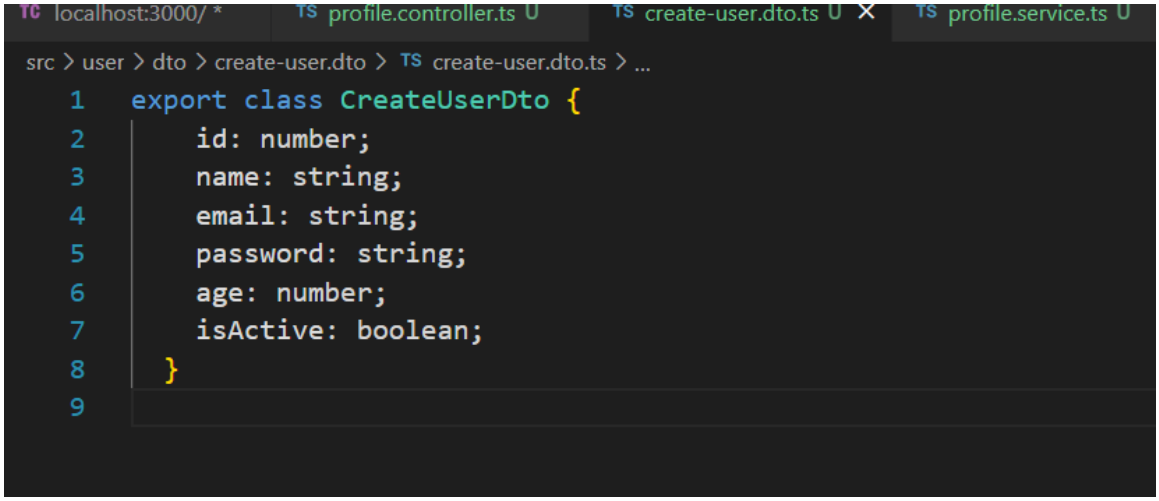

GET
http://localhost:3000/profile/get-profile/1
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
Status: 200 OK
Size: 80 Bytes
Time: 6 ms
Response
Headers 6
Cookies
Results
Docs
{
"userId": 1,
"bio": "Welcome to my bio !",
"user": {
"id": "1",
"name": "Jafar Beldar"
}
}

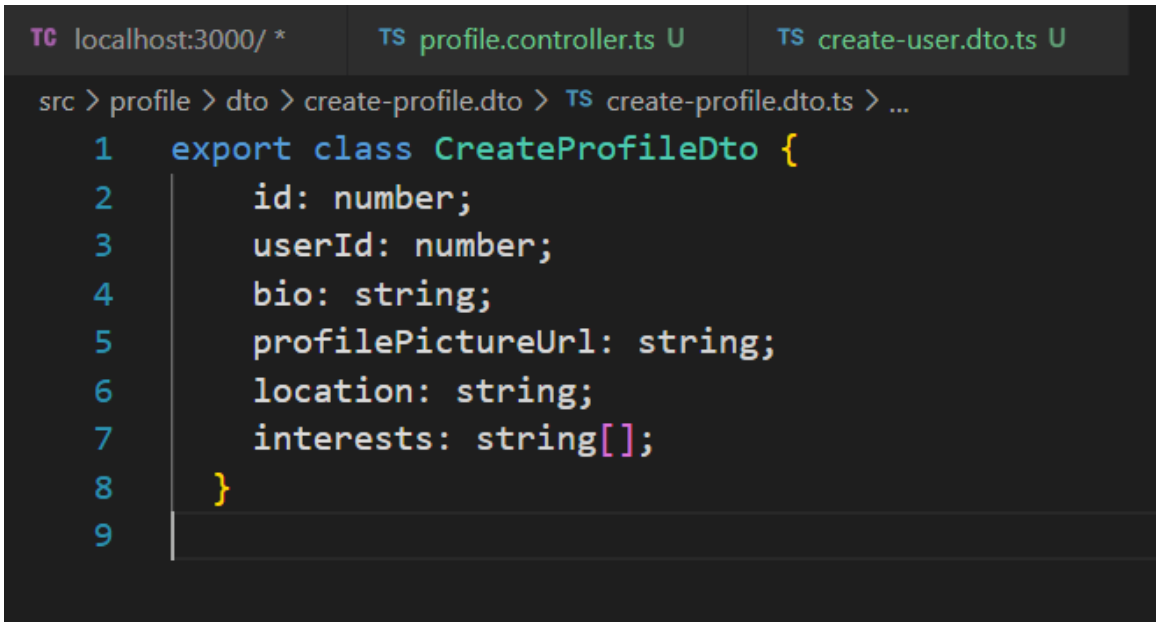

PUT
http://localhost:3000/profile/update-profile/1
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{"bio":"Jafar"}
Status: 200 OK
Size: 56 Bytes
Time: 31 ms
Response
Headers 6
Cookies
Results
Docs
{
"message": "Updated profile for user 1 with bio: Jafar"
}


DELETE
http://localhost:3000/profile/delete-profile/1
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
Status: 200 OK
Size: 40 Bytes
Time: 36 ms
Response
Headers 6
Cookies
Results
Docs
{
"message": "Deleted profile for user 1"
}

9. Define a CreateUserDto and a CreateProfileDto. Both should be plain TypeScript classes with appropriate fields (no decorators required).

DTOs can be created as simple classes like `class CreateUserDto { name: string; email: string; }`. Similarly, CreateProfileDto might have fields like bio, userId, etc. Using DTOs ensures the data format is well-defined and enforces type safety. They act as a contract for the data that a controller should expect.





10. Demonstrate with code how to inject one service (e.g., UserService) into another module's service (e.g., ProfileService) using NestJS dependency injection.

First, UserService is exported in UserModule using the `exports` array. Then, import UserModule in ProfileModule's imports. In ProfileService, we inject UserService via the constructor. This allows services across modules to communicate and use each other's functionalities.

Step 1: Export UserService in UserModule

```ts
// user.module.ts
import { Module } from '@nestjs/common';
import { UserService } from './user.service';
import { UserController } from './user.controller';

@Module({
  providers: [UserService],
  controllers: [UserController],
  exports: [UserService], //  for injection into other modules
})
export class UserModule {}
```

Step 2: Import UserModule in ProfileModule

```
// profile.module.ts
import { Module } from '@nestjs/common';
import { ProfileService } from './profile.service';
import { ProfileController } from './profile.controller';
import { UserModule } from '../user/user.module'; //  importing UserModule

@Module({
  imports: [UserModule], // important for using exported services
  providers: [ProfileService],
  controllers: [ProfileController],
})
export class ProfileModule {}
```

Step 3: Inject UserService in ProfileService

```
  // profile.service.ts
import { Injectable } from '@nestjs/common';
import { UserService } from '../user/user.service';

@Injectable()
export class ProfileService {
  constructor(private readonly userService: UserService) {}

  getUserData(id: number) {
    return this.userService.findUserById(id); // using method from UserService
  }
}
```