NestJS – Day 6 Assignment

Theory-Based Questions (Short Answer)

1. What is NestJS, and what are its main advantages compared to Express?

NestJS is a Node.js framework for building scalable server-side applications using TypeScript. Compared to Express, it provides a modular structure, built-in support for dependency injection, and follows OOP and functional programming principles.

2. Explain the role of the following in a NestJS application:
    - @Controller()
    - @Injectable()
    - @Module()

@Controller() defines a controller class that handles HTTP requests.
@Injectable() marks a class as available for dependency injection.
@Module() organizes code into modules to manage related components.

3. Describe what the main.ts file is responsible for in a NestJS project.

The main.ts file is the entry point of a NestJS app. It bootstraps the application by creating an instance of the app module.

4. What is Dependency Injection, and how does NestJS implement it?

Dependency Injection is a design pattern that allows a class to receive its dependencies from outside. NestJS uses @Injectable and the providers array in modules to handle DI automatically.

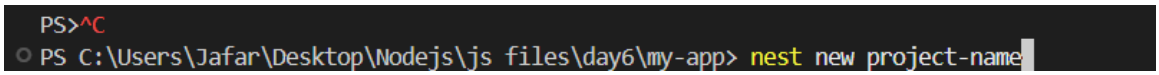
5. List and explain any 3 CLI commands you can use with the NestJS CLI. Provide an example usage of each.

1. nest new project-name → Creates a new NestJS project.
2. nest g module user → Generates a new module.
3. nest g controller user → Generates a new controller in the user module.

Practical / Hands-On Questions

6. Create a new NestJS application using the CLI. What command did you run, and what is the structure of the generated project?

Command: nest new project-name

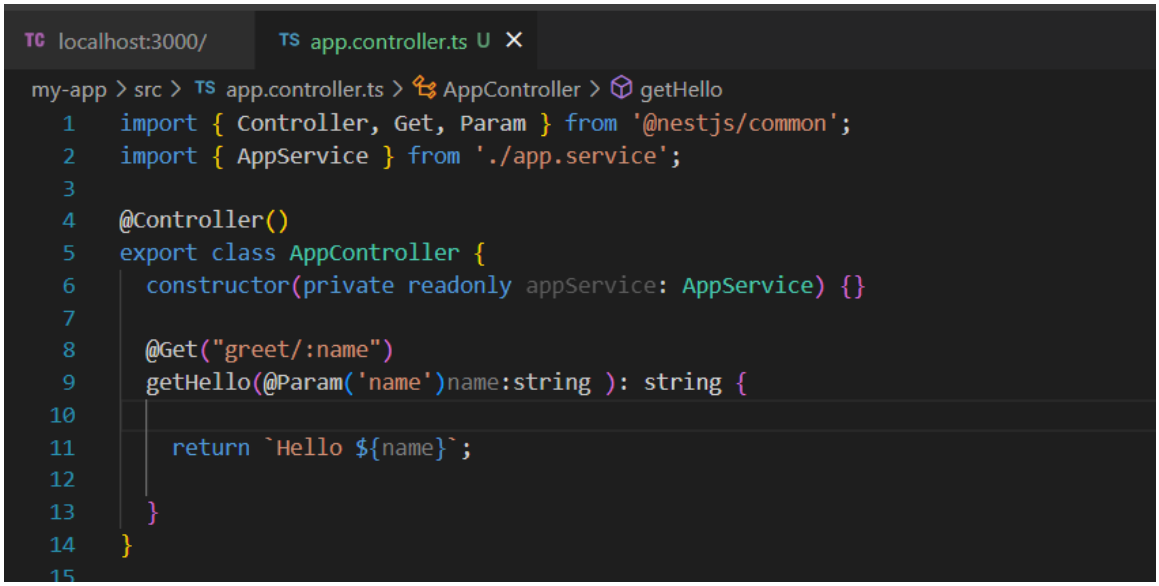


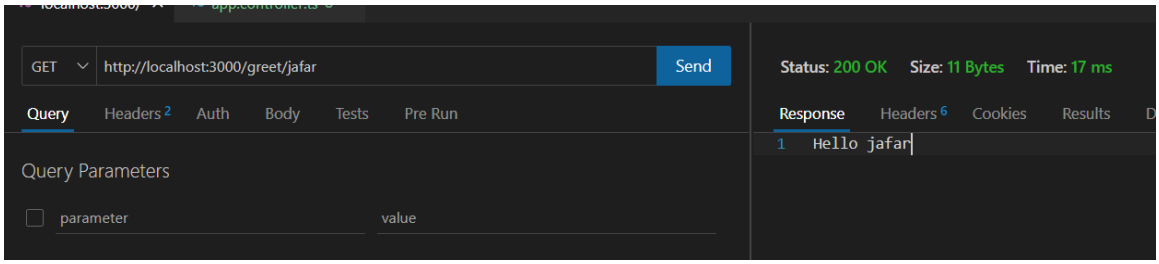
Structure:

my-app/

```
├──── dist/                # Compiled output (ignored by .gitignore)
├──── node_modules/        # Installed dependencies (ignored by .gitignore)
├──── src/                 # Source code
│    ├──── app.controller.spec.ts # Unit test for AppController
│    ├──── app.controller.ts     # Controller handling HTTP requests
│    ├──── app.module.ts         # Root module of the application
│    ├──── app.service.ts        # Service providing business logic
│    └──── main.ts               # Entry point of the application
├──── test/                # End-to-end (e2e) tests
│    ├──── app.e2e-spec.ts       # E2E test for AppController
│    └──── jest-e2e.json         # Jest configuration for e2e tests
├──── .gitignore           # Files and directories to ignore in Git
├──── .prettierrc          # Prettier configuration
├──── eslint.config.mjs    # ESLint configuration
├──── nest-cli.json        # Nest CLI configuration
├──── package.json         # Project metadata and dependencies
├──── README.md            # Project documentation
├──── tsconfig.build.json  # TypeScript configuration for building the app
├──── tsconfig.json        # TypeScript configuration for development
```

7. Add a new GET route /greet/:name that returns a personalized message like Hello, John!. Share the updated controller code.



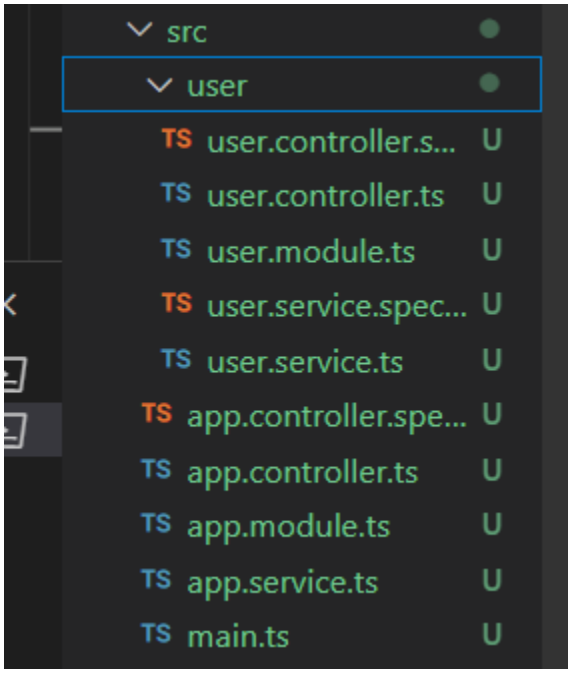


8. Generate a new module named user using the CLI. Then, add a controller and service to this module. What are the commands and file paths created?

Commands:
- nest g module user
- nest g controller user
- nest g service user

```
PS C:\Users\Jafar\Desktop\Nodejs\js files\day6\my-app> nest g module user
CREATE src/user/user.module.ts (85 bytes)
UPDATE src/app.module.ts (318 bytes)
PS C:\Users\Jafar\Desktop\Nodejs\js files\day6\my-app> nest g controller user
CREATE src/user/user.controller.ts (101 bytes)
CREATE src/user/user.controller.spec.ts (496 bytes)
UPDATE src/user/user.module.ts (170 bytes)
PS C:\Users\Jafar\Desktop\Nodejs\js files\day6\my-app> 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 (244 bytes)
PS C:\Users\Jafar\Desktop\Nodejs\js files\day6\my-app>
```

Files created:
src/user/user.module.ts
src/user/user.controller.ts
src/user/user.service.ts



9. Modify the AppService to return a list of technologies (as a string array) via a new route /tech-stack. Provide the updated service and controller code.

Service:

```
getTechStack() {
 return ['NestJS', 'TypeScript', 'Node.js'];
}
```

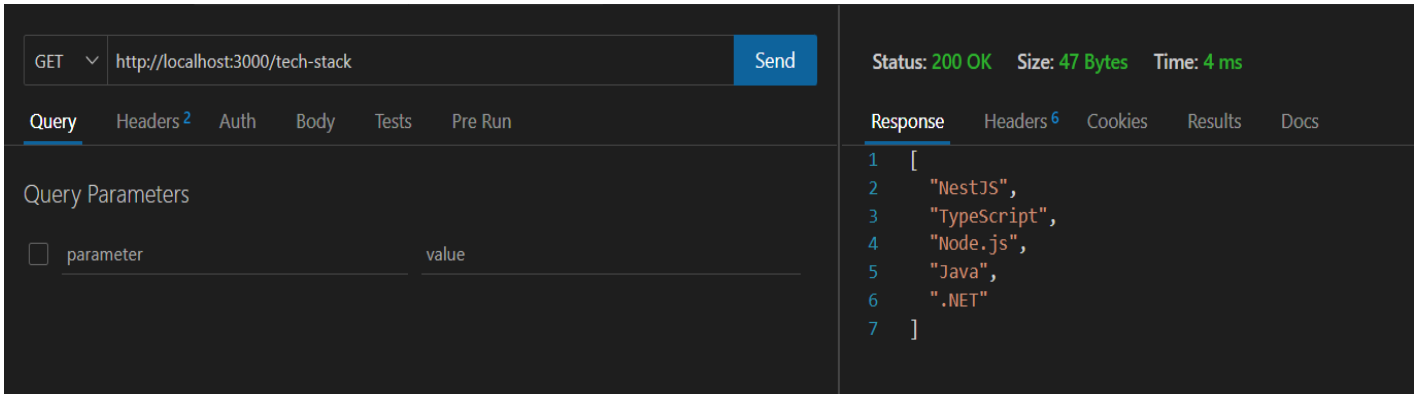
Controller:

```
@Get('tech-stack')
getStack() {
 return this.appService.getTechStack();
}
```

my-app > src > app.service.ts > ...

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

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

  getTechStack(): string[] {
    return ['NestJS', 'TypeScript', 'Node.js','Java','.NET'];

  }

}

```

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

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

  @Get("greet/:name")
  getHello(@Param('name')name:string ): string {

    return `Hello ${name}`;

  }

  @Get("/tech-stack")
  getStack(){
    return this.appService.getTechStack()
  }

}

```



10. Explain how you would test your routes in a browser or using a tool like Postman. What URL did you use for testing your /greet/:name endpoint?

We can use Postman or browser to make GET requests. Also there is an extension in VSCODE called **ThunderClient** so I have ThunderClient: