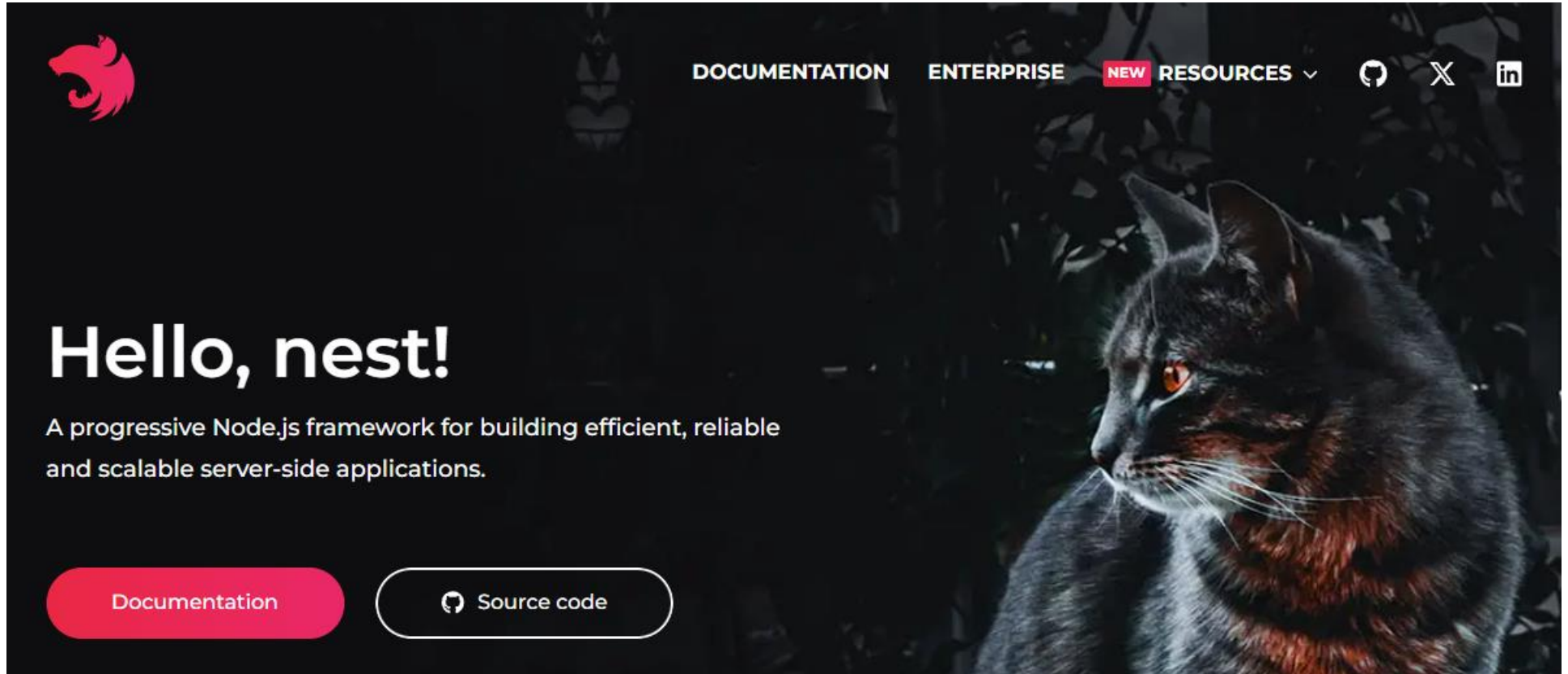
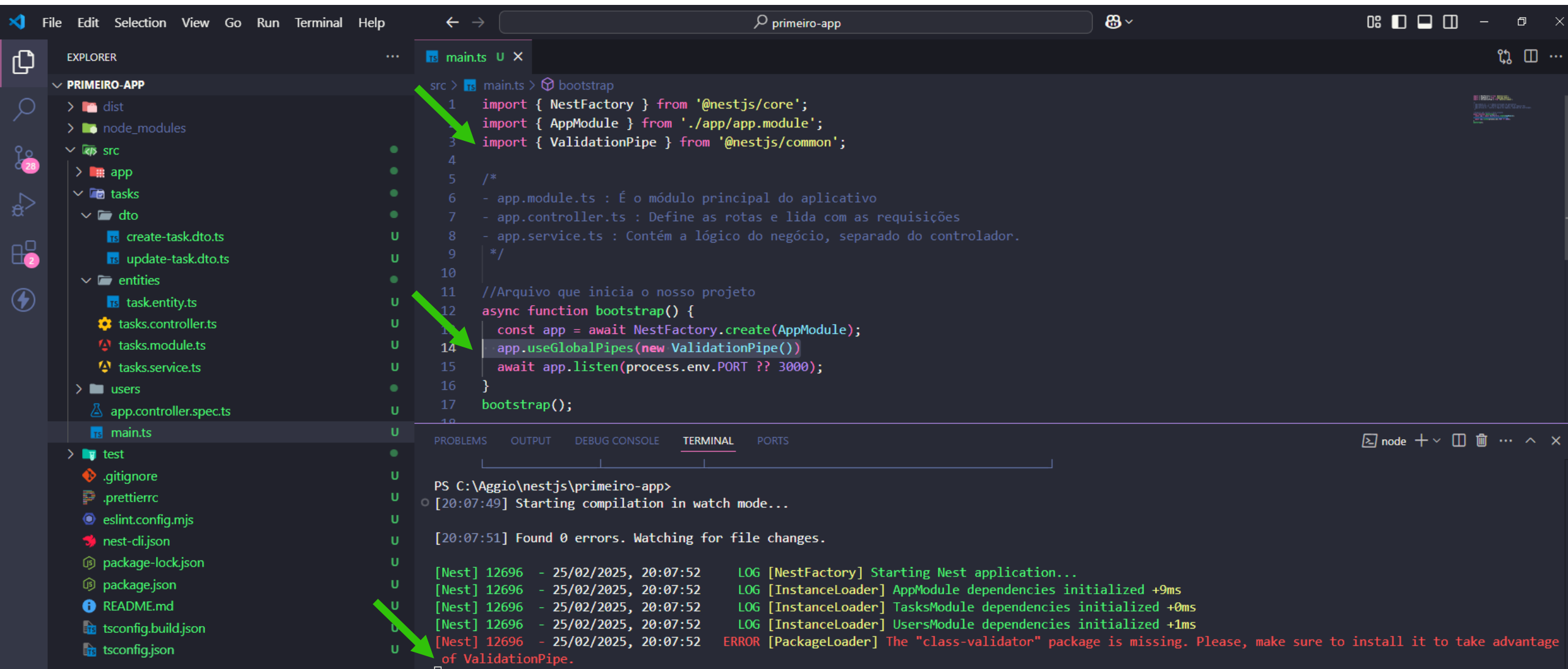


Nest.js



Prof. Dr. Robyson Aggio

www.aggioirati.com.br



The screenshot shows the Visual Studio Code editor interface. The Explorer on the left displays the project structure for 'PRIMEIRO-APP', including folders like 'dist', 'node_modules', 'src', 'tasks', 'dto', 'entities', 'users', and files like 'main.ts', 'test', '.gitignore', '.prettierrc', 'eslint.config.mjs', 'nest-cli.json', 'package-lock.json', 'package.json', 'README.md', 'tsconfig.build.json', and 'tsconfig.json'.

The main editor shows the 'main.ts' file with the following code:

```
src > main.ts > bootstrap
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app/app.module';
3 import { ValidationPipe } from '@nestjs/common';
4
5 /*
6  - app.module.ts : É o módulo principal do aplicativo
7  - app.controller.ts : Define as rotas e lida com as requisições
8  - app.service.ts : Contém a lógica do negócio, separado do controlador.
9  */
10
11 //Arquivo que inicia o nosso projeto
12 async function bootstrap() {
13   const app = await NestFactory.create(AppModule);
14   app.useGlobalPipes(new ValidationPipe());
15   await app.listen(process.env.PORT ?? 3000);
16 }
17 bootstrap();
```

Two green arrows point to the imports of 'ValidationPipe' on line 3 and its usage in 'app.useGlobalPipes' on line 14.

The bottom panel shows the 'TERMINAL' output:

```
PS C:\Aggio\nestjs\primeiro-app>
[20:07:49] Starting compilation in watch mode...

[20:07:51] Found 0 errors. Watching for file changes.

[Nest] 12696 - 25/02/2025, 20:07:52 LOG [NestFactory] Starting Nest application...
[Nest] 12696 - 25/02/2025, 20:07:52 LOG [InstanceLoader] AppModule dependencies initialized +9ms
[Nest] 12696 - 25/02/2025, 20:07:52 LOG [InstanceLoader] TasksModule dependencies initialized +0ms
[Nest] 12696 - 25/02/2025, 20:07:52 LOG [InstanceLoader] UsersModule dependencies initialized +1ms
[Nest] 12696 - 25/02/2025, 20:07:52 ERROR [PackageLoader] The "class-validator" package is missing. Please, make sure to install it to take advantage of ValidationPipe.
```

A green arrow points to the error message in the terminal.



INTRODUCTION

OVERVIEW

- First steps
- Controllers
- Providers
- Modules
- Middleware
- Exception filters
- Pipes
- Guards
- Interceptors
- Custom decorators

FUNDAMENTALS

TECHNIQUES

SECURITY

GRAPHQL

WEBSOCKETS

MICROSERVICES

NEW DEPLOYMENT

STANDALONE APPS

Global scoped pipes

Since the `ValidationPipe` was created to be as generic as possible, we can realize its full utility by setting it up as a **global-scoped** pipe so that it is applied to every route handler across the entire application.

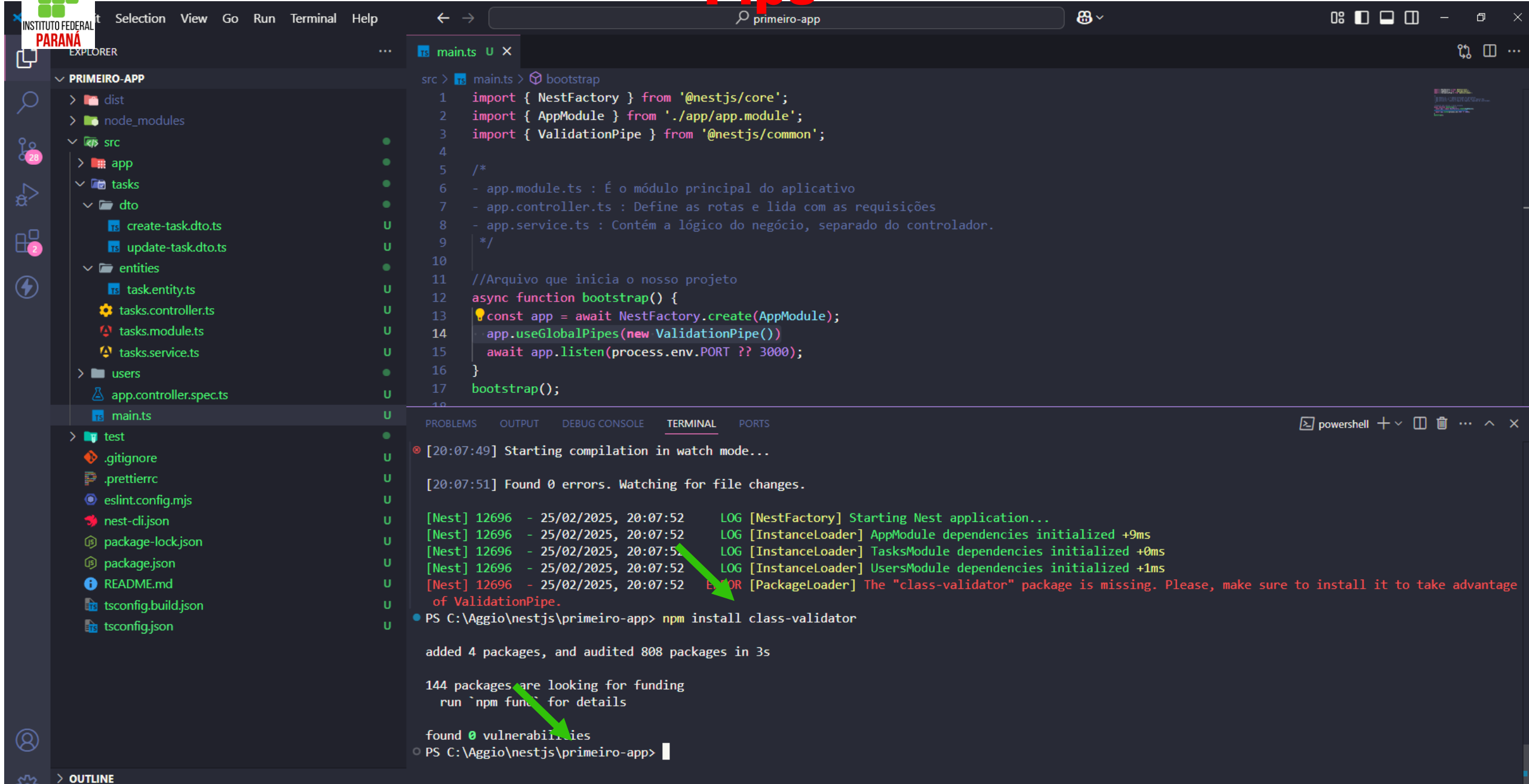
main.ts

JS

```
async function bootstrap() {  
  const app = await NestFactory.create(AppModule);  
  app.useGlobalPipes(new ValidationPipe());  
  await app.listen(process.env.PORT ?? 3000);  
}  
bootstrap();
```

NOTICE

In the case of **hybrid apps** the `useGlobalPipes()` method doesn't set up pipes for gateways and microservices. For "standard" (non-hybrid) microservice apps, `useGlobalPipes()` does mount pipes globally.



INSTITUTO FEDERAL DO PARANÁ

PRIMEIRO-APP

- dist
- node_modules
- src
 - app
 - tasks
 - dto
 - create-task.dto.ts
 - update-task.dto.ts
 - entities
 - task.entity.ts
 - tasks.controller.ts
 - tasks.module.ts
 - tasks.service.ts
 - users
 - app.controller.spec.ts

```

src > main.ts > bootstrap
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app/app.module';
3 import { ValidationPipe } from '@nestjs/common';
4
5 /*
6  - app.module.ts : É o módulo principal do aplicativo
7  - app.controller.ts : Define as rotas e lida com as requisições
8  - app.service.ts : Contém a lógica do negócio, separado do controlador.
9  */
10
11 //Arquivo que inicia o nosso projeto
12 async function bootstrap() {
13   const app = await NestFactory.create(AppModule);
14   app.useGlobalPipes(new ValidationPipe());
15   await app.listen(process.env.PORT ?? 3000);
16 }
17 bootstrap();
18

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

[20:07:49] Starting compilation in watch mode...

[20:07:51] Found 0 errors. Watching for file changes.

[Nest] 12696 - 25/02/2025, 20:07:52 LOG [NestFactory] Starting Nest application...

[Nest] 12696 - 25/02/2025, 20:07:52 LOG [InstanceLoader] AppModule dependencies initialized +9ms

[Nest] 12696 - 25/02/2025, 20:07:52 LOG [InstanceLoader] TasksModule dependencies initialized +0ms

[Nest] 12696 - 25/02/2025, 20:07:52 LOG [InstanceLoader] UsersModule dependencies initialized +1ms

[Nest] 12696 - 25/02/2025, 20:07:52 ERROR [PackageLoader] The "class-validator" package is missing. Please, make sure to install it to take advantage of ValidationPipe.

PS C:\Aggio\nestjs\primeiro-app> npm install class-validator

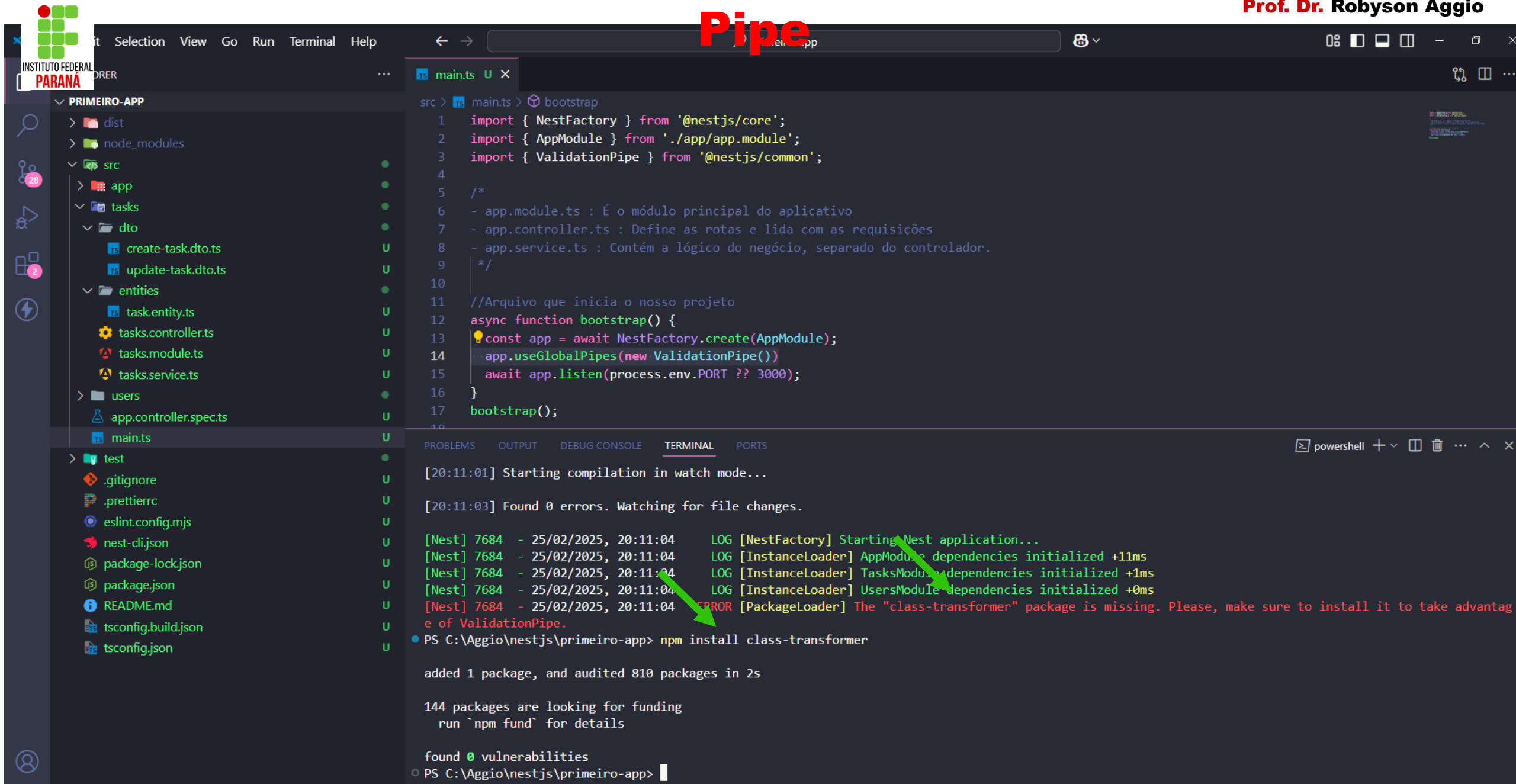
added 4 packages, and audited 808 packages in 3s

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

found 0 vulnerabilities

PS C:\Aggio\nestjs\primeiro-app>

Pipe



The screenshot displays a Visual Studio Code editor window with a project named "PRIMEIRO-APP". The file explorer on the left shows the project structure, including folders like "dist", "node_modules", "src", "tasks", "dto", "entities", "users", and "test". The main editor shows the "main.ts" file with the following code:

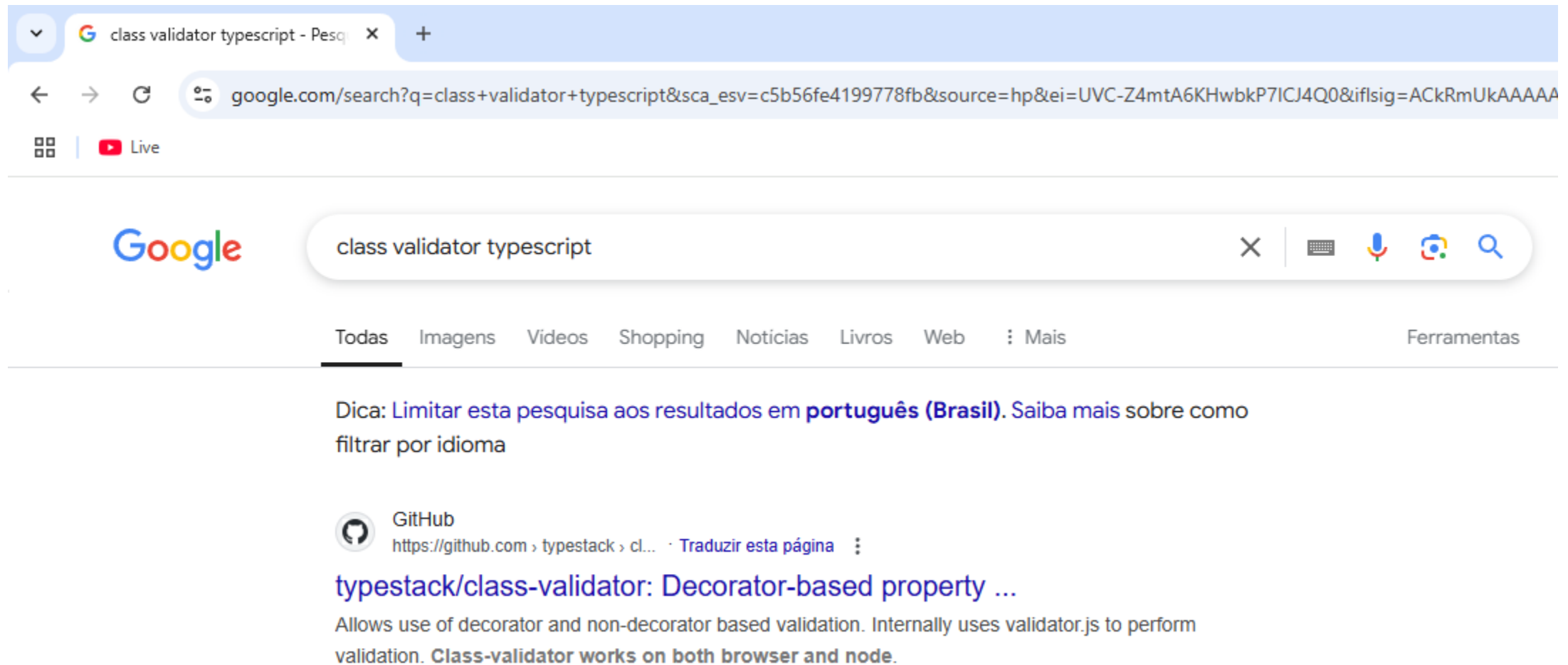
```
src > main.ts > bootstrap
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app/app.module';
3 import { ValidationPipe } from '@nestjs/common';
4
5 /*
6  - app.module.ts : É o módulo principal do aplicativo
7  - app.controller.ts : Define as rotas e lida com as requisições
8  - app.service.ts : Contém a lógica do negócio, separado do controlador.
9  */
10
11 //Arquivo que inicia o nosso projeto
12 async function bootstrap() {
13   const app = await NestFactory.create(AppModule);
14   app.useGlobalPipes(new ValidationPipe());
15   await app.listen(process.env.PORT ?? 3000);
16 }
17 bootstrap();
```

The terminal at the bottom shows the output of the command `npm install class-transformer`:

```
[20:11:01] Starting compilation in watch mode...
[20:11:03] Found 0 errors. Watching for file changes.
[Nest] 7684 - 25/02/2025, 20:11:04 LOG [NestFactory] Starting Nest application...
[Nest] 7684 - 25/02/2025, 20:11:04 LOG [InstanceLoader] AppModule dependencies initialized +11ms
[Nest] 7684 - 25/02/2025, 20:11:04 LOG [InstanceLoader] TasksModule dependencies initialized +1ms
[Nest] 7684 - 25/02/2025, 20:11:04 LOG [InstanceLoader] UsersModule dependencies initialized +0ms
[Nest] 7684 - 25/02/2025, 20:11:04 ERROR [PackageLoader] The "class-transformer" package is missing. Please, make sure to install it to take advantage of ValidationPipe.
PS C:\Aggio\nestjs\primeiro-app> npm install class-transformer
added 1 package, and audited 810 packages in 2s
144 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
PS C:\Aggio\nestjs\primeiro-app>
```

Two green arrows point from the error message in the terminal to the `ValidationPipe` import in the `main.ts` file.

class-validator



The screenshot shows a web browser window with a Google search for "class validator typescript". The search results include a tip to filter by language and a result for the GitHub repository "typestack/class-validator".

class validator typescript - Pesq x +

google.com/search?q=class+validator+typescript&sca_esv=c5b56fe4199778fb&source=hp&ei=UVC-Z4mtA6KHwbkP7ICJ4Q0&iflsig=ACkRmUkAAAAA

Google

class validator typescript

Todas Imagens Videos Shopping Notícias Livros Web : Mais Ferramentas

Dica: Limitar esta pesquisa aos resultados em **português (Brasil)**. Saiba mais sobre como filtrar por idioma

GitHub
https://github.com > typestack > cl... · Traduzir esta página

typestack/class-validator: Decorator-based property ...

Allows use of decorator and non-decorator based validation. Internally uses validator.js to perform validation. **Class-validator works on both browser and node.**

class-validator

README MIT license

Usage

Create your class and put some validation decorators on the properties you want to validate:

```
import {
  validate,
  validateOrReject,
  Contains,
  IsInt,
  Length,
  IsEmail,
  IsFQDN,
  IsDate,
  Min,
  Max,
} from 'class-validator';

export class Post {
  @Length(10, 20)
  title: string;

  @Contains('hello')
  text: string;

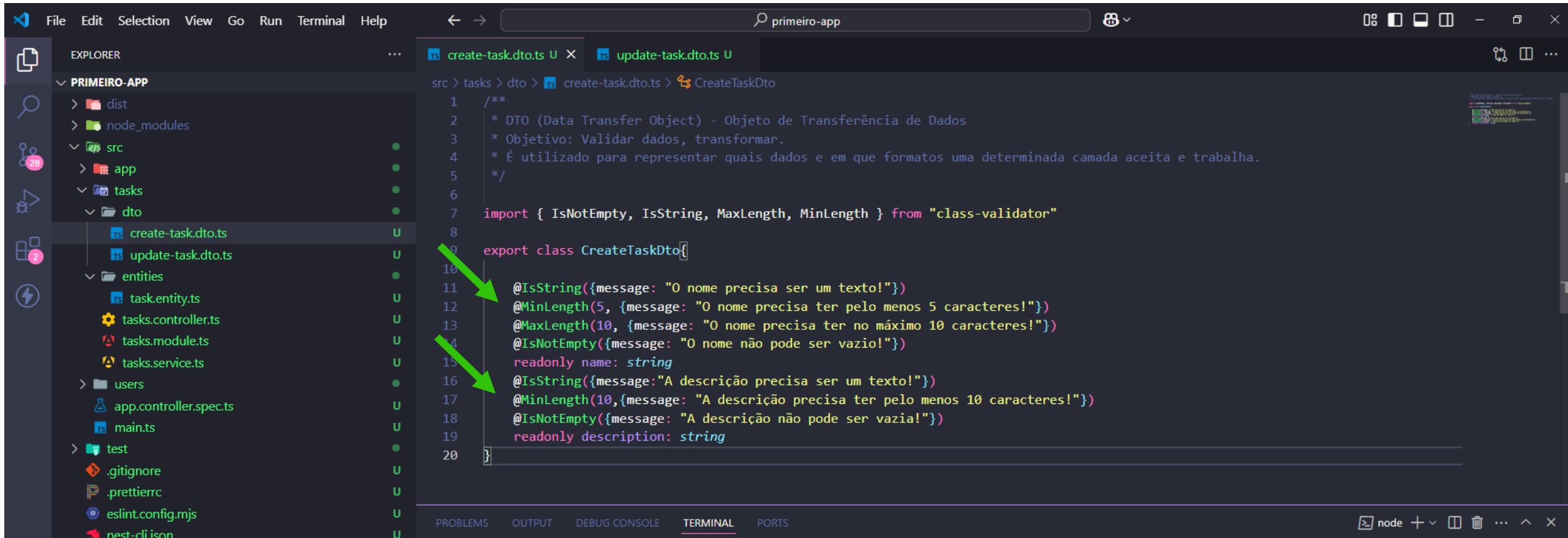
  @IsInt()
  @Min(0)
  @Max(10)
  rating: number;

  @IsEmail()
  email: string;

  @IsFQDN()
  site: string;

  @IsDate()
  createDate: Date;
}
```

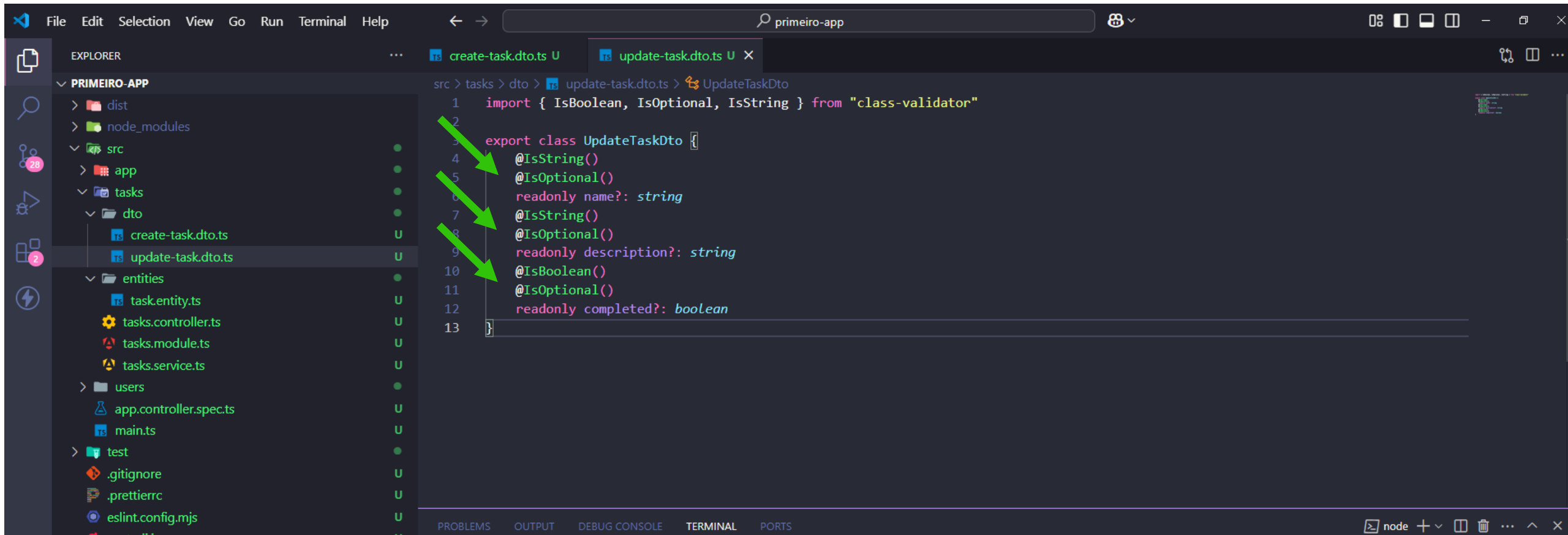
class-validator



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure for 'PRIMEIRO-APP', including folders like 'dist', 'node_modules', 'src', 'tasks', 'dto', 'entities', and 'users'. The 'tasks' folder is expanded, showing 'create-task.dto.ts' and 'update-task.dto.ts'. The 'dto' folder is also expanded, showing 'create-task.dto.ts' and 'update-task.dto.ts'. The 'tasks' folder is selected, and the 'create-task.dto.ts' file is open in the editor. The editor shows the following code:

```
src > tasks > dto > create-task.dto.ts > CreateTaskDto
1  /**
2   * DTO (Data Transfer Object) - Objeto de Transferência de Dados
3   * Objetivo: Validar dados, transformar.
4   * É utilizado para representar quais dados e em que formatos uma determinada camada aceita e trabalha.
5   */
6
7  import { IsNotEmpty, IsString, MaxLength, MinLength } from "class-validator"
8
9  export class CreateTaskDto{
10
11     @IsString({message: "O nome precisa ser um texto!"})
12     @MinLength(5, {message: "O nome precisa ter pelo menos 5 caracteres!"})
13     @MaxLength(10, {message: "O nome precisa ter no máximo 10 caracteres!"})
14     @IsNotEmpty({message: "O nome não pode ser vazio!"})
15     readonly name: string
16
17     @IsString({message: "A descrição precisa ser um texto!"})
18     @MinLength(10, {message: "A descrição precisa ter pelo menos 10 caracteres!"})
19     @IsNotEmpty({message: "A descrição não pode ser vazia!"})
20     readonly description: string
21 }
```

Two green arrows point to the decorators used in the code: one points to the `@IsString` decorator on line 11, and the other points to the `@MinLength` decorator on line 12.



The screenshot shows a VS Code editor window with a project named "primeiro-app". The Explorer sidebar on the left shows the project structure, with the file "update-task.dto.ts" selected under the "tasks" > "dto" directory. The main editor area displays the content of "update-task.dto.ts", which imports decorators from "class-validator" and defines the "UpdateTaskDto" class with several decorated properties. Four green arrows point to the decorators used in the class definition: @IsString() on line 4, @IsOptional() on line 5, @IsString() on line 7, and @IsBoolean() on line 10.

```
src > tasks > dto > update-task.dto.ts > UpdateTaskDto
1  import { IsBoolean, IsOptional, IsString } from "class-validator"
2
3
4  export class UpdateTaskDto {
5      @IsString()
6      @IsOptional()
7      readonly name?: string
8      @IsString()
9      @IsOptional()
10     readonly description?: string
11     @IsBoolean()
12     @IsOptional()
13     readonly completed?: boolean
14 }
```



INSTITUTO FEDERAL
PARANÁ

class-validator

The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the project structure for 'PRIMEIRO-APP'. The 'tasks' folder is expanded, showing 'dto' and 'update-task.dto.ts'. The main editor area shows the content of 'update-task.dto.ts', which imports 'IsBoolean', 'IsOptional', and 'IsString' from 'class-validator' and defines the 'UpdateTaskDto' class with decorators. The bottom panel shows the 'TERMINAL' tab with the command 'npm install @nestjs/mapped-types' and its output.

EXPLORER

PRIMEIRO-APP

- dist
- node_modules
- src
 - app
 - tasks
 - dto
 - create-task.dto.ts
 - update-task.dto.ts
 - entities
 - task.entity.ts
 - tasks.controller.ts
 - tasks.module.ts
 - tasks.service.ts
- users
 - app.controller.spec.ts
 - main.ts
- test
 - .gitignore
 - .prettierrc
 - eslint.config.mjs
 - nest-cli.json
 - package-lock.json

src > tasks > dto > update-task.dto.ts > UpdateTaskDto

```
1 import { IsBoolean, IsOptional, IsString } from "class-validator"
2
3 export class UpdateTaskDto {
4   @IsString()
5   @IsOptional()
6   readonly name?: string
7   @IsString()
8   @IsOptional()
9   readonly description?: string
10  @IsBoolean()
11  @IsOptional()
12  readonly completed?: boolean
13 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

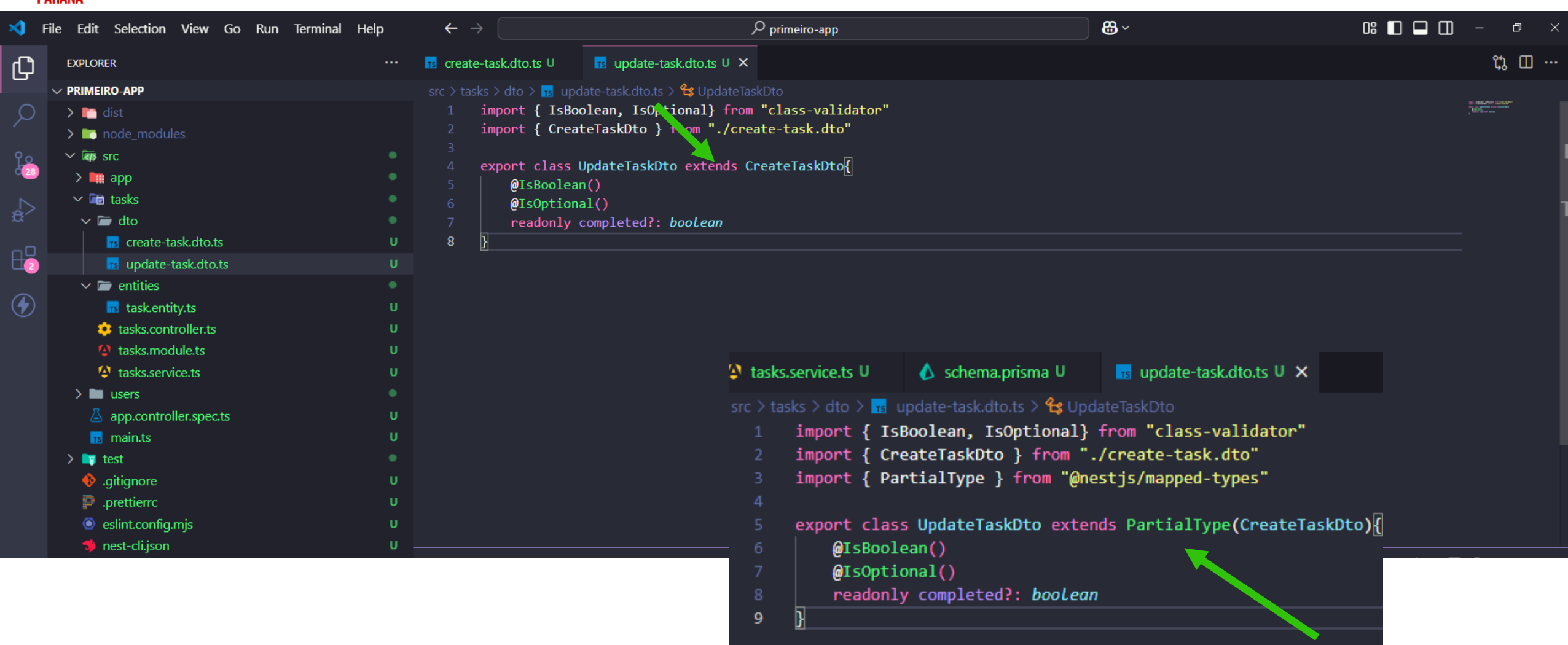
PS C:\Aggio\nestjs\primeiro-app> npm install @nestjs/mapped-types

up to date, audited 810 packages in 2s

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

found 0 vulnerabilities

PS C:\Aggio\nestjs\primeiro-app>



The image shows a VS Code editor window with the file explorer on the left and the code editor on the right. The file explorer shows the project structure for 'PRIMEIRO-APP', including folders like 'dist', 'node_modules', 'src', 'tasks', 'dto', 'entities', 'users', and 'test'. The code editor displays the 'update-task.dto.ts' file, which defines the 'UpdateTaskDto' class. A green arrow points to the 'extends' keyword in the class definition, highlighting the inheritance from 'CreateTaskDto'.

```
src > tasks > dto > update-task.dto.ts > UpdateTaskDto
1 import { IsBoolean, IsOptional } from "class-validator"
2 import { CreateTaskDto } from "../create-task.dto"
3
4 export class UpdateTaskDto extends CreateTaskDto {
5   @IsBoolean()
6   @IsOptional()
7   readonly completed?: boolean
8 }
```

The code editor also shows the 'tasks.service.ts' file, which defines the 'UpdateTaskService' class. A green arrow points to the 'extends' keyword in the class definition, highlighting the inheritance from 'CreateTaskService'.

```
src > tasks > dto > update-task.dto.ts > UpdateTaskDto
1 import { IsBoolean, IsOptional } from "class-validator"
2 import { CreateTaskDto } from "../create-task.dto"
3 import { PartialType } from "@nestjs/mapped-types"
4
5 export class UpdateTaskDto extends PartialType(CreateTaskDto) {
6   @IsBoolean()
7   @IsOptional()
8   readonly completed?: boolean
9 }
```

Transformar Dados - Pipe

EXPLORER

PRIMEIRO-APP

- > dist
- > node_modules
- ▼ src
 - > app
 - ▼ tasks
 - ▼ dto
 - create-task.dto.ts
 - update-task.dto.ts
 - ▼ entities
 - task.entity.ts
 - tasks.controller.ts
 - tasks.module.ts
 - tasks.service.ts
 - > users
 - app.controller.spec.ts
 - main.ts
 - > test
 - .gitignore
 - .prettierrc
 - eslint.config.mjs
 - nest-cli.json
 - package-lock.json
 - package.json
 - README.md
 - tsconfig.build.json
 - tsconfig.json

```
src > ts main.ts > bootstrap
1  import { NestFactory } from '@nestjs/core';
2  import { AppModule } from './app/app.module';
3  import { ValidationPipe } from '@nestjs/common';
4
5  /*
6   - app.module.ts : É o módulo principal do aplicativo
7   - app.controller.ts : Define as rotas e lida com as requisições
8   - app.service.ts : Contém a lógica do negócio, separado do controlador.
9   */
10
11 //Arquivo que inicia o nosso projeto
12 async function bootstrap() {
13   const app = await NestFactory.create(AppModule);
14   app.useGlobalPipes(new ValidationPipe({
15     whitelist: true //Se o usuário enviar algum parâmetro inexistente!
16   }));
17   await app.listen(process.env.PORT ?? 3000);
18 }
19 bootstrap();
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

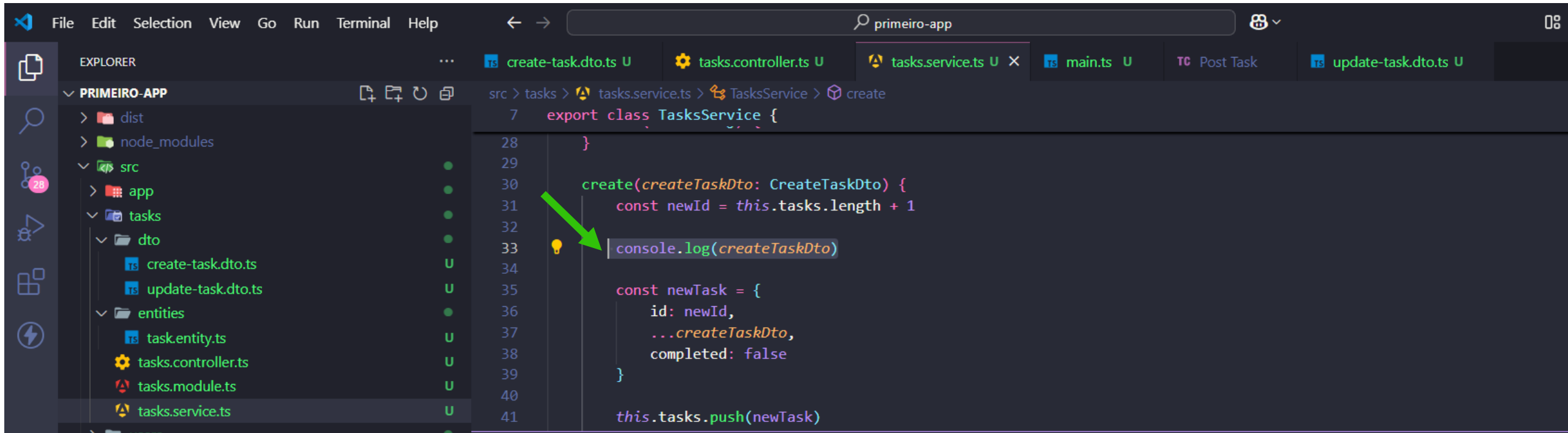
```
PS C:\Aggio\nestjs\primeiro-app> npm install @nestjs/mapped-types

up to date, audited 810 packages in 2s

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

found 0 vulnerabilities
PS C:\Aggio\nestjs\primeiro-app>
```

Transformar Dados - Pipe

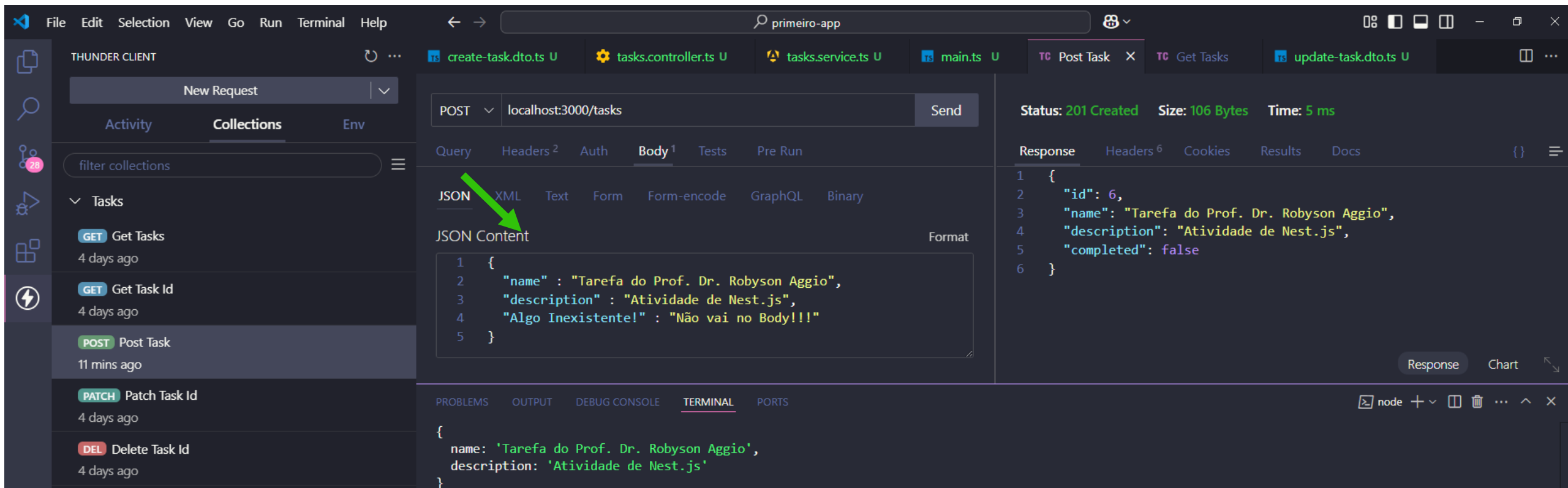


The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure for 'PRIMEIRO-APP', including folders like 'dist', 'node_modules', 'src', 'app', 'tasks', 'dto', and 'entities'. The 'tasks' folder is expanded, showing files like 'create-task.dto.ts', 'update-task.dto.ts', 'task.entity.ts', 'tasks.controller.ts', 'tasks.module.ts', and 'tasks.service.ts'. The code editor shows the 'tasks.service.ts' file with the following code:

```
export class TaskService {  
  28  
  29  
  30  create(createTaskDto: CreateTaskDto) {  
  31    const newId = this.tasks.length + 1  
  32  
  33    console.log(createTaskDto)  
  34  
  35    const newTask = {  
  36      id: newId,  
  37      ...createTaskDto,  
  38      completed: false  
  39    }  
  40  
  41    this.tasks.push(newTask)  
  42  }  
}
```

A green arrow points to the `console.log(createTaskDto)` line, indicating a logging step in the data transformation process.

Transformar Dados - Pipe



The screenshot displays the Thunder Client interface with a REST client request and response. The request is a POST to `localhost:3000/tasks` with a JSON body. The response is a 201 Created status with a JSON body. A green arrow points to the 'JSON Content' tab in the request body section.

Request:

- Method: POST
- URL: localhost:3000/tasks
- Body (JSON Content):

```
1 {
2   "name" : "Tarefa do Prof. Dr. Robyson Aggio",
3   "description" : "Atividade de Nest.js",
4   "Algo Inexistente!" : "Não vai no Body!!!"
5 }
```

Response:

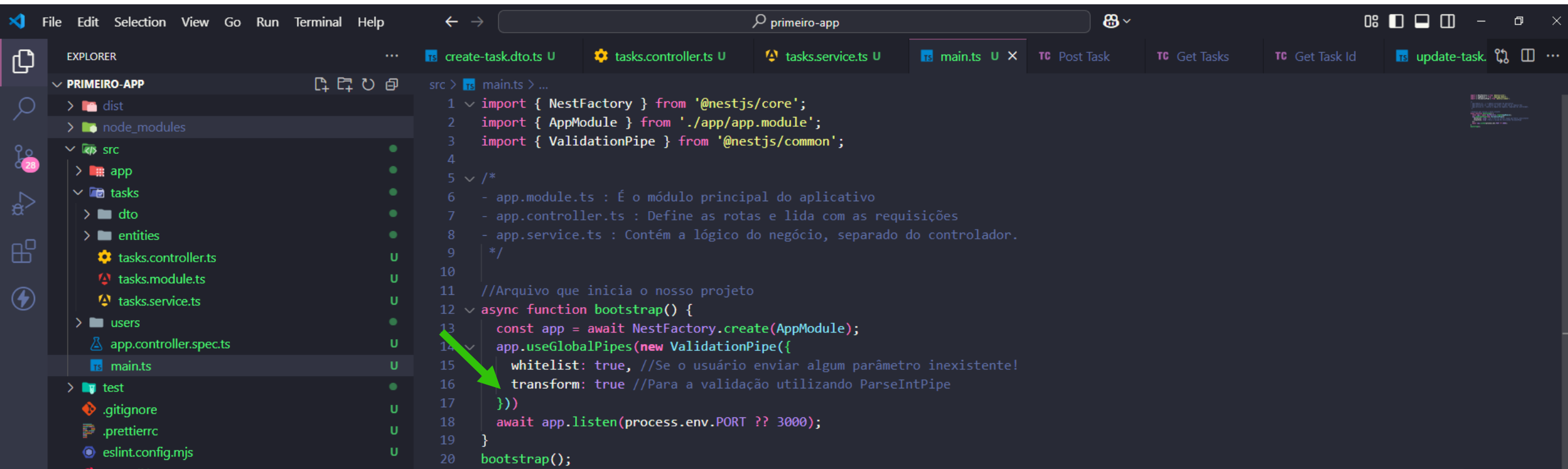
- Status: 201 Created
- Size: 106 Bytes
- Time: 5 ms
- Body (JSON):

```
1 {
2   "id": 6,
3   "name": "Tarefa do Prof. Dr. Robyson Aggio",
4   "description": "Atividade de Nest.js",
5   "completed": false
6 }
```

Terminal:

```
{
  name: 'Tarefa do Prof. Dr. Robyson Aggio',
  description: 'Atividade de Nest.js'
}
```

Transformar Dados - Pipe

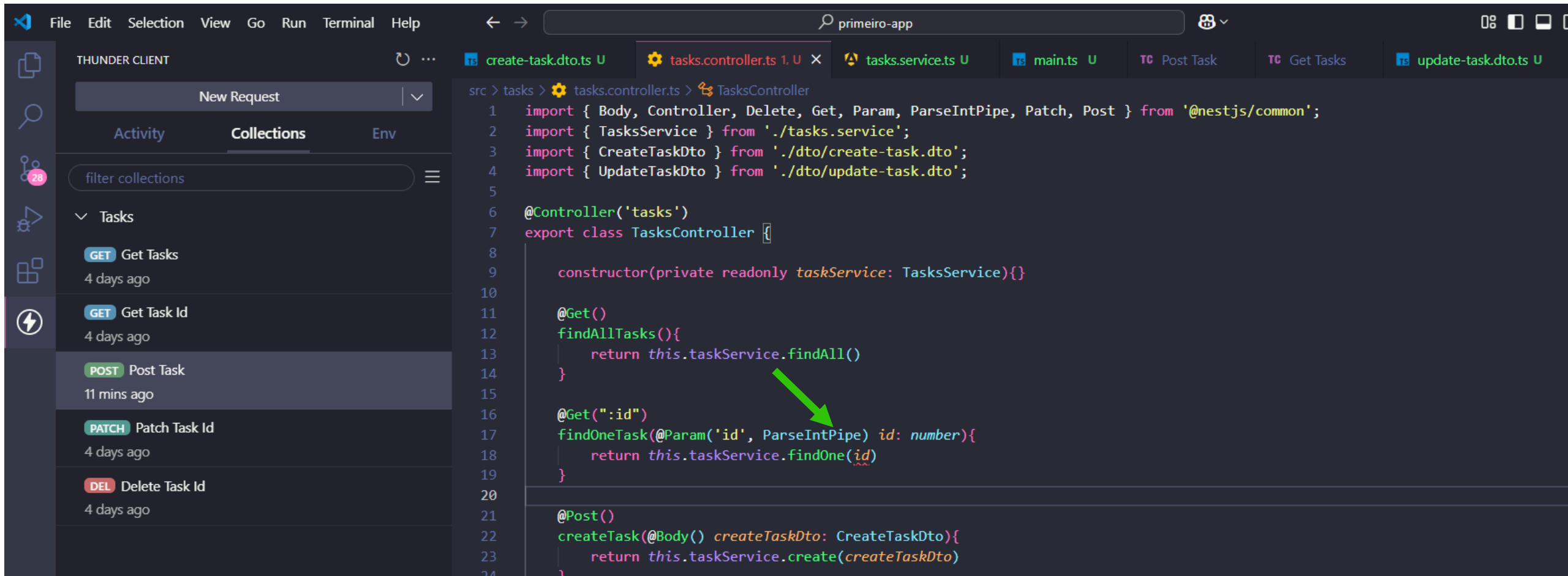


The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the project structure for 'PRIMEIRO-APP'. The 'src' directory is expanded, showing subdirectories 'app', 'tasks', 'dto', 'entities', and 'users'. The 'tasks' directory contains 'tasks.controller.ts', 'tasks.module.ts', and 'tasks.service.ts'. The 'main.ts' file is selected and open in the editor. The editor window shows the following code:

```
src > main.ts > ...
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app/app.module';
3 import { ValidationPipe } from '@nestjs/common';
4
5 /**
6  - app.module.ts : É o módulo principal do aplicativo
7  - app.controller.ts : Define as rotas e lida com as requisições
8  - app.service.ts : Contém a lógica do negócio, separado do controlador.
9  */
10
11 //Arquivo que inicia o nosso projeto
12 async function bootstrap() {
13   const app = await NestFactory.create(AppModule);
14   app.useGlobalPipes(new ValidationPipe({
15     whitelist: true, //Se o usuário enviar algum parâmetro inexistente!
16     transform: true //Para a validação utilizando ParseIntPipe
17   }));
18   await app.listen(process.env.PORT ?? 3000);
19 }
20 bootstrap();
```

A green arrow points to the 'transform: true' line in the ValidationPipe configuration.

Transformar Dados - Pipe



The image shows a screenshot of a development environment (VS Code) with a REST client on the left and a TypeScript controller on the right. The REST client is titled 'THUNDER CLIENT' and shows a list of requests under the 'Collections' tab. The requests are:

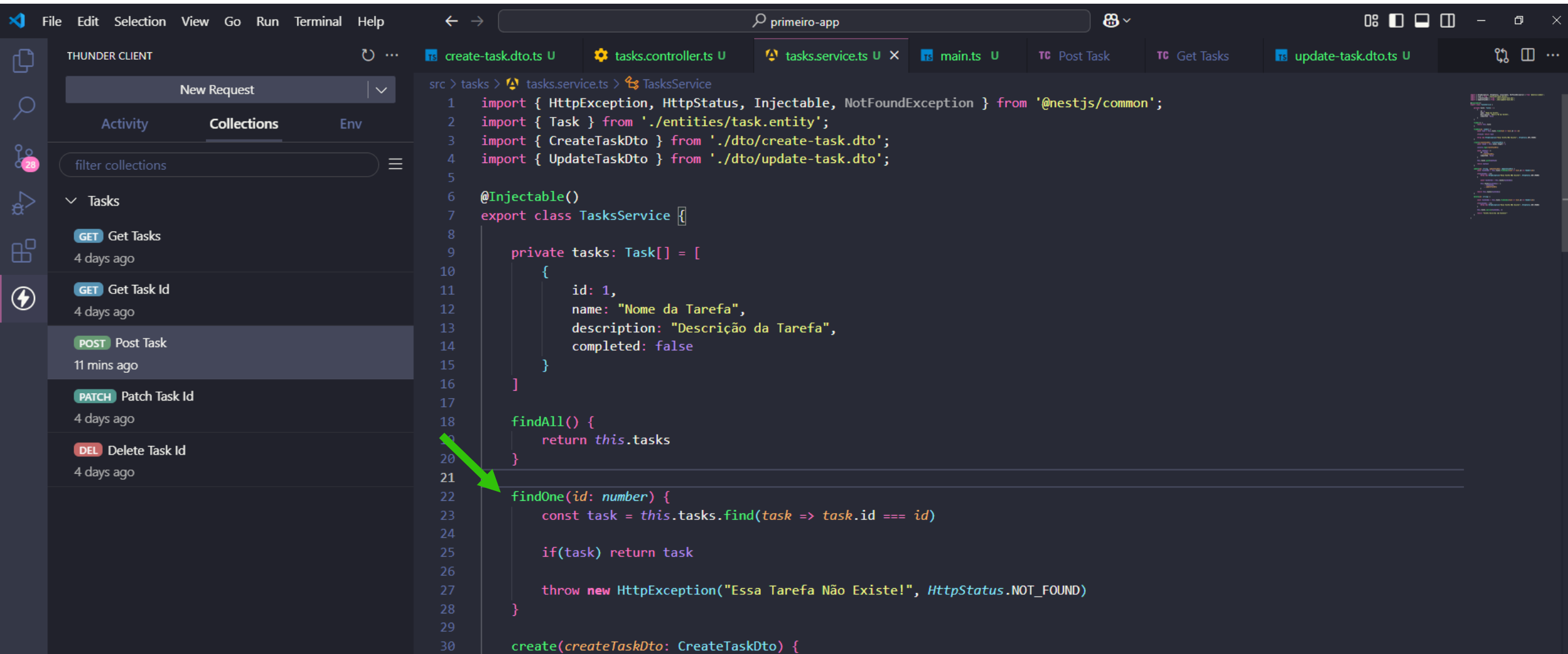
- GET Get Tasks (4 days ago)
- GET Get Task Id (4 days ago)
- POST Post Task (11 mins ago)
- PATCH Patch Task Id (4 days ago)
- DEL Delete Task Id (4 days ago)

The TypeScript controller is titled 'tasks.controller.ts' and shows the following code:

```
src > tasks > tasks.controller.ts > TasksController
1 import { Body, Controller, Delete, Get, Param, ParseIntPipe, Patch, Post } from '@nestjs/common';
2 import { TasksService } from '../tasks.service';
3 import { CreateTaskDto } from '../dto/create-task.dto';
4 import { UpdateTaskDto } from '../dto/update-task.dto';
5
6 @Controller('tasks')
7 export class TasksController {
8
9     constructor(private readonly taskService: TasksService){}
10
11     @Get()
12     findAllTasks(){
13         return this.taskService.findAll()
14     }
15
16     @Get(":id")
17     findOneTask(@Param('id', ParseIntPipe) id: number){
18         return this.taskService.findOne(id)
19     }
20
21     @Post()
22     createTask(@Body() createTaskDto: CreateTaskDto){
23         return this.taskService.create(createTaskDto)
24     }
25 }
```

A green arrow points from the `findOne` call in the `findOneTask` method to the `POST` request in the REST client.

Transformar Dados - Pipe



The image shows a screenshot of a development environment (VS Code) with a REST client on the left and a TypeScript service file on the right. The REST client, titled 'THUNDER CLIENT', shows a list of API endpoints under the 'Collections' tab. The endpoints are:

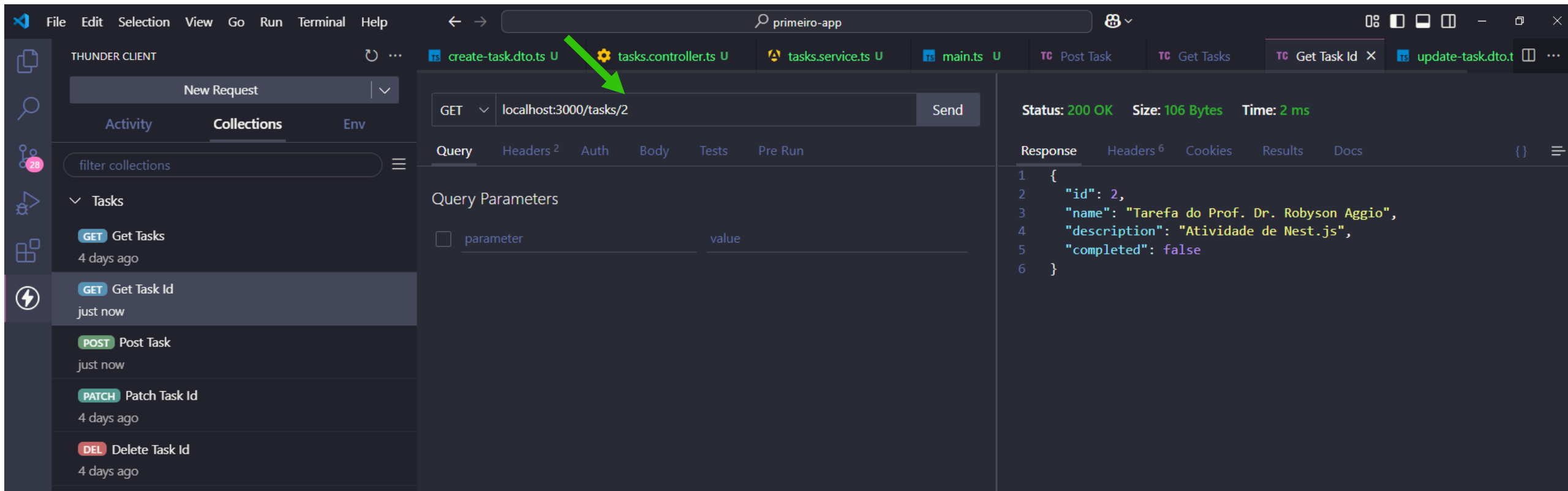
- GET** Get Tasks (4 days ago)
- GET** Get Task Id (4 days ago)
- POST** Post Task (11 mins ago)
- PATCH** Patch Task Id (4 days ago)
- DEL** Delete Task Id (4 days ago)

The right pane shows the source code for `tasks.service.ts` in the `TasksService` class. The code is as follows:

```
src > tasks > tasks.service.ts > TasksService
1  import { HttpException, HttpStatus, Injectable, NotFoundException } from '@nestjs/common';
2  import { Task } from '../entities/task.entity';
3  import { CreateTaskDto } from '../dto/create-task.dto';
4  import { UpdateTaskDto } from '../dto/update-task.dto';
5
6  @Injectable()
7  export class TasksService {
8
9      private tasks: Task[] = [
10         {
11             id: 1,
12             name: "Nome da Tarefa",
13             description: "Descrição da Tarefa",
14             completed: false
15         }
16     ]
17
18     findAll() {
19         return this.tasks
20     }
21
22     findOne(id: number) {
23         const task = this.tasks.find(task => task.id === id)
24
25         if(task) return task
26
27         throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)
28     }
29
30     create(createTaskDto: CreateTaskDto) {
```

A green arrow points from the `findOne` method in the code to the `POST` 'Post Task' endpoint in the REST client, indicating a mapping between the API call and the service method.

Transformar Dados - Pipe



The screenshot displays the Thunder Client interface with a REST client request and response. A green arrow points to the `tasks.controller.ts` file in the editor.

Request:

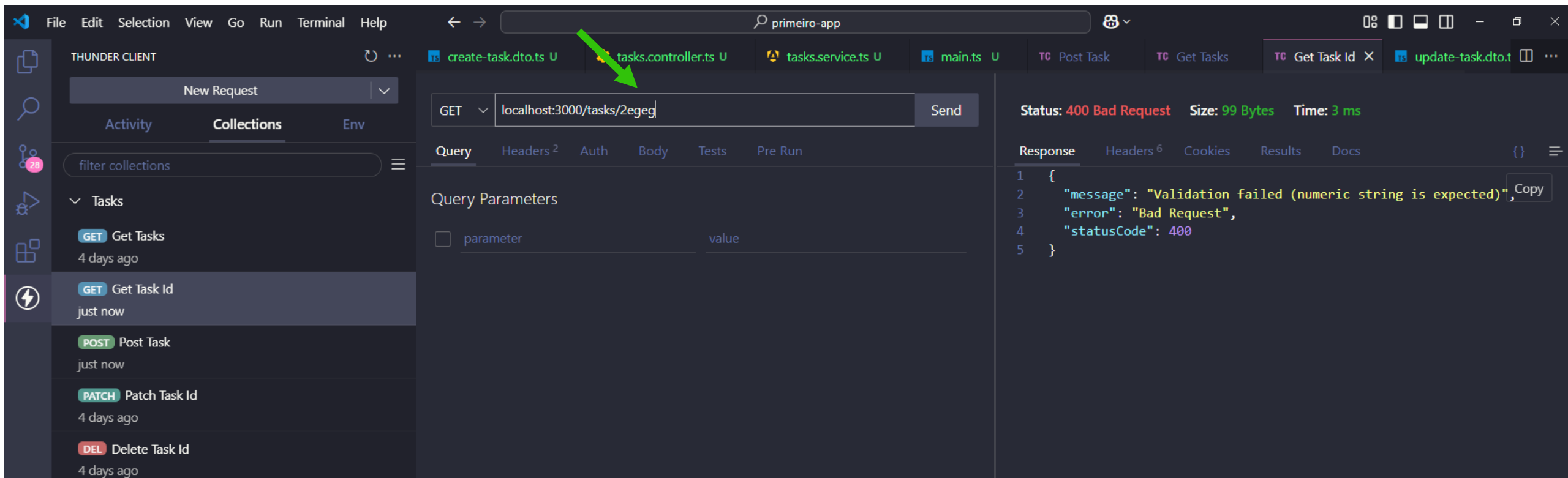
- Method: GET
- URL: `localhost:3000/tasks/2`
- Send button

Response:

Status: 200 OK Size: 106 Bytes Time: 2 ms

```
1 {
2   "id": 2,
3   "name": "Tarefa do Prof. Dr. Robyson Aggio",
4   "description": "Atividade de Nest.js",
5   "completed": false
6 }
```

Transformar Dados - Pipe



The screenshot displays the Thunder Client interface. The top bar shows the application name 'primeiro-app' and several open tabs including 'create-task.dto.ts', 'tasks.controller.ts', 'tasks.service.ts', 'main.ts', 'Post Task', 'Get Tasks', 'Get Task Id', and 'update-task.dto.t'. A green arrow points to the 'tasks.controller.ts' tab.

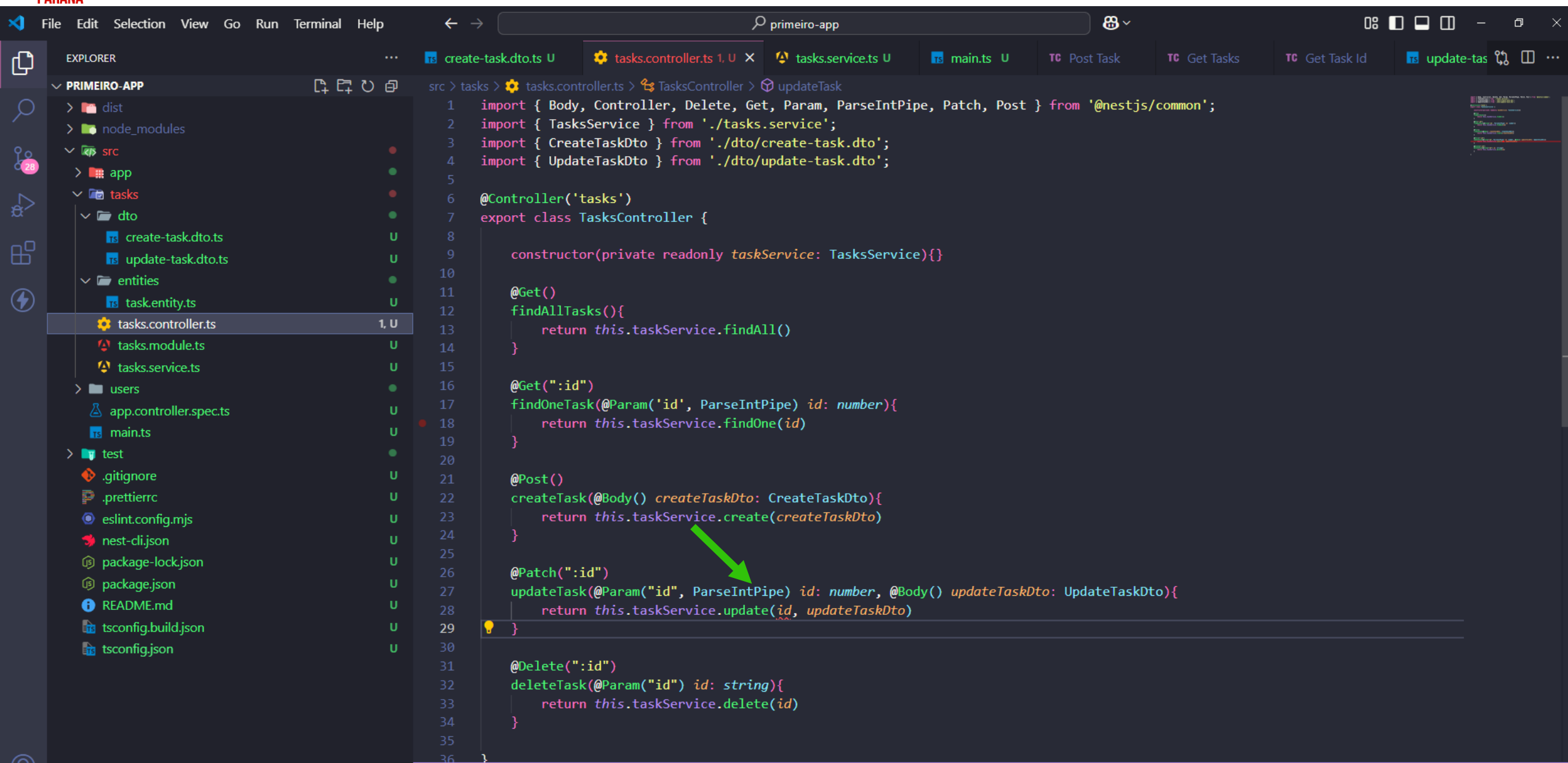
The main workspace is divided into three panels:

- Left Panel (Collections):** Shows a list of tasks under the 'Tasks' collection. The 'Get Task Id' task is selected, showing it was executed 'just now'.
- Center Panel (Request Editor):** Displays a GET request to 'localhost:3000/tasks/2egeg'. The 'Query' tab is active, showing a 'parameter' field with a 'value' input.
- Right Panel (Response):** Shows the response status '400 Bad Request' with a size of '99 Bytes' and a time of '3 ms'. The response body is a JSON object:

```
{  "message": "Validation failed (numeric string is expected)",  "error": "Bad Request",  "statusCode": 400}
```

. A 'Copy' button is visible next to the response text.

Transformar Dados - Pipe



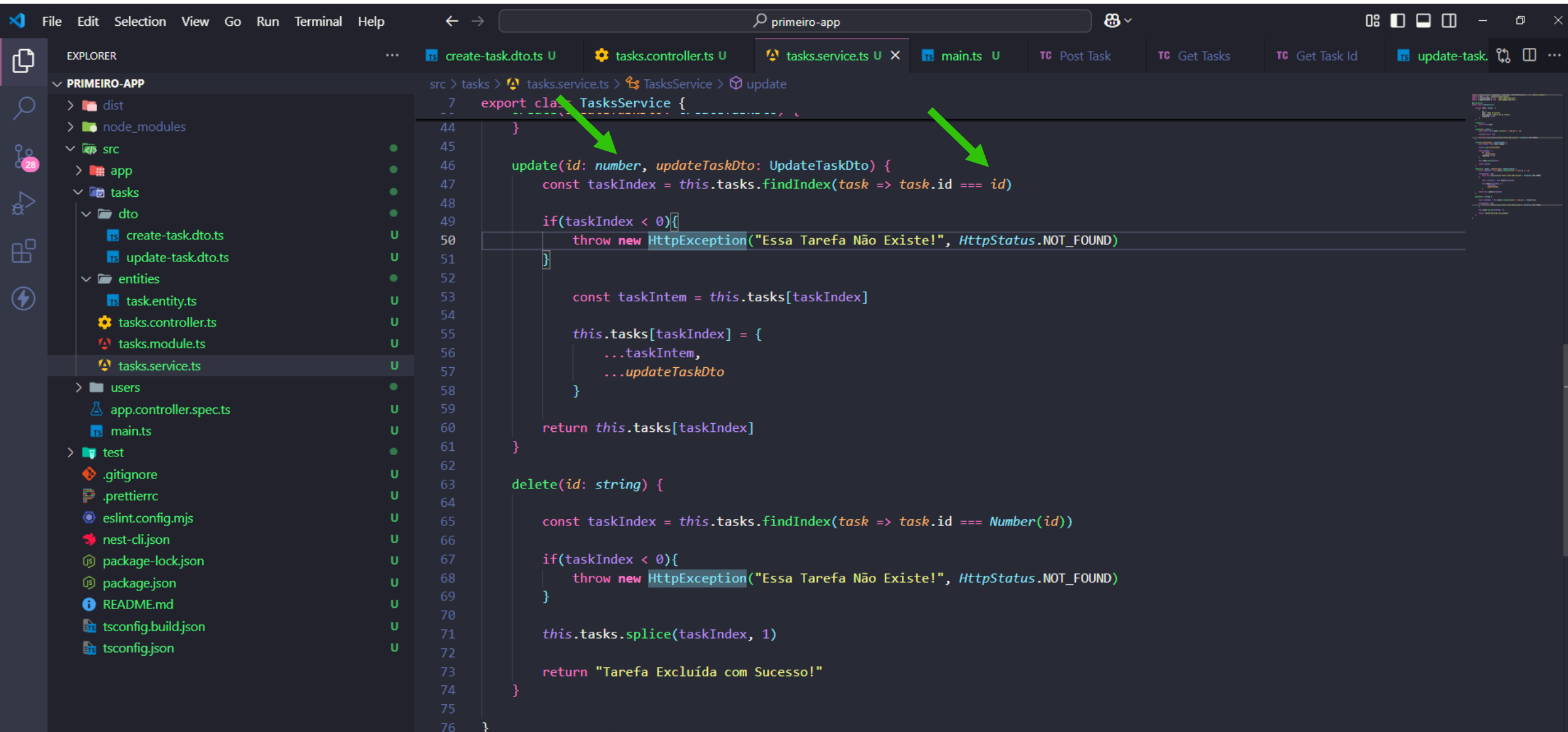
The screenshot shows a VS Code editor with a project named 'primeiro-app'. The Explorer on the left shows the file structure, including 'src/tasks/tasks.controller.ts'. The main editor displays the code for 'tasks.controller.ts', which is a NestJS controller for tasks. A green arrow points to the 'updateTask' method in the 'updateTask' decorator.

```

1  import { Body, Controller, Delete, Get, Param, ParseIntPipe, Patch, Post } from '@nestjs/common';
2  import { TaskService } from './tasks.service';
3  import { CreateTaskDto } from './dto/create-task.dto';
4  import { UpdateTaskDto } from './dto/update-task.dto';
5
6  @Controller('tasks')
7  export class TasksController {
8
9      constructor(private readonly taskService: TaskService){}
10
11     @Get()
12     findAllTasks(){
13         return this.taskService.findAll()
14     }
15
16     @Get(":id")
17     findOneTask(@Param('id', ParseIntPipe) id: number){
18         return this.taskService.findOne(id)
19     }
20
21     @Post()
22     createTask(@Body() createTaskDto: CreateTaskDto){
23         return this.taskService.create(createTaskDto)
24     }
25
26     @Patch(":id")
27     updateTask(@Param("id", ParseIntPipe) id: number, @Body() updateTaskDto: UpdateTaskDto){
28         return this.taskService.update(id, updateTaskDto)
29     }
30
31     @Delete(":id")
32     deleteTask(@Param("id") id: string){
33         return this.taskService.delete(id)
34     }
35 }

```

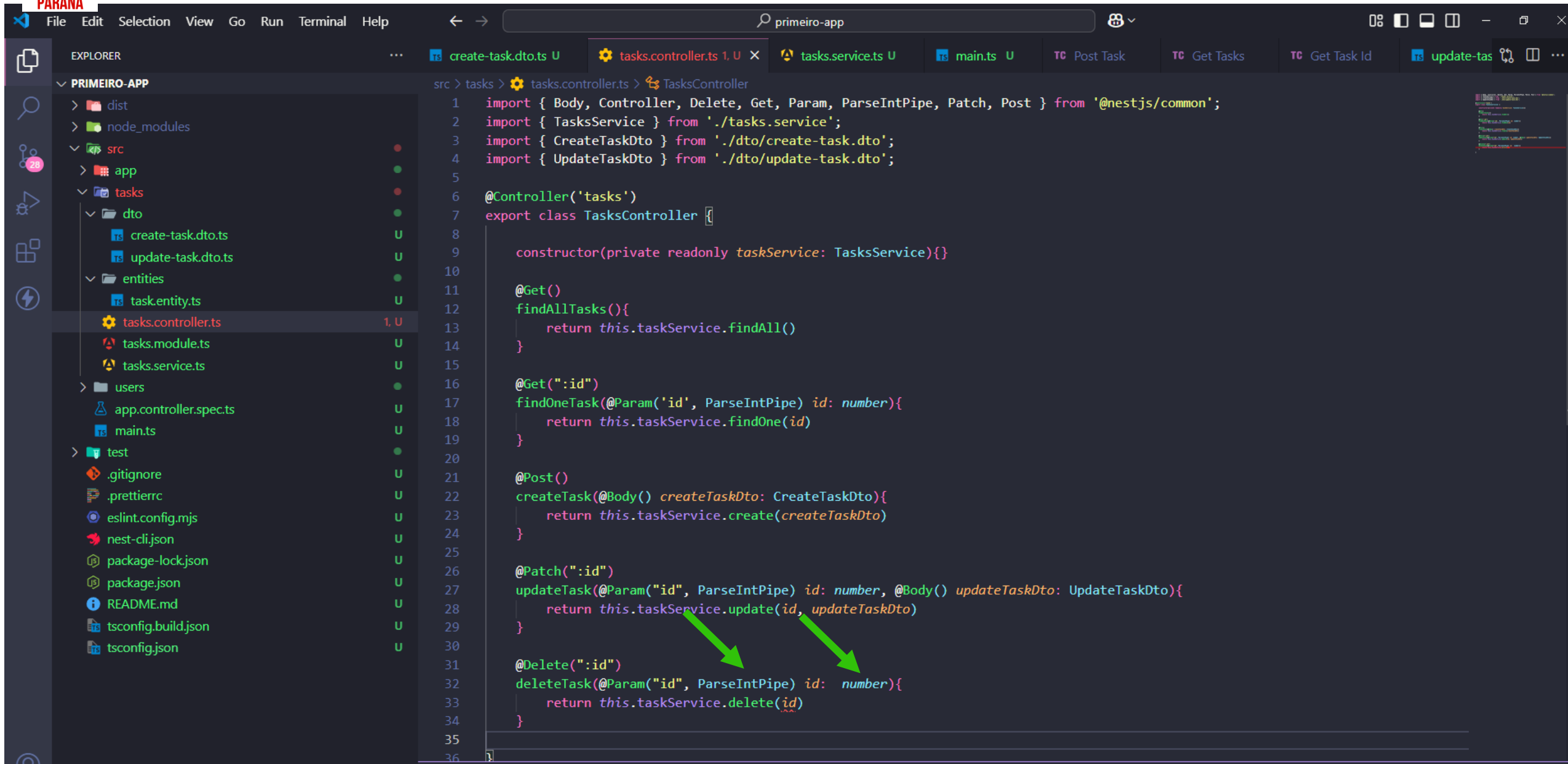
Transformar Dados - Pipe



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor in the center. The file explorer shows the project structure for 'PRIMEIRO-APP', including folders like 'dist', 'node_modules', 'src', 'tasks', 'dto', 'entities', 'users', and 'test'. The code editor displays the 'tasks.service.ts' file, specifically the 'update' method. Two green arrows point to the 'update' method signature and the 'throw new' statement, highlighting the use of the pipe operator.

```
export class TaskService {  
  // ...  
  update(id: number, updateTaskDto: UpdateTaskDto) {  
    const taskIndex = this.tasks.findIndex(task => task.id === id)  
    if(taskIndex < 0){  
      throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)  
    }  
    const taskIntem = this.tasks[taskIndex]  
    this.tasks[taskIndex] = {  
      ...taskIntem,  
      ...updateTaskDto  
    }  
    return this.tasks[taskIndex]  
  }  
  delete(id: string) {  
    const taskIndex = this.tasks.findIndex(task => task.id === Number(id))  
    if(taskIndex < 0){  
      throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)  
    }  
    this.tasks.splice(taskIndex, 1)  
    return "Tarefa Excluída com Sucesso!"  
  }  
}
```

Transformar Dados - Pipe

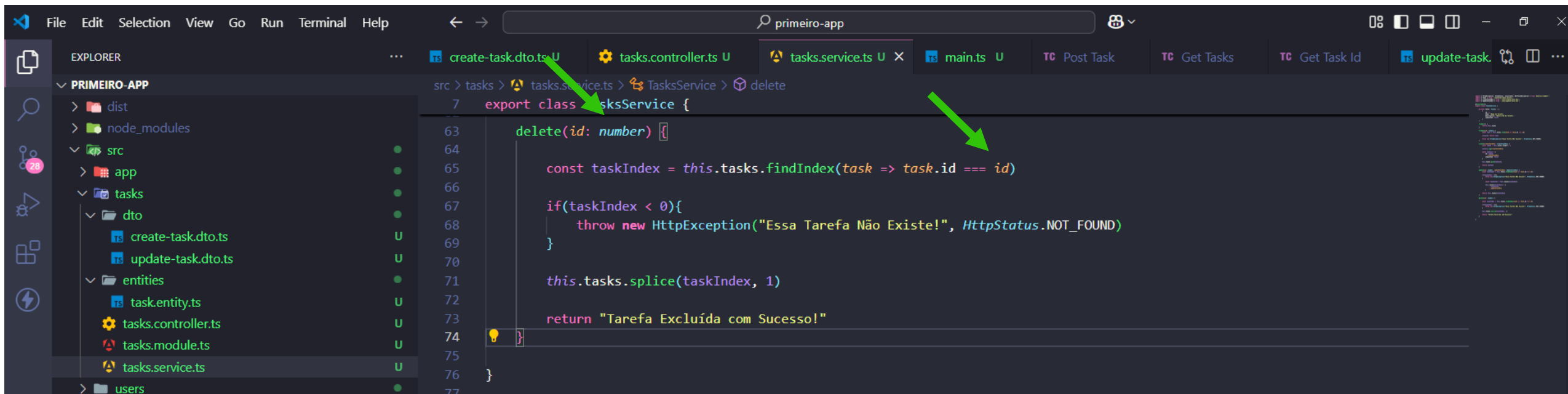


The screenshot shows a Visual Studio Code editor with the following components:

- EXPLORER (Left):** Displays the file structure of the 'PRIMEIRO-APP' project. The 'tasks' folder is expanded, showing 'dto' and 'entities' subfolders. The file 'tasks.controller.ts' is selected and highlighted in red.
- Editor (Center):** Displays the code for 'tasks.controller.ts'. The code implements a NestJS controller for tasks, using data transformation pipes. Two green arrows point from the 'id' parameter in the 'updateTask' and 'deleteTask' methods to the 'ParseIntPipe' pipe used in their decorators.
- Terminal/Output (Bottom):** Shows the command prompt with the path 'src > tasks > tasks.controller.ts'.

```
1 import { Body, Controller, Delete, Get, Param, ParseIntPipe, Patch, Post } from '@nestjs/common';
2 import { TasksService } from '../tasks.service';
3 import { CreateTaskDto } from '../dto/create-task.dto';
4 import { UpdateTaskDto } from '../dto/update-task.dto';
5
6 @Controller('tasks')
7 export class TasksController {
8
9   constructor(private readonly taskService: TasksService){
10
11   @Get()
12   findAllTasks(){
13     return this.taskService.findAll()
14   }
15
16   @Get(":id")
17   findOneTask(@Param('id', ParseIntPipe) id: number){
18     return this.taskService.findOne(id)
19   }
20
21   @Post()
22   createTask(@Body() createTaskDto: CreateTaskDto){
23     return this.taskService.create(createTaskDto)
24   }
25
26   @Patch(":id")
27   updateTask(@Param("id", ParseIntPipe) id: number, @Body() updateTaskDto: UpdateTaskDto){
28     return this.taskService.update(id, updateTaskDto)
29   }
30
31   @Delete(":id")
32   deleteTask(@Param("id", ParseIntPipe) id: number){
33     return this.taskService.delete(id)
34   }
35
36 }
```

Transformar Dados - Pipe



The screenshot shows the Visual Studio Code editor with a project named "PRIMEIRO-APP". The Explorer sidebar on the left shows the file structure, including a "tasks" folder with a "dto" subfolder. The main editor area displays the "tasks.service.ts" file, which contains the implementation of the "delete" method. The code is as follows:

```
src > tasks > tasks.service.ts > TasksService > delete
7 export class TasksService {
63   delete(id: number) {
64
65     const taskIndex = this.tasks.findIndex(task => task.id === id)
66
67     if(taskIndex < 0){
68       throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)
69     }
70
71     this.tasks.splice(taskIndex, 1)
72
73     return "Tarefa Excluída com Sucesso!"
74   }
75
76 }
77
```

Two green arrows are present: one pointing from the "tasks.service.ts" tab in the top editor bar to the "TasksService" class definition, and another pointing from the "delete" method signature to its implementation body.



Search

COURSES

ENTERPRISE

NEW DEVTOOLS

DEPLOY WITH MAU



INTRODUCTION

OVERVIEW

FUNDAMENTALS

TECHNIQUES

Configuration

Database

Mongo

Validation

Caching

Serialization

Versioning

Task scheduling

Queues

Logging

Cookies

Events

Compression

File upload

Streaming files

HTTP module

Session

Model-View-Controller

Performance (Fastify)

Server-Sent Events

SECURITY

GRAPHQL

WEBSOCKETS

MICROSERVICES

NEW DEPLOYMENT

Validation

It is best practice to validate the correctness of any data sent into a web application. To automatically validate incoming requests, Nest provides several pipes available right out-of-the-box:

- ValidationPipe
- ParseIntPipe
- ParseBoolPipe
- ParseArrayPipe
- ParseUUIDPipe

The `ValidationPipe` makes use of the powerful `class-validator` package and its declarative validation decorators. The `ValidationPipe` provides a convenient approach to enforce validation rules for all incoming client payloads, where the specific rules are declared with simple annotations in local class/DTO declarations in each module.

Overview

In the `Pipes` chapter, we went through the process of building simple pipes and binding them to controllers, methods or to the global app to demonstrate how the process works. Be sure to review that chapter to best understand the topics of this chapter. Here, we'll focus on various **real world** use cases of the `ValidationPipe`, and show how to use some of its advanced customization features.

Using the built-in ValidationPipe

To begin using it, we first install the required dependency.

```
$ npm i --save class-validator class-transformer
```

HINT

The `ValidationPipe` is exported from the `@nestjs/common` package.



Validation

Overview

Using the built-in
ValidationPipe

Auto-validation

Disable detailed errors

Stripping properties

Transform payload objects

Explicit conversion

Mapped types

Parsing and validating arrays

WebSockets and

Microservices

Learn more



Design and
Development tips in
your inbox. Every
weekday.

Atividade

1) Dentro de cada camada: Guests, Users e Teachers, implemente o ValidationPipe

2) Dentro de cada camada no update*Dto utilize o PartialType

3) Em cada camada utilize ParseIntPipe, nos verbos: Get(), Patch(), Delete(), onde passa o parâmetro id

4) Em nosso desenvolvimento em sala, iremos criar uma camada de usuário e faremos a associação com as tarefas (1 (usuário) → n (tarefas)).

**Crie um projeto do zero com 2 camadas (em breve serão associadas: 1 → n).
Implemente todos os códigos até chegar nesse slide.**

Obs: Na próxima aula, iniciaremos o estudo do Prisma ORM.

O Prisma ORM é uma ferramenta de código aberto que permite definir modelos e interagir com bancos de dados.