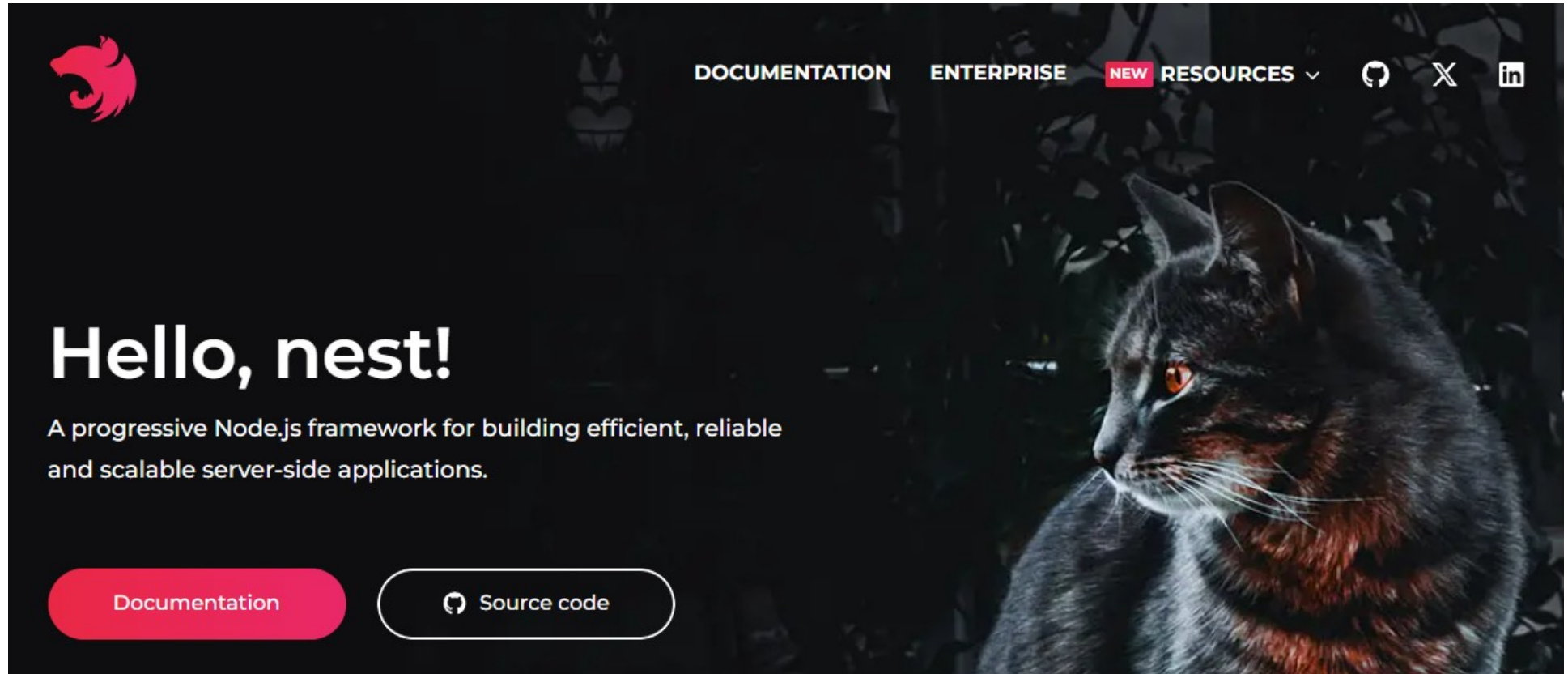


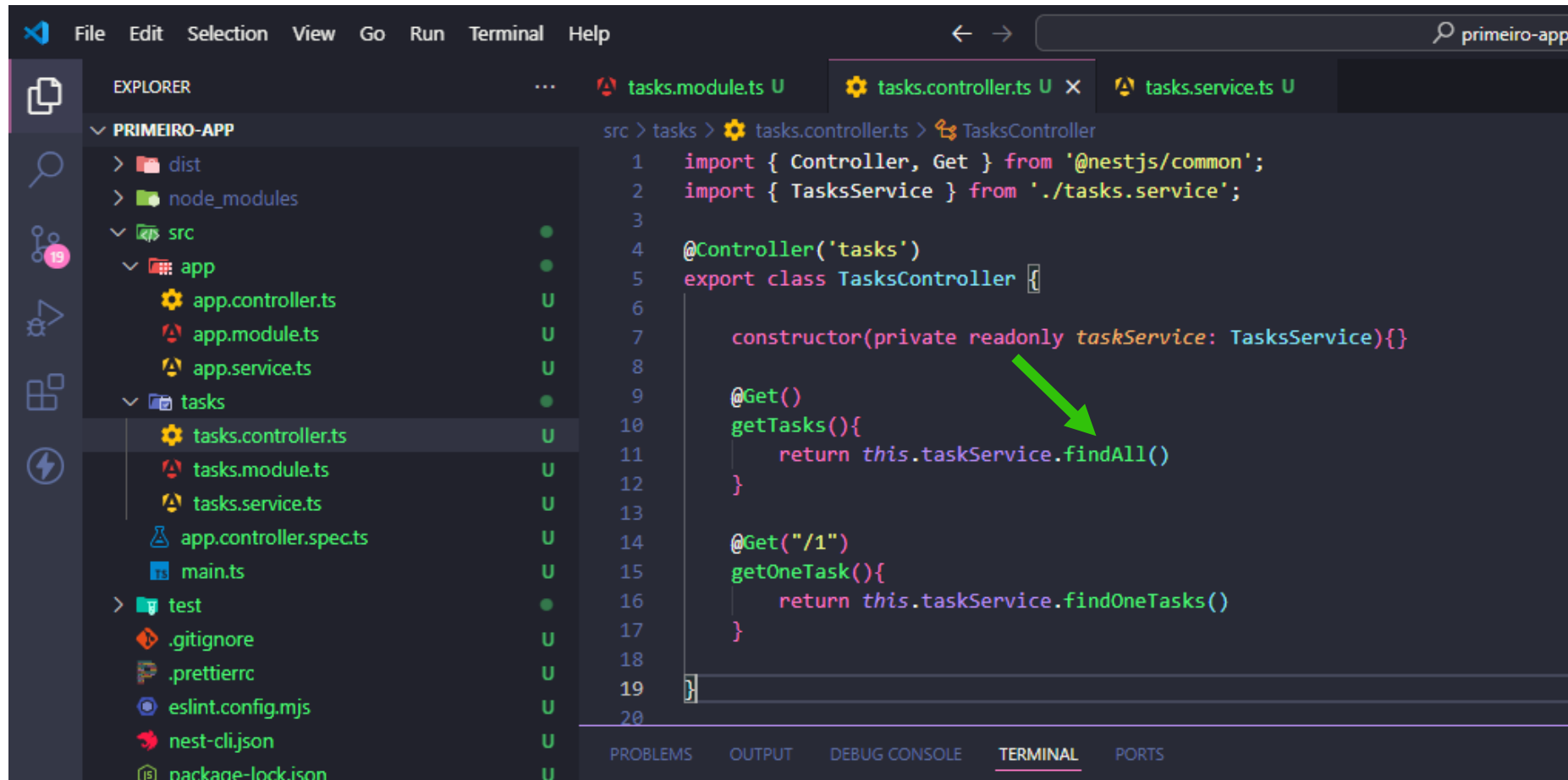
Nest.js



Prof. Dr. José Robyson Aggio Molinari

www.aggioirati.com.br

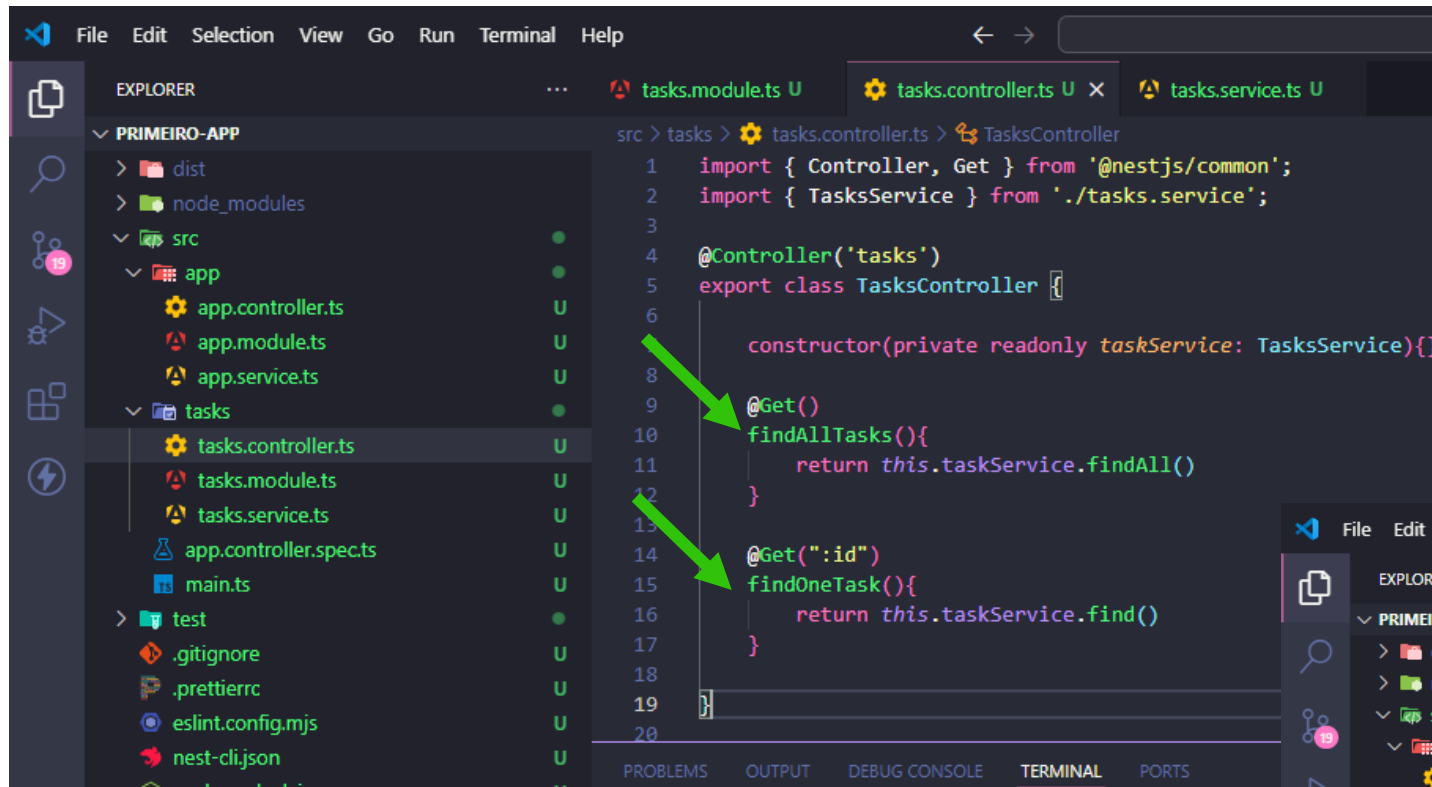
Rotas com parâmetros



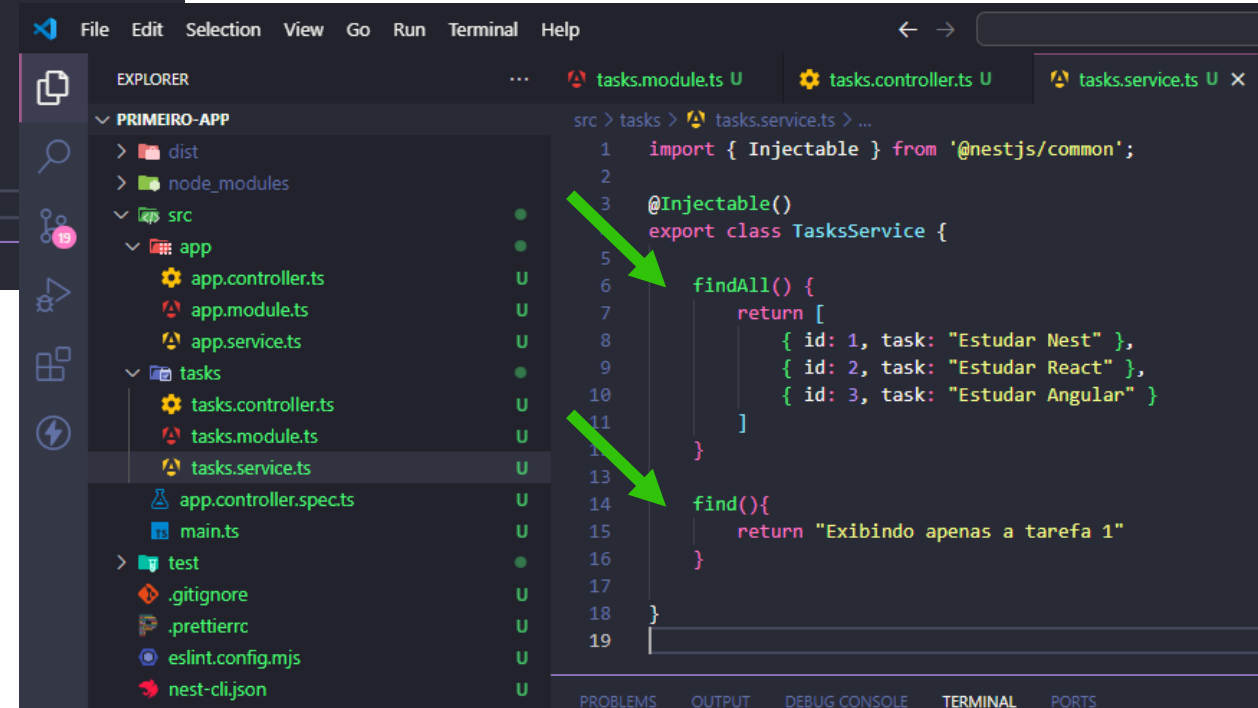
The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the project structure for 'PRIMEIRO-APP'. The 'tasks' folder is expanded, showing 'tasks.controller.ts' selected. The main editor area displays the code for 'tasks.controller.ts'. The code imports 'Controller' and 'Get' from '@nestjs/common' and 'TasksService' from './tasks.service'. It defines a 'TasksController' class with a constructor that takes 'taskService' as a parameter. Two methods are shown: 'getTasks()' which calls 'this.taskService.findAll()', and 'getOneTask()' which calls 'this.taskService.findOneTasks()'. A green arrow points to the 'taskService' parameter in the constructor.

```
1 import { Controller, Get } from '@nestjs/common';
2 import { TasksService } from './tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7     constructor(private readonly taskService: TasksService){}
8
9     @Get()
10    getTasks(){
11        return this.taskService.findAll()
12    }
13
14    @Get("/1")
15    getOneTask(){
16        return this.taskService.findOneTasks()
17    }
18
19 }
20
```

Rotas com parâmetros



```
1 import { Controller, Get } from '@nestjs/common';
2 import { TasksService } from './tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7     constructor(private readonly taskService: TasksService){}
8
9     @Get()
10    findAllTasks(){
11        return this.taskService.findAll()
12    }
13
14    @Get(":id")
15    findOneTask(){
16        return this.taskService.find()
17    }
18
19 }
20
```



```
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class TasksService {
5
6     findAll() {
7         return [
8             { id: 1, task: "Estudar Nest" },
9             { id: 2, task: "Estudar React" },
10            { id: 3, task: "Estudar Angular" }
11        ]
12    }
13
14    find(){
15        return "Exibindo apenas a tarefa 1"
16    }
17
18 }
19
```

Rotas com parâmetros

The screenshot shows a Visual Studio Code editor with a NestJS application. The Explorer on the left displays the project structure, including the 'tasks' directory. The main editor shows the 'tasks.controller.ts' file. The code defines a 'TasksController' with two routes: 'findAllTasks()' and 'findOneTask()'. The 'findOneTask()' route is decorated with '@Get(':id')' and '@Param()'. A dropdown menu is open for the '@Param()' decorator, showing various options. A green arrow points to the 'Param' option, which is highlighted in the dropdown.

```

1 import { Controller, Get } from '@nestjs/common';
2 import { TasksService } from '../tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7   constructor(private readonly taskService: TasksService){}
8
9   @Get()
10  findAllTasks(){
11    return this.taskService.findAll()
12  }
13
14  @Get(":id")
15  findOneTask(@Param()){
16    return this.taskService.findOneById()
17  }
18 }

```

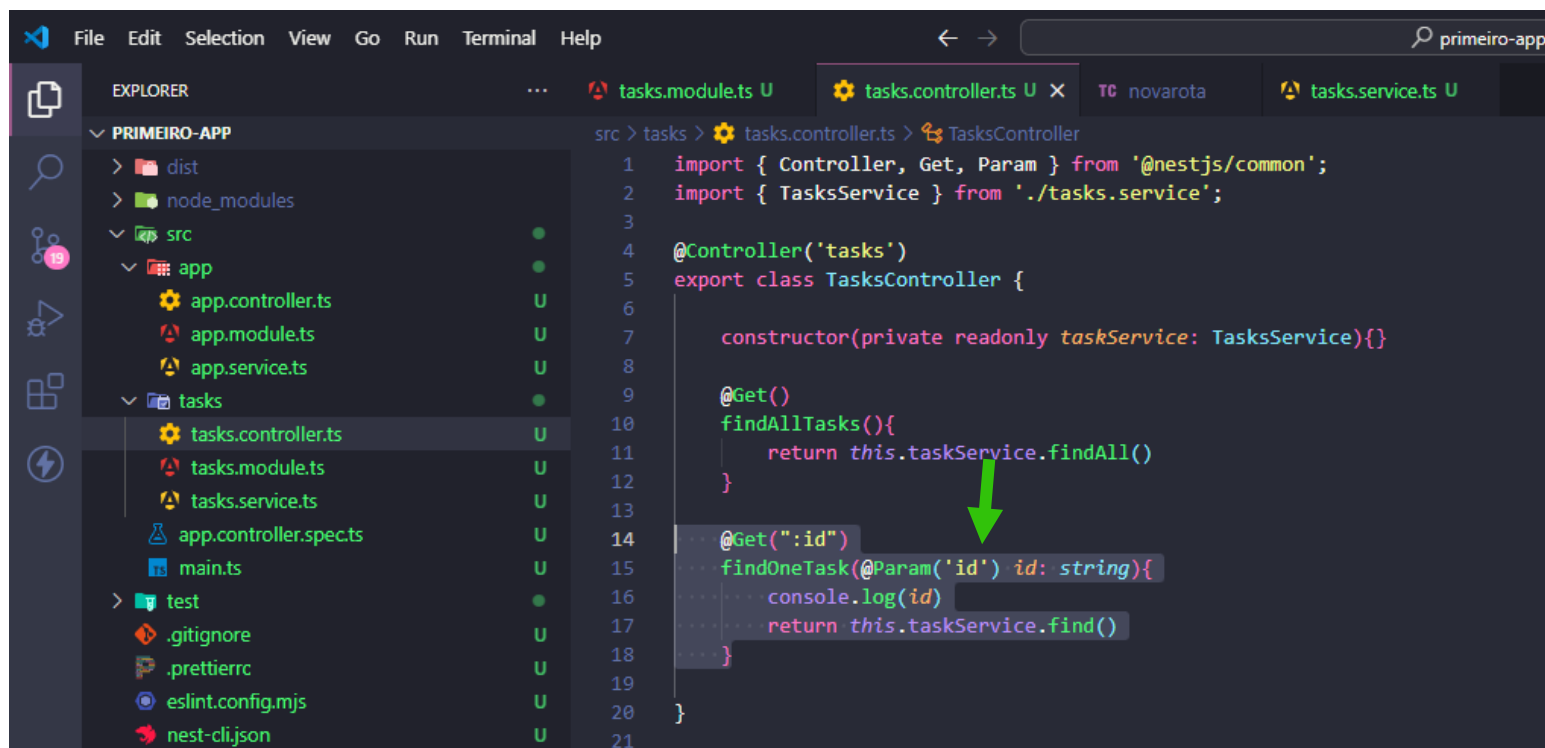
Dropdown menu options:

- Param @nestjs/common
- ParamsTokenFactory @nestjs/core/pipes
- ParseArrayPipe @nestjs/common
- ParseBoolPipe @nestjs/common
- ParseDatePipe @nestjs/common

Rotas com parâmetros

The screenshot shows the implementation of the `findOneTask` method in `tasks.controller.ts`. The code imports `Controller`, `Get`, and `Param` from `@nestjs/common`, and `TaskService` from `./tasks.service`. The `TasksController` class has a constructor for `TaskService` and two methods: `findAllTasks()` which returns `this.taskService.findAll()`, and `findOneTask(@Param() params: any)` which logs the `params` and returns `this.taskService.find(params)`. The bottom panel shows the Thunder Client interface with a GET request to `localhost:3000/tasks/4`, and the response in the terminal showing an array of tasks with IDs 1, 2, 3, and 4.

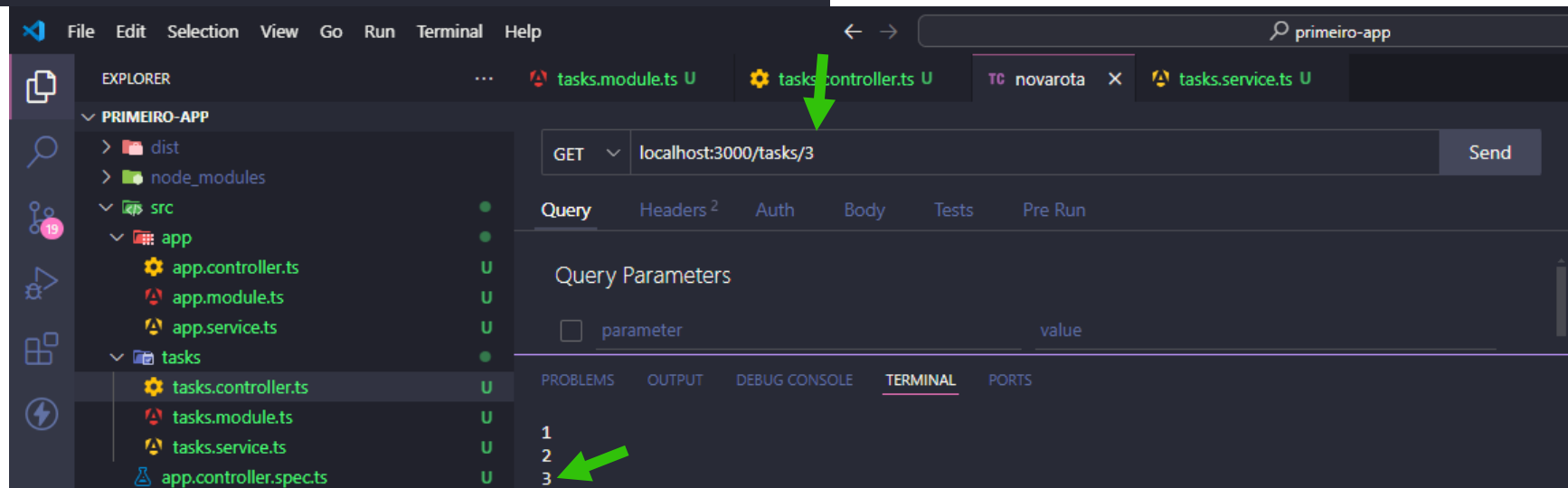
Rotas com parâmetros



```

1  import { Controller, Get, Param } from '@nestjs/common';
2  import { TasksService } from '../tasks.service';
3
4  @Controller('tasks')
5  export class TasksController {
6
7      constructor(private readonly taskService: TasksService){}
8
9      @Get()
10     findAllTasks(){
11         return this.taskService.findAll()
12     }
13
14     ... @Get('/:id')
15     ... findOneTask(@Param('id') id: string){
16         ... console.log(id)
17         ... return this.taskService.find()
18         ... }
19
20     }
21

```



GET localhost:3000/tasks/3

Query Parameters

parameter	value

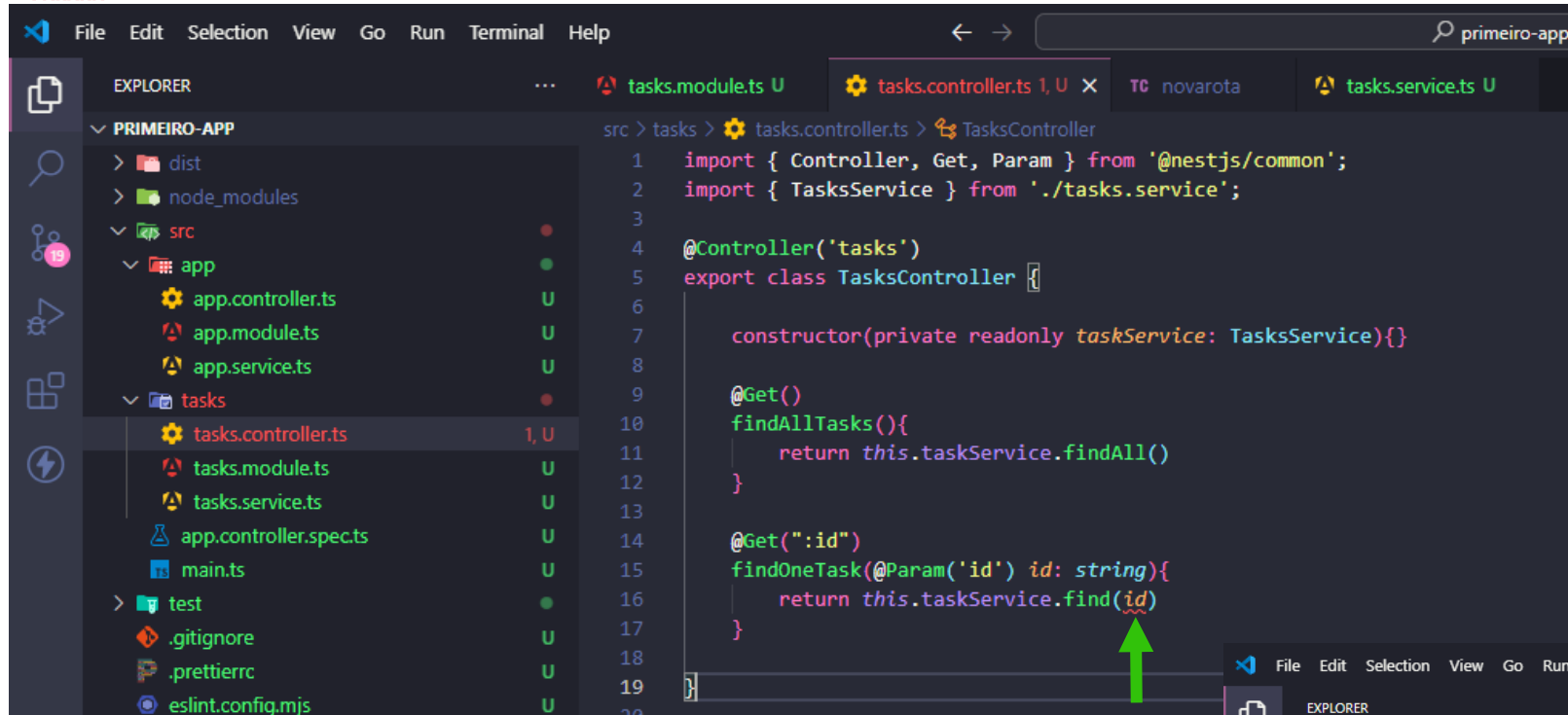
PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

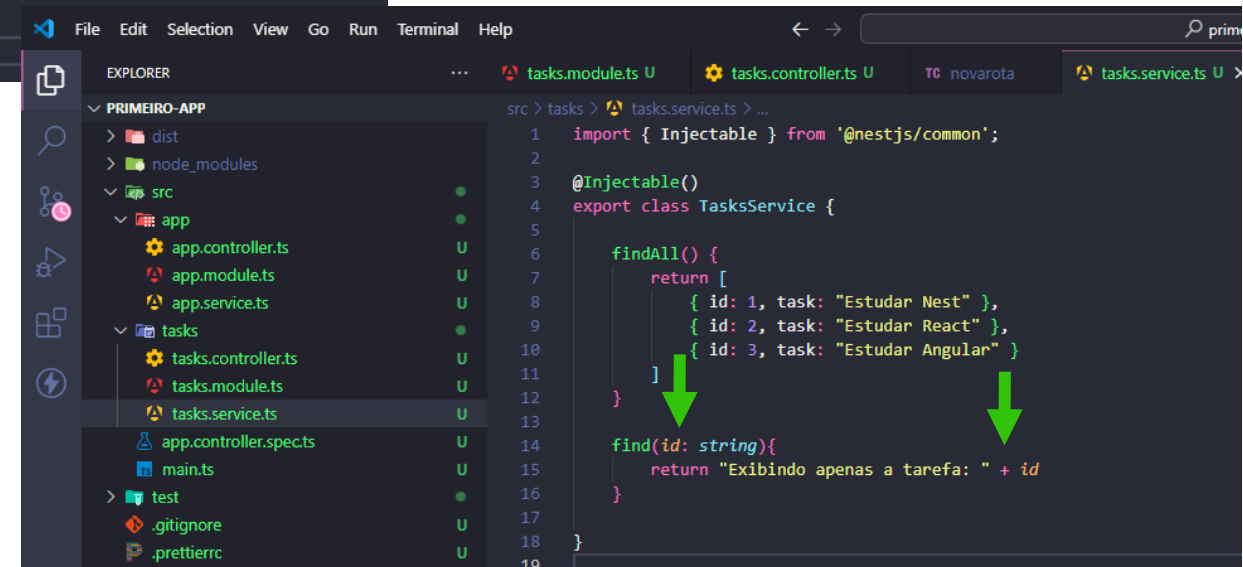
1
2
3

```

Rotas com parâmetros

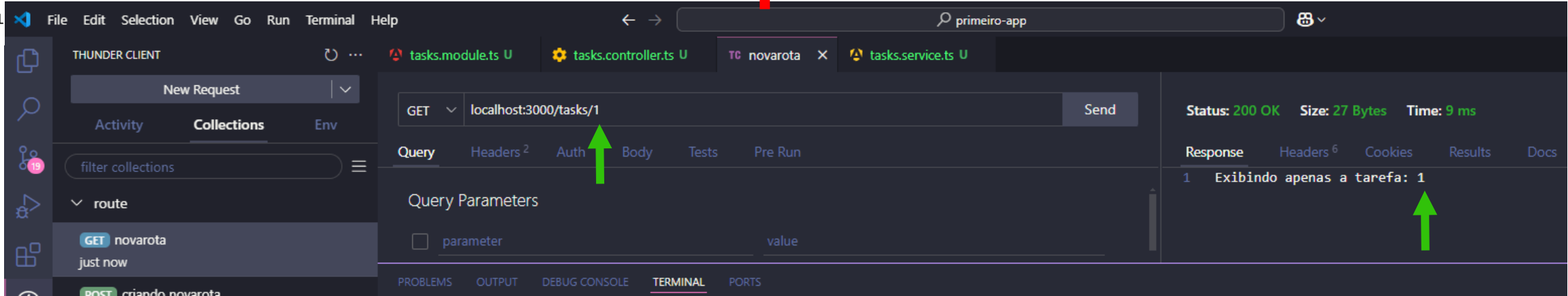


```
1 import { Controller, Get, Param } from '@nestjs/common';
2 import { TasksService } from './tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7   constructor(private readonly taskService: TasksService){}
8
9   @Get()
10  findAllTasks(){
11    return this.taskService.findAll()
12  }
13
14  @Get(":id")
15  findOneTask(@Param('id') id: string){
16    return this.taskService.find(id)
17  }
18
19 }
```



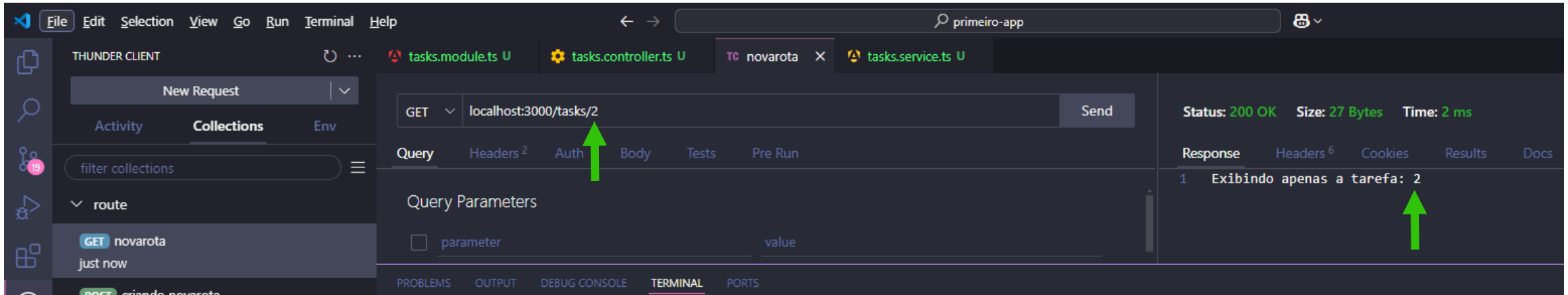
```
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class TasksService {
5
6   findAll() {
7     return [
8       { id: 1, task: "Estudar Nest" },
9       { id: 2, task: "Estudar React" },
10      { id: 3, task: "Estudar Angular" }
11    ]
12  }
13
14  find(id: string){
15    return "Exibindo apenas a tarefa: " + id
16  }
17
18 }
19 }
```

Rotas com parâmetros



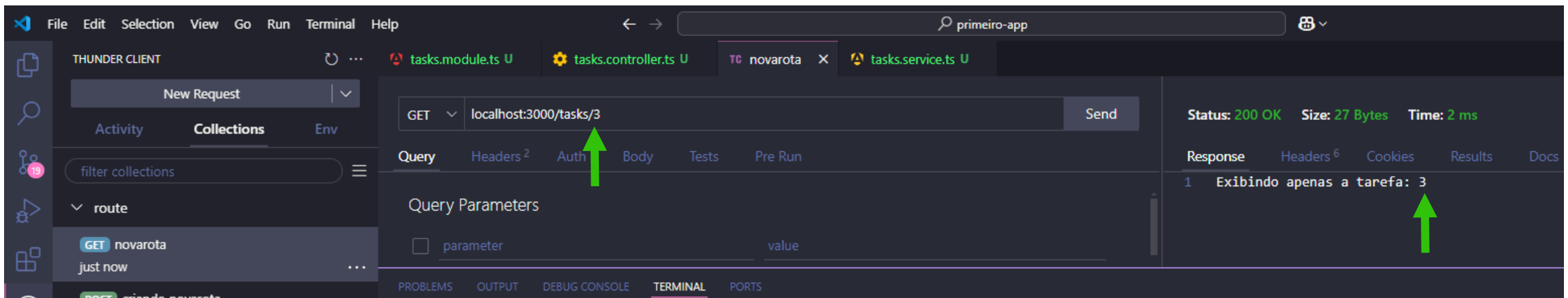
Thunder Client interface showing a GET request to `localhost:3000/tasks/1`. The response is `Exibindo apenas a tarefa: 1`. A green arrow points to the `Auth` tab, and another green arrow points to the response text.

Response	Headers	Cookies	Results	Docs
1			Exibindo apenas a tarefa: 1	



Thunder Client interface showing a GET request to `localhost:3000/tasks/2`. The response is `Exibindo apenas a tarefa: 2`. A green arrow points to the `Auth` tab, and another green arrow points to the response text.

Response	Headers	Cookies	Results	Docs
1			Exibindo apenas a tarefa: 2	



Thunder Client interface showing a GET request to `localhost:3000/tasks/3`. The response is `Exibindo apenas a tarefa: 3`. A green arrow points to the `Auth` tab, and another green arrow points to the response text.

Response	Headers	Cookies	Results	Docs
1			Exibindo apenas a tarefa: 3	

Rotas com parâmetros

The image displays two screenshots of a Visual Studio Code editor showing the implementation of REST routes in a NestJS application. The top screenshot shows the initial setup of the `TasksController` with a `findAllTasks` method using `@Query`. The bottom screenshot shows the updated controller with query parameters and a console log.

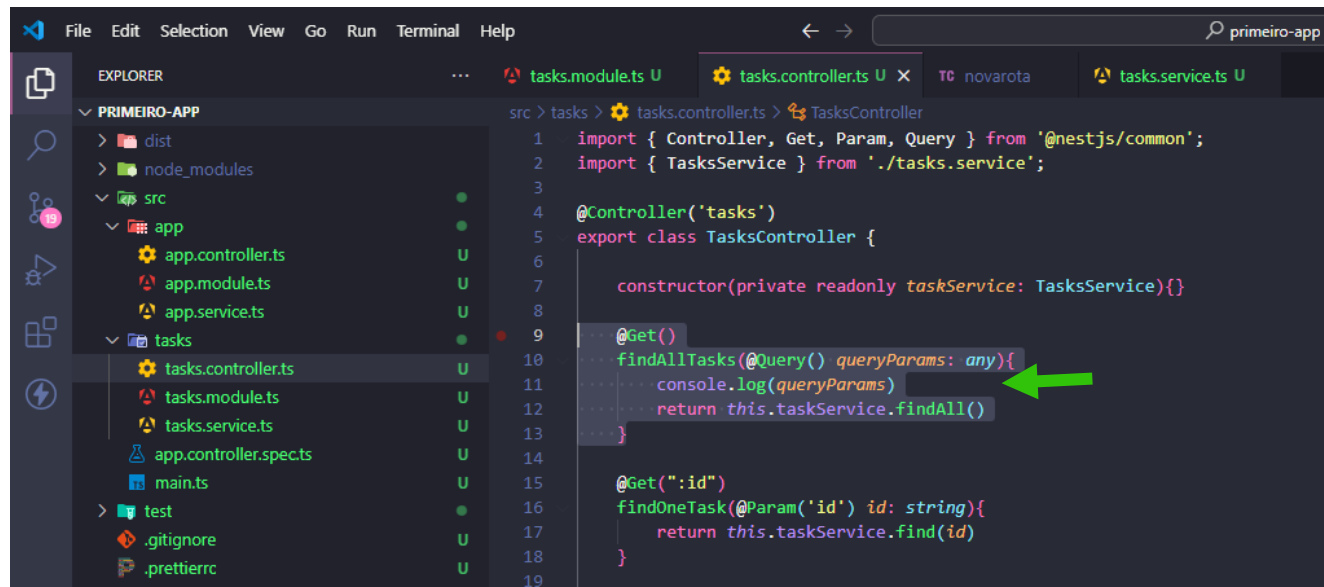
Top Screenshot: The `tasks.controller.ts` file is open, showing the `findAllTasks` method. A green arrow points to the `@Query` decorator.

```
1 import { Controller, Get, Param } from '@nestjs/common';
2 import { TasksService } from '../tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7   constructor(private readonly taskService: TasksService){}
8
9   @Get()
10  findAllTasks(@Query){
11    return this.taskService.findAll();
12  }
13 }
```

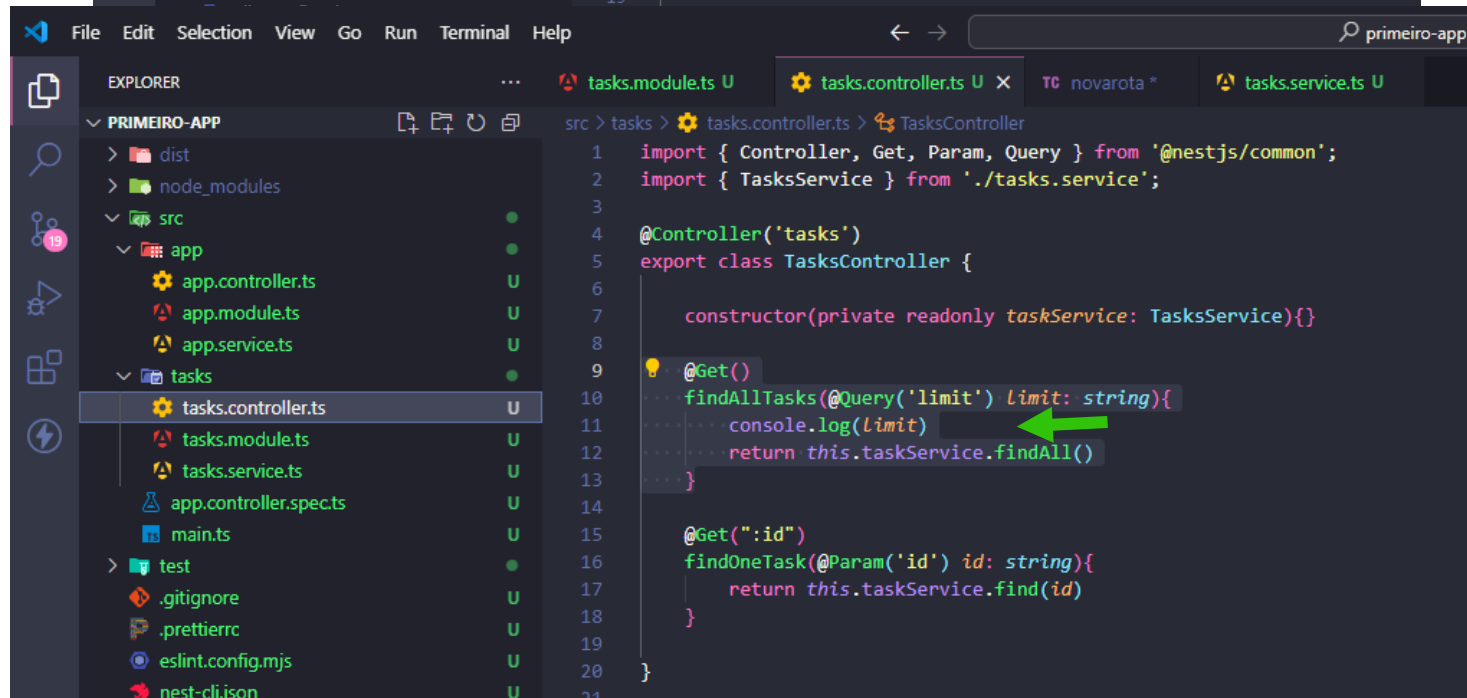
Bottom Screenshot: The `tasks.controller.ts` file is open, showing the updated `findAllTasks` method. A green arrow points to the `queryParams` parameter in the `@Query` decorator.

```
1 import { Controller, Get, Param, Query } from '@nestjs/common';
2 import { TasksService } from '../tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7   constructor(private readonly taskService: TasksService){}
8
9   @Get()
10  findAllTasks(@Query() queryParams: any){
11    console.log(queryParams);
12    return this.taskService.findAll();
13  }
14
15  @Get(':id')
16  findOneTask(@Param('id') id: string){
17    return this.taskService.find(id);
18  }
19
20 }
21 }
```

Rotas com parâmetros

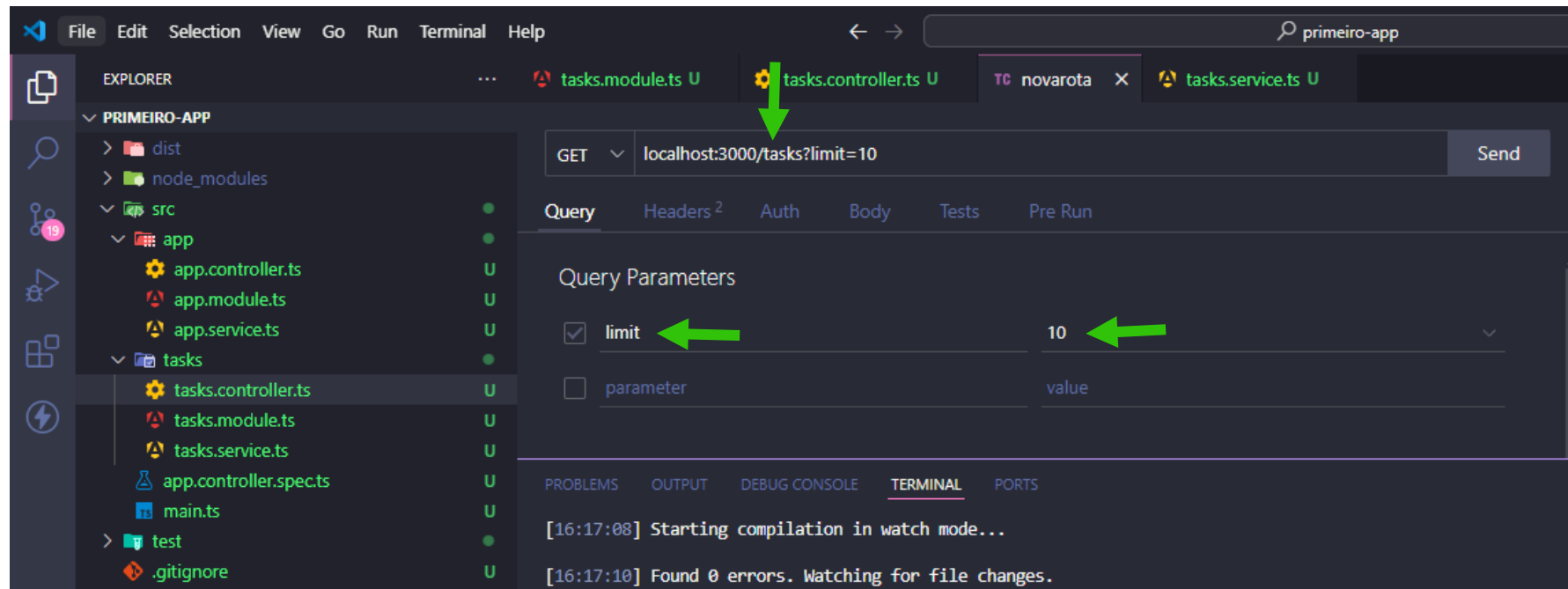


```
1 import { Controller, Get, Param, Query } from '@nestjs/common';
2 import { TaskService } from './tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7     constructor(private readonly taskService: TaskService){}
8
9     @Get()
10     findAllTasks(@Query() queryParameters: any){
11         console.log(queryParameters);
12         return this.taskService.findAll();
13     }
14
15     @Get('/:id')
16     findOneTask(@Param('id') id: string){
17         return this.taskService.find(id);
18     }
19 }
```



```
1 import { Controller, Get, Param, Query } from '@nestjs/common';
2 import { TaskService } from './tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7     constructor(private readonly taskService: TaskService){}
8
9     @Get()
10     findAllTasks(@Query('limit') limit: string){
11         console.log(limit);
12         return this.taskService.findAll();
13     }
14
15     @Get('/:id')
16     findOneTask(@Param('id') id: string){
17         return this.taskService.find(id);
18     }
19 }
20
21 }
```

Rotas com parâmetros



VS Code interface showing the Explorer view on the left with the file structure of 'PRIMEIRO-APP'. The REST client interface is open, showing a GET request to `localhost:3000/tasks?limit=10`. The Query Parameters section shows `limit` set to `10`. The terminal output shows the compilation process.

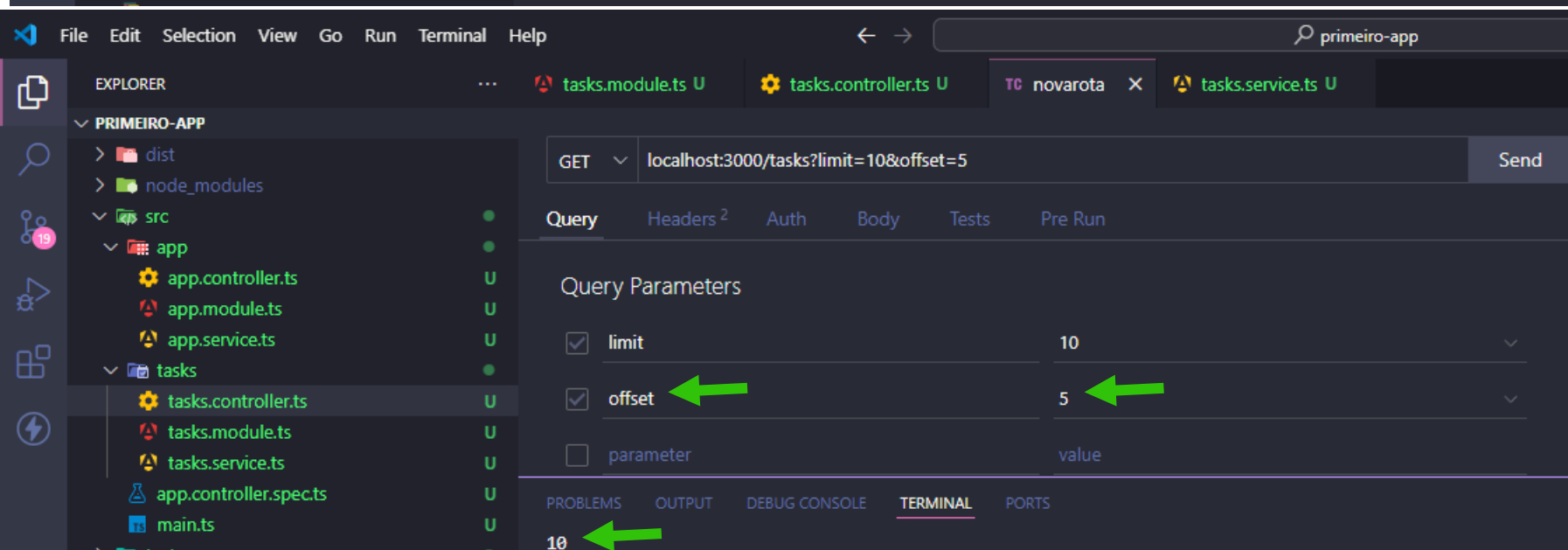
```
GET localhost:3000/tasks?limit=10 Send
```

Query Parameters

Parameter	Value
☒ limit	10
☐ parameter	value

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
[16:17:08] Starting compilation in watch mode...  
[16:17:10] Found 0 errors. Watching for file changes.
```



VS Code interface showing the Explorer view on the left with the file structure of 'PRIMEIRO-APP'. The REST client interface is open, showing a GET request to `localhost:3000/tasks?limit=10&offset=5`. The Query Parameters section shows `limit` set to `10` and `offset` set to `5`. The terminal output shows the compilation process.

```
GET localhost:3000/tasks?limit=10&offset=5 Send
```

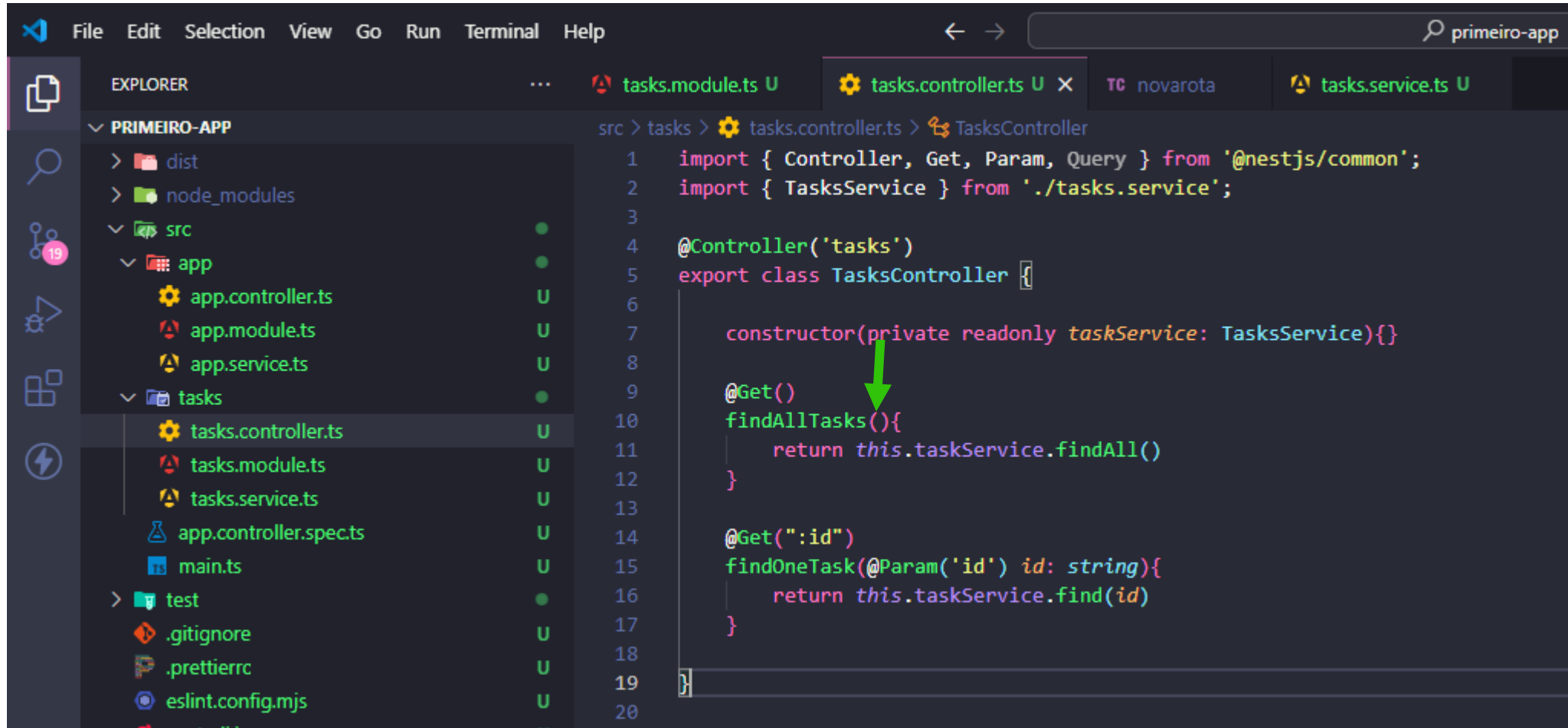
Query Parameters

Parameter	Value
☒ limit	10
☒ offset	5
☐ parameter	value

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
10
```

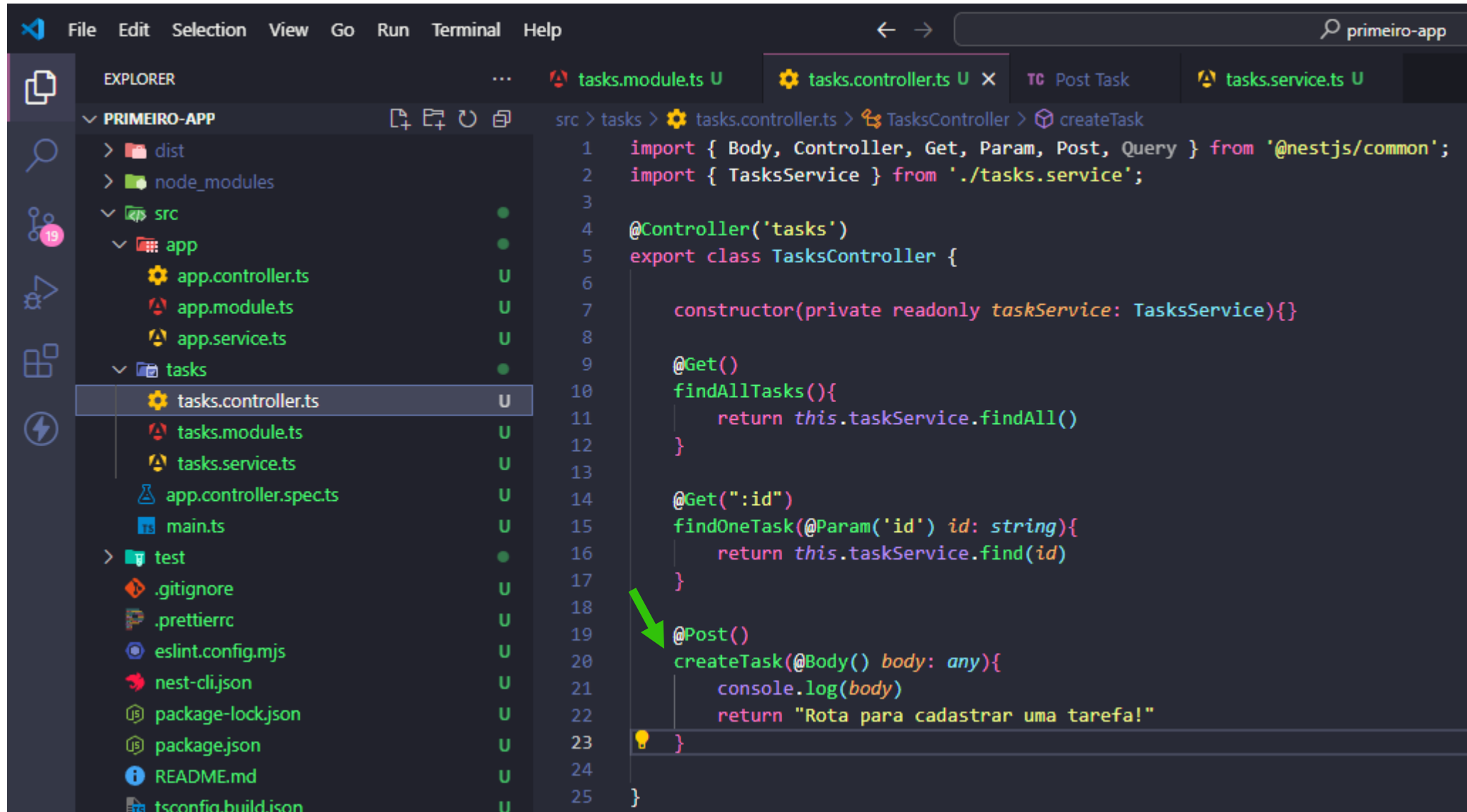
Rotas com parâmetros



The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the project structure for 'PRIMEIRO-APP'. The 'tasks' directory is expanded, showing 'tasks.controller.ts' selected. The main editor area displays the content of 'tasks.controller.ts'. The code defines a 'TasksController' class with a constructor and two methods: 'findAllTasks()' and 'findOneTask()'. A green arrow points to the 'findAllTasks()' method. The breadcrumb at the top indicates the current file path: 'src > tasks > tasks.controller.ts > TasksController'.

```
1 import { Controller, Get, Param, Query } from '@nestjs/common';
2 import { TasksService } from '../tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7     constructor(private readonly taskService: TasksService){}
8
9     @Get()
10    findAllTasks(){
11        return this.taskService.findAll()
12    }
13
14    @Get(":id")
15    findOneTask(@Param('id') id: string){
16        return this.taskService.find(id)
17    }
18
19 }
20
```

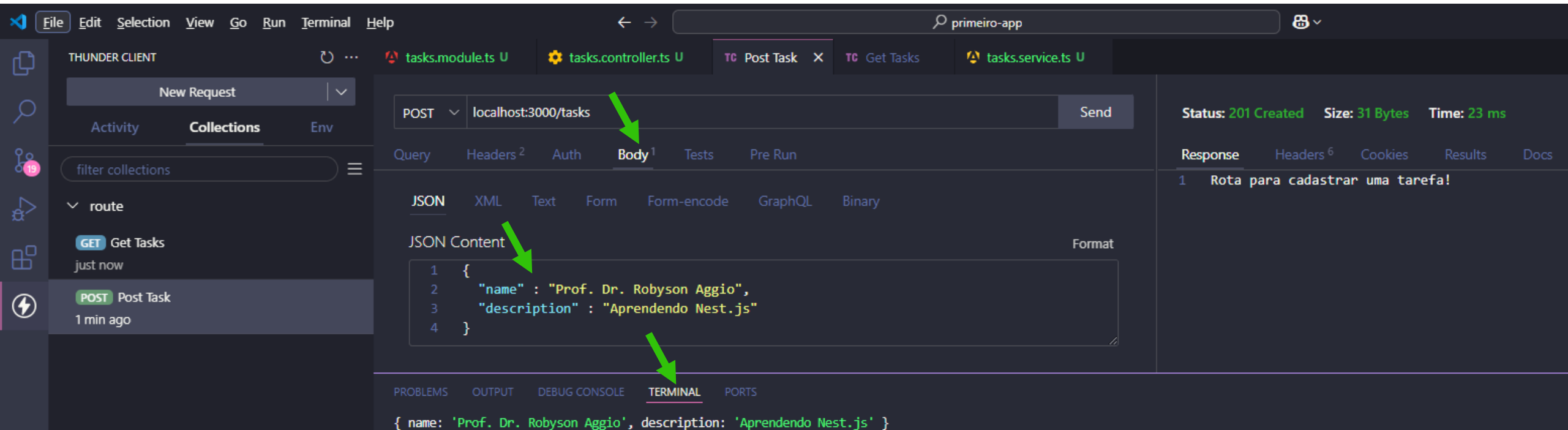
Rotas com Post



The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the project structure for 'PRIMEIRO-APP'. The file 'tasks.controller.ts' is selected and highlighted. The main editor area shows the code for 'tasks.controller.ts', which is a NestJS controller for the 'tasks' module. The code includes imports for NestJS decorators and the 'TasksService'. It defines a 'TasksController' class with a constructor that injects 'TasksService'. The controller has three methods: 'findAllTasks()' (GET), 'findOneTask()' (GET with ID parameter), and 'createTask()' (POST). A green arrow points to the '@Post()' decorator on the 'createTask' method. The breadcrumb at the top of the editor indicates the current file path: 'src > tasks > tasks.controller.ts > TasksController > createTask'.

```
1 import { Body, Controller, Get, Param, Post, Query } from '@nestjs/common';
2 import { TasksService } from '../tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7     constructor(private readonly taskService: TasksService){
8
9     }
10
11     @Get()
12     findAllTasks(){
13         return this.taskService.findAll()
14     }
15
16     @Get(":id")
17     findOneTask(@Param('id') id: string){
18         return this.taskService.find(id)
19     }
20
21     @Post()
22     createTask(@Body() body: any){
23         console.log(body)
24         return "Rota para cadastrar uma tarefa!"
25     }
26 }
```

Rotas com Post



The screenshot shows the Thunder Client interface with a POST request configured. The URL is `localhost:3000/tasks`. The request body is a JSON object: `{ "name": "Prof. Dr. Robyson Aggio", "description": "Aprendendo Nest.js" }`. The response status is `201 Created`, with a size of `31 Bytes` and a time of `23 ms`. The response message is `1 Rota para cadastrar uma tarefa!`. The terminal at the bottom shows the JSON body of the request.

THUNDER CLIENT

File Edit Selection View Go Run Terminal Help

primeiro-app

tasks.module.ts U tasks.controller.ts U TC Post Task X TC Get Tasks tasks.service.ts U

New Request

Activity Collections Env

filter collections

route

GET Get Tasks just now

POST Post Task 1 min ago

POST localhost:3000/tasks Send

Query Headers² Auth Body¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "name" : "Prof. Dr. Robyson Aggio",
3   "description" : "Aprendendo Nest.js"
4 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

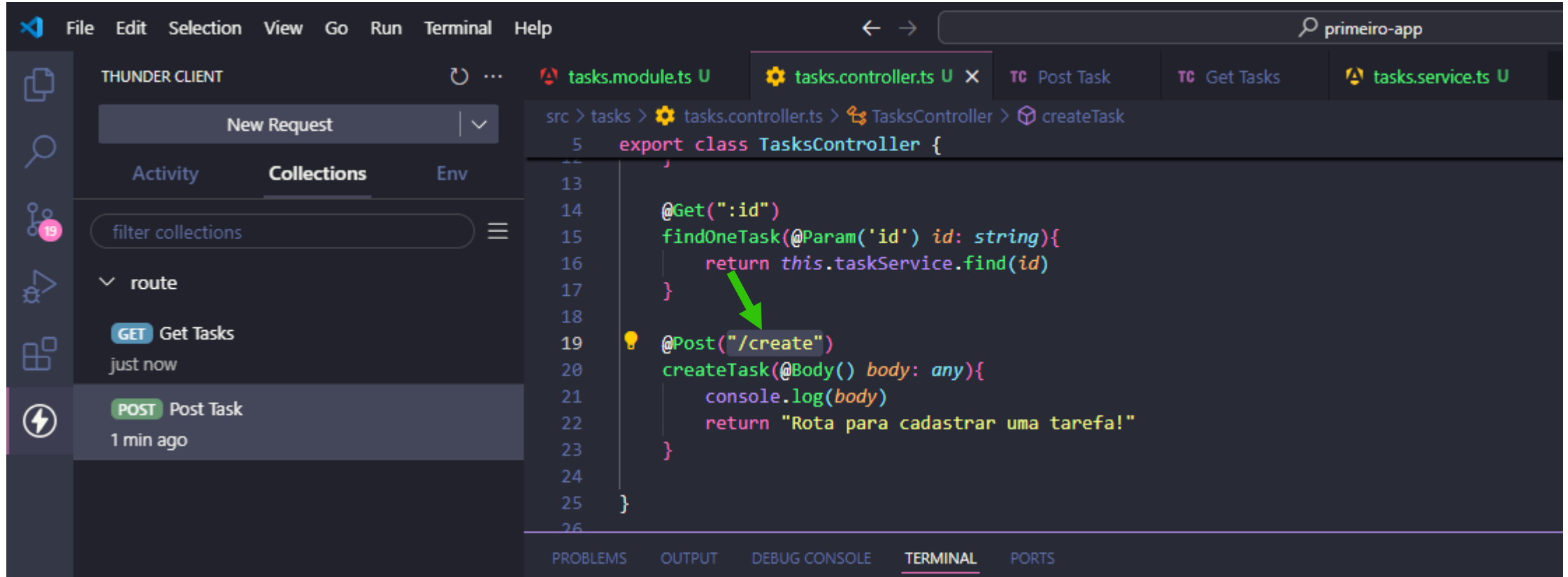
```
{ name: 'Prof. Dr. Robyson Aggio', description: 'Aprendendo Nest.js' }
```

Status: 201 Created Size: 31 Bytes Time: 23 ms

Response Headers⁶ Cookies Results Docs

1 Rota para cadastrar uma tarefa!

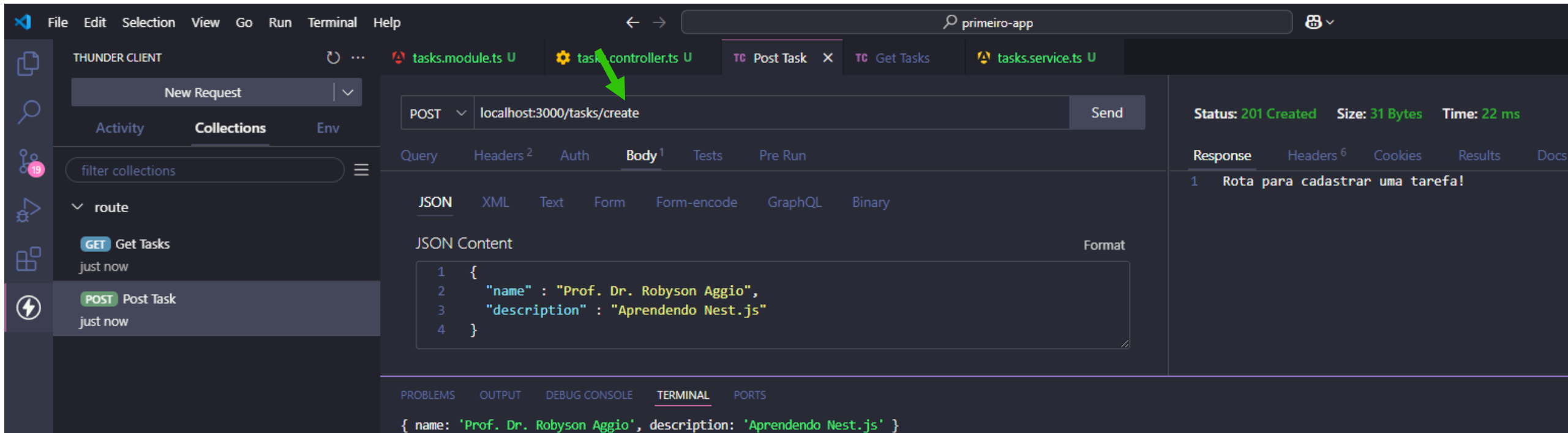
Rotas com Post



The screenshot displays the Thunder Client application interface. On the left sidebar, under the 'Collections' tab, a 'route' is expanded showing two requests: 'GET Get Tasks' and 'POST Post Task'. The 'POST Post Task' request is selected and highlighted. The main editor area shows the code for the 'tasks.controller.ts' file, specifically the 'TasksController' class. A green arrow points from the 'POST /create' route in the sidebar to the '@Post("/create")' decorator in the code. The code defines a 'createTask' method that logs the body and returns a success message.

```
src > tasks > tasks.controller.ts > TasksController > createTask
5  export class TasksController {
13
14    @Get(":id")
15    findOneTask(@Param('id') id: string){
16      return this.taskService.find(id)
17    }
18
19    ⚡ @Post("/create")
20    createTask(@Body() body: any){
21      console.log(body)
22      return "Rota para cadastrar uma tarefa!"
23    }
24
25  }
26
```

Rotas com Post



The screenshot shows the Thunder Client interface with a POST request configured. A green arrow points to the 'task_controller.ts' file in the editor. The request is to 'localhost:3000/tasks/create' with a JSON body containing the name 'Prof. Dr. Robyson Aggio' and the description 'Aprendendo Nest.js'. The response is 201 Created, 31 Bytes, and 22 ms.

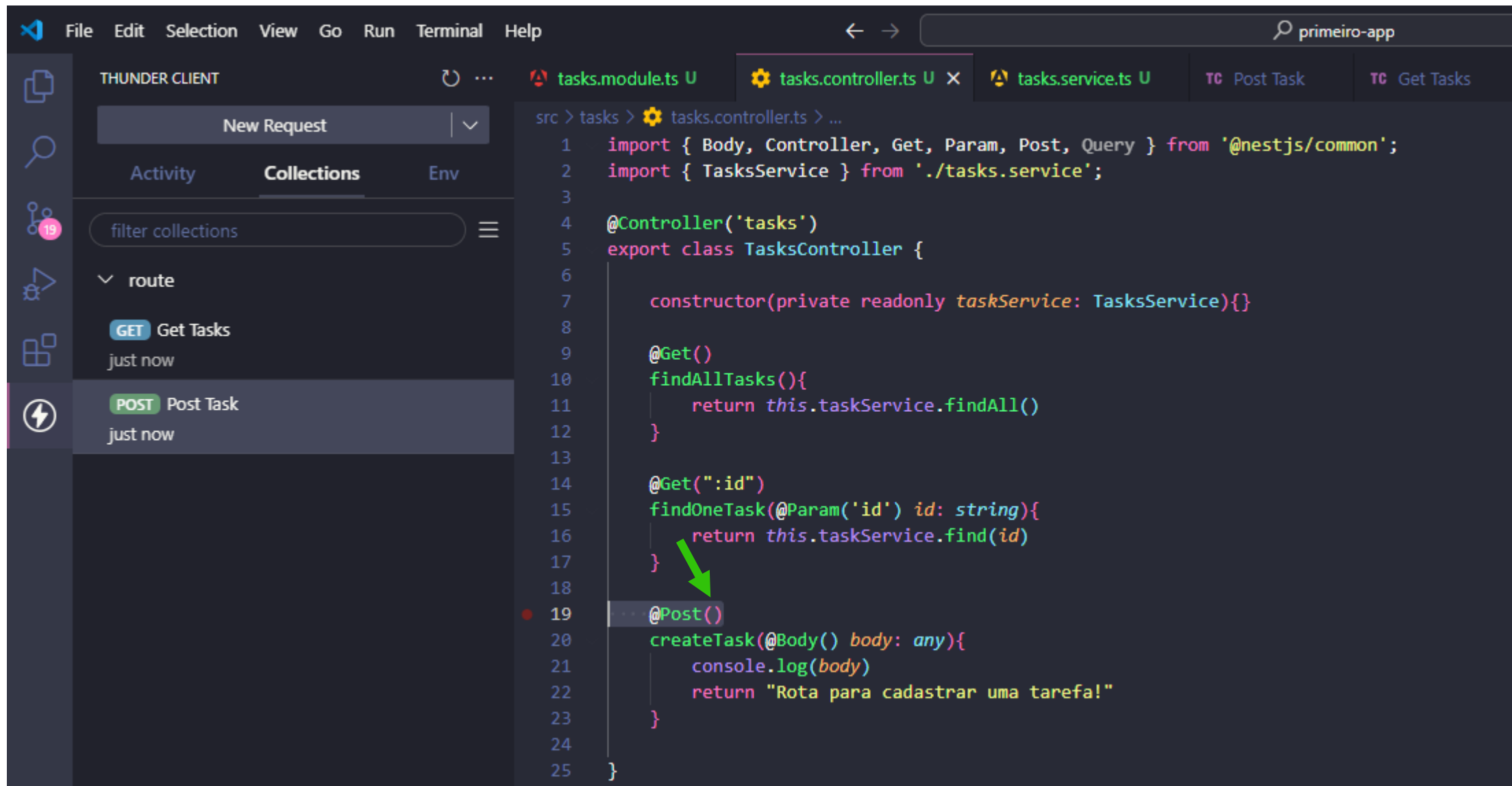
Thunder Client Interface:

- Top Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help. Search: primeiro-app.
- Left Panel:** THUNDER CLIENT. Activity: Collections. Filter: filter collections. Collections: route (GET Get Tasks, POST Post Task).
- Request Editor:**
 - Method: POST
 - URL: localhost:3000/tasks/create
 - Body: JSON Content:

```
{ 1 { 2   "name" : "Prof. Dr. Robyson Aggio", 3   "description" : "Aprendendo Nest.js" 4 }}
```
 - Buttons: Send, Format
- Response Panel:**
 - Status: 201 Created
 - Size: 31 Bytes
 - Time: 22 ms
 - Response: 1 Rota para cadastrar uma tarefa!
- Bottom Panel:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS. Terminal output:

```
{ name: 'Prof. Dr. Robyson Aggio', description: 'Aprendendo Nest.js' }
```

Rotas com Post

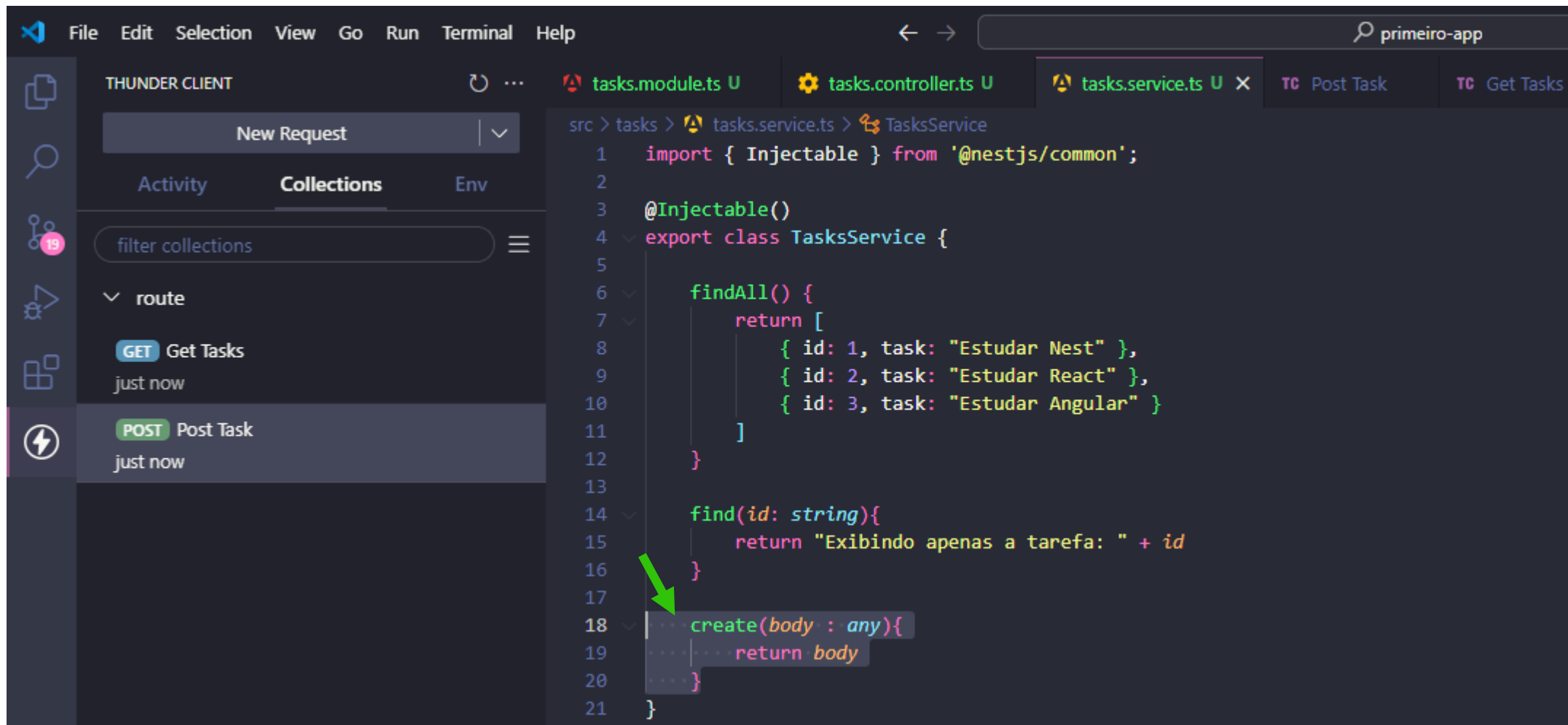


The screenshot displays the Thunder Client application interface. On the left sidebar, the 'route' section is expanded, showing a 'POST' method for 'Post Task' with a 'just now' timestamp. The main editor area shows the 'tasks.controller.ts' file with the following TypeScript code:

```
src > tasks > tasks.controller.ts > ...
1  import { Body, Controller, Get, Param, Post, Query } from '@nestjs/common';
2  import { TasksService } from '../tasks.service';
3
4  @Controller('tasks')
5  export class TasksController {
6
7      constructor(private readonly taskService: TasksService){}
8
9      @Get()
10     findAllTasks(){
11         return this.taskService.findAll()
12     }
13
14     @Get(":id")
15     findOneTask(@Param('id') id: string){
16         return this.taskService.find(id)
17     }
18
19     ... @Post()
20     createTask(@Body() body: any){
21         console.log(body)
22         return "Rota para cadastrar uma tarefa!"
23     }
24
25 }
```

A green arrow points to the `@Post()` decorator on line 19, which is highlighted with a mouse cursor.

Rotas com Post



The screenshot shows the Visual Studio Code interface with a REST client on the left and a TypeScript service file on the right.

Left Panel (REST Client):

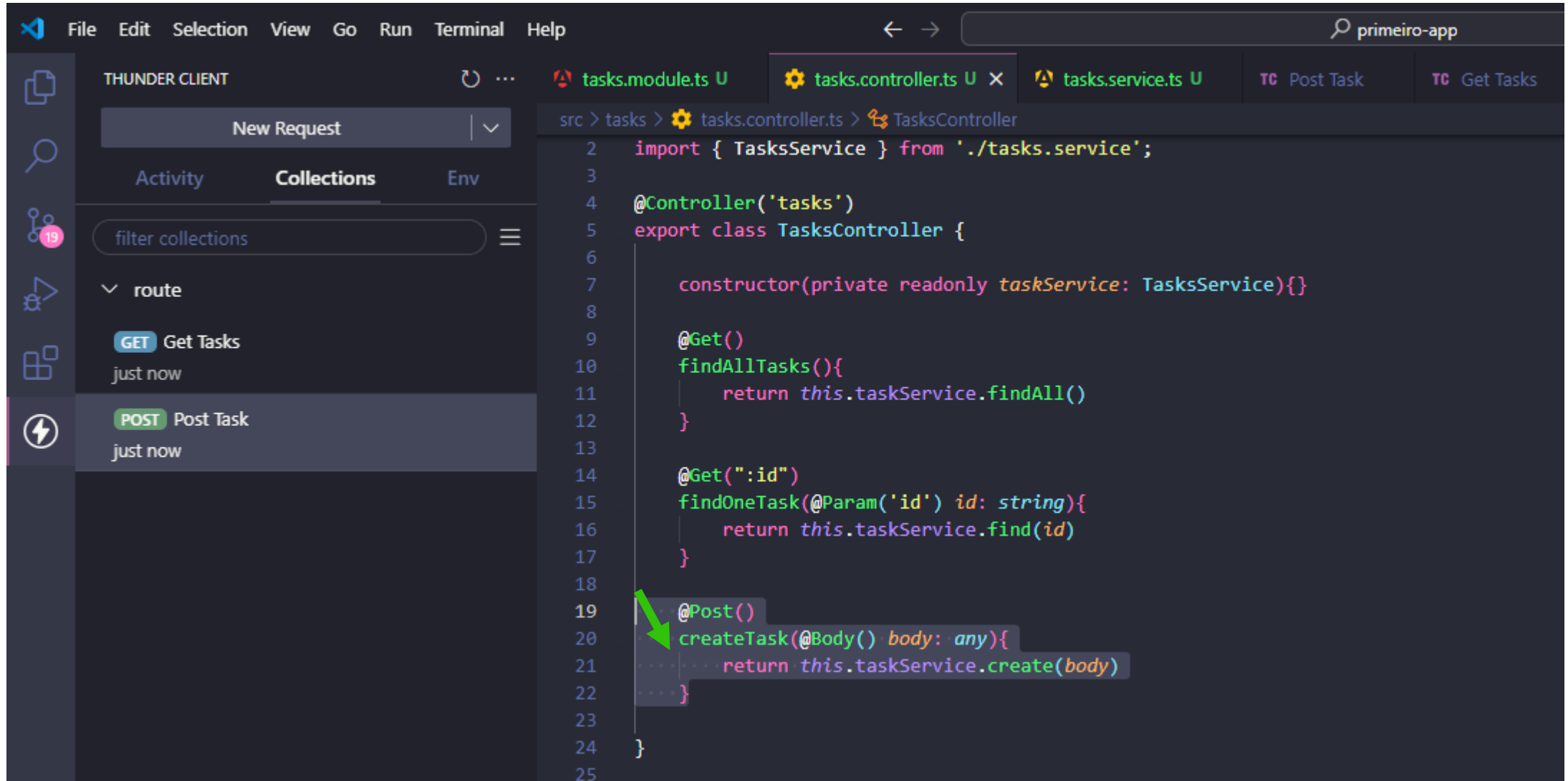
- THUNDER CLIENT
- Activity: Collections
- filter collections
- route
 - GET Get Tasks (just now)
 - POST Post Task (just now)**

Right Panel (tasks.service.ts):

```
src > tasks > tasks.service.ts > TasksService
1  import { Injectable } from '@nestjs/common';
2
3  @Injectable()
4  export class TasksService {
5
6      findAll() {
7          return [
8              { id: 1, task: "Estudar Nest" },
9              { id: 2, task: "Estudar React" },
10             { id: 3, task: "Estudar Angular" }
11          ]
12      }
13
14      find(id: string){
15          return "Exibindo apenas a tarefa: " + id
16      }
17
18      create(body: any){
19          return body
20      }
21  }
```

A green arrow points to the `create` method in the `TasksService` class.

Rotas com Post



The screenshot shows the Visual Studio Code editor with a REST client interface on the left and a TypeScript file on the right.

REST Client Interface (Left Panel):

- THUNDER CLIENT** tab is active.
- New Request** dropdown is set to **POST**.
- Activity** tab is selected, showing a collection of requests.
- route** is expanded, showing two requests: **GET Get Tasks** and **POST Post Task**.
- POST Post Task** is selected, showing it was executed **just now**.

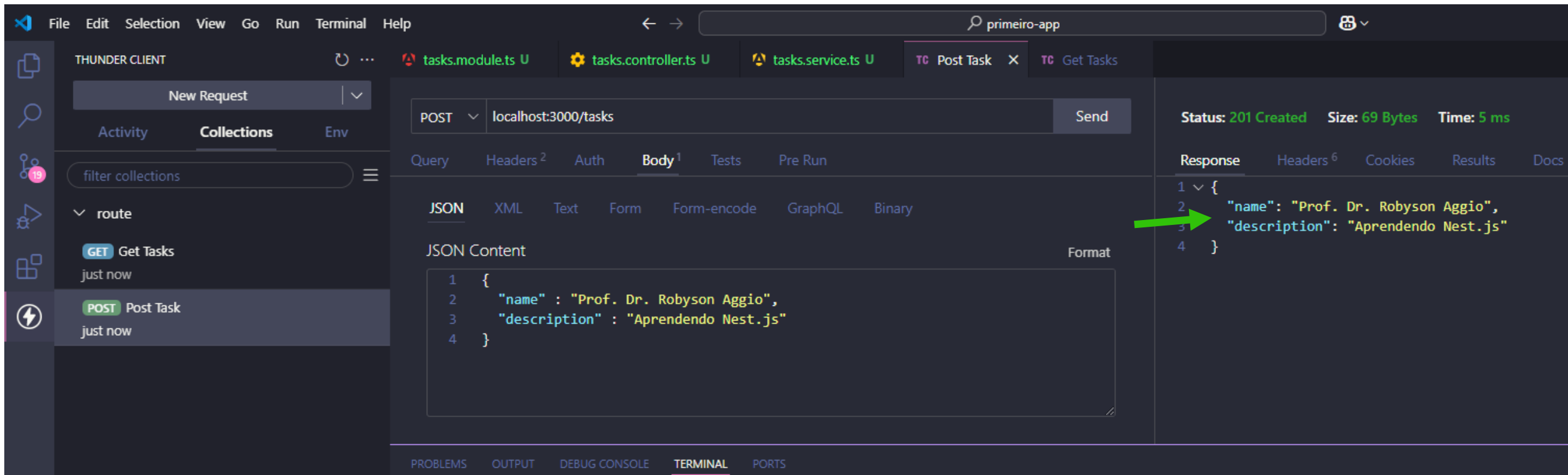
TypeScript File (Right Panel):

The file is `tasks.controller.ts` located at `src > tasks > tasks.controller.ts`. It defines a `TasksController` class with the following methods:

```
2 import { TaskService } from './tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7     constructor(private readonly taskService: TaskService){}
8
9     @Get()
10    findAllTasks(){
11        return this.taskService.findAll()
12    }
13
14    @Get(":id")
15    findOneTask(@Param('id') id: string){
16        return this.taskService.find(id)
17    }
18
19    @Post()
20    createTask(@Body() body: any){
21        return this.taskService.create(body)
22    }
23
24 }
```

A green arrow points to the `@Post()` decorator and the `createTask` method.

Rotas com Post



The screenshot displays the Thunder Client interface with a POST request to `localhost:3000/tasks` successfully executed. The request body is a JSON object containing the name and description of the user. The response is a 201 Created status with the same JSON object in the body.

Request Details:

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

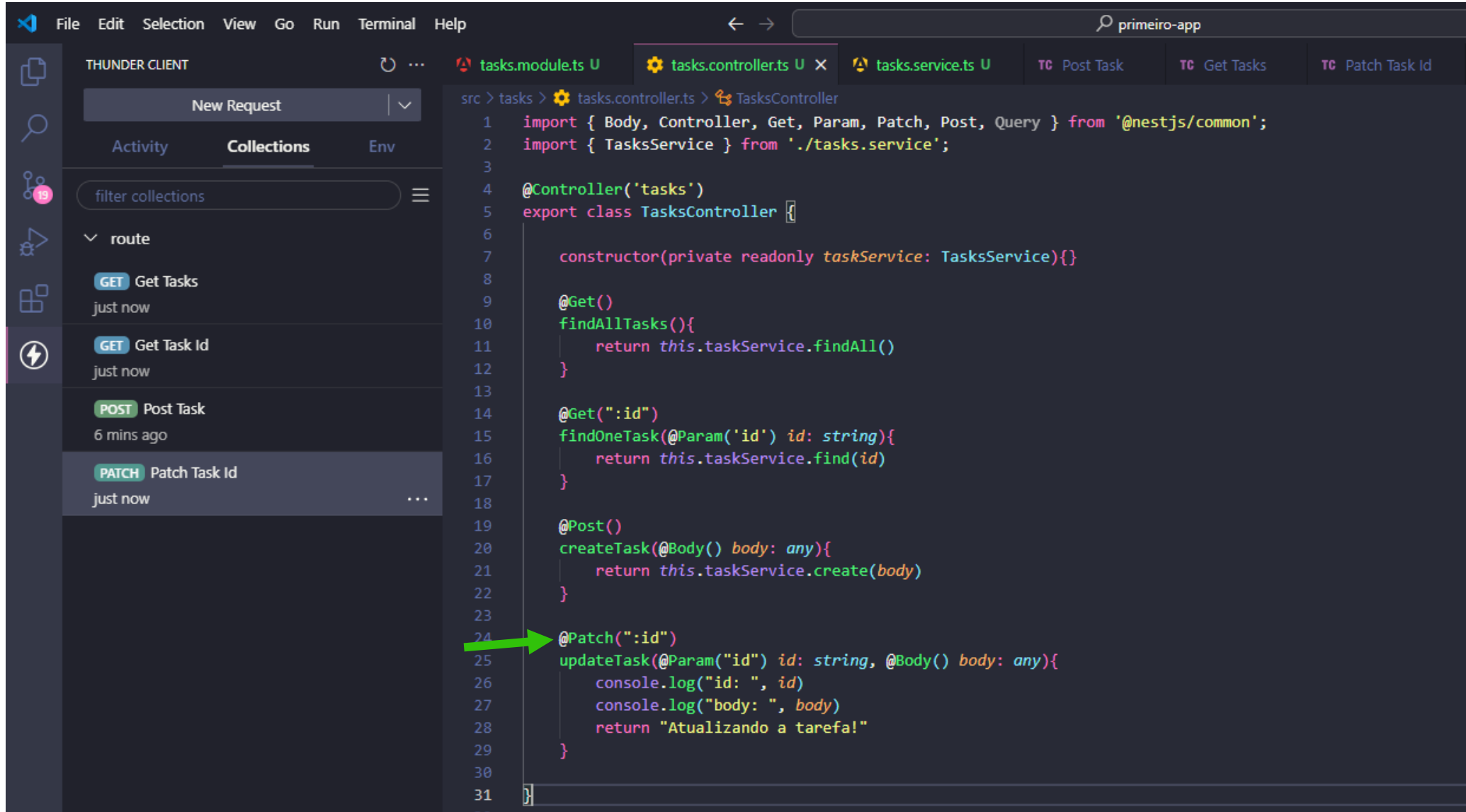
```
1 {
2   "name" : "Prof. Dr. Robyson Aggio",
3   "description" : "Aprendendo Nest.js"
4 }
```

Response Details:

- Status: 201 Created
- Size: 69 Bytes
- Time: 5 ms
- Response (JSON):

```
1 {
2   "name": "Prof. Dr. Robyson Aggio",
3   "description": "Aprendendo Nest.js"
4 }
```

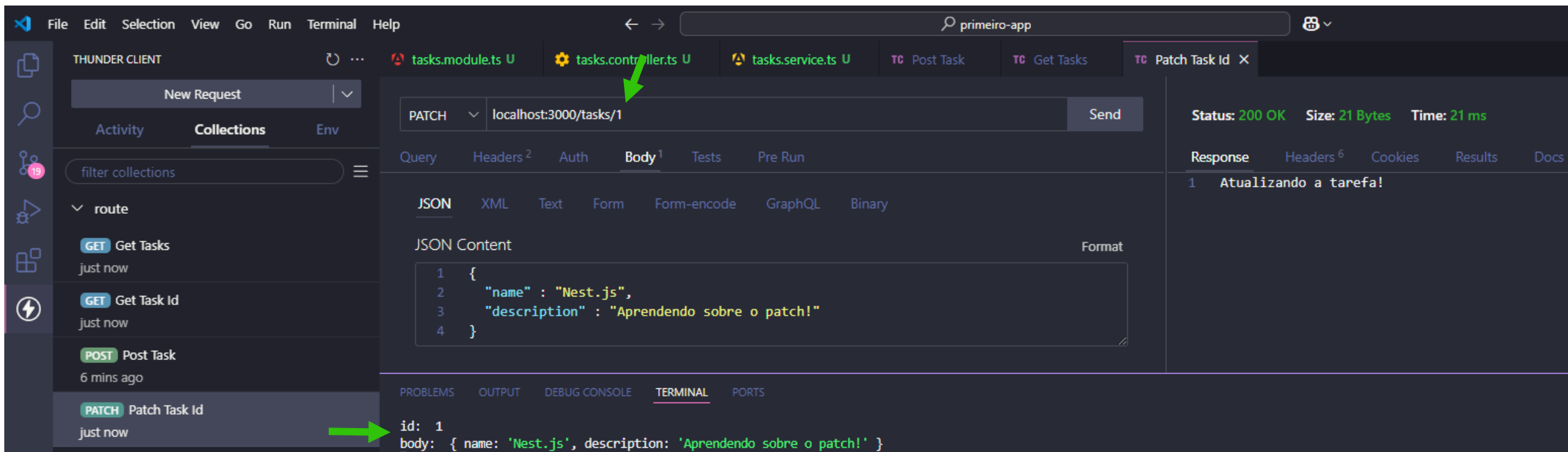
Rotas com Patch



The image shows a development environment with a REST client and a TypeScript controller. The REST client on the left lists several routes, including a **PATCH Patch Task Id** route. The TypeScript code on the right implements the `TasksController` with a `@Patch` method that updates a task. A green arrow points to the `@Patch` decorator in the code.

```
src > tasks > tasks.controller.ts > TasksController
1  import { Body, Controller, Get, Param, Patch, Post, Query } from '@nestjs/common';
2  import { TasksService } from '../tasks.service';
3
4  @Controller('tasks')
5  export class TasksController {
6
7      constructor(private readonly taskService: TasksService){}
8
9      @Get()
10     findAllTasks(){
11         return this.taskService.findAll()
12     }
13
14     @Get(":id")
15     findOneTask(@Param('id') id: string){
16         return this.taskService.find(id)
17     }
18
19     @Post()
20     createTask(@Body() body: any){
21         return this.taskService.create(body)
22     }
23
24     @Patch(":id")
25     updateTask(@Param("id") id: string, @Body() body: any){
26         console.log("id: ", id)
27         console.log("body: ", body)
28         return "Atualizando a tarefa!"
29     }
30
31 }
```

Rotas com Patch



The screenshot displays the Thunder Client interface with a PATCH request configured. The request URL is `localhost:3000/tasks/1`. The body is a JSON object: `{ "name": "Nest.js", "description": "Aprendendo sobre o patch!" }`. The response status is `200 OK` with a size of `21 Bytes` and a time of `21 ms`. The response body is `1 Atualizando a tarefa!`.

Left sidebar (Collections):

- route
 - GET Get Tasks (just now)
 - GET Get Task Id (just now)
 - POST Post Task (6 mins ago)
 - PATCH Patch Task Id (just now)

Request details:

- Method: PATCH
- URL: localhost:3000/tasks/1
- Body (JSON):

```
1 {
2   "name" : "Nest.js",
3   "description" : "Aprendendo sobre o patch!"
4 }
```

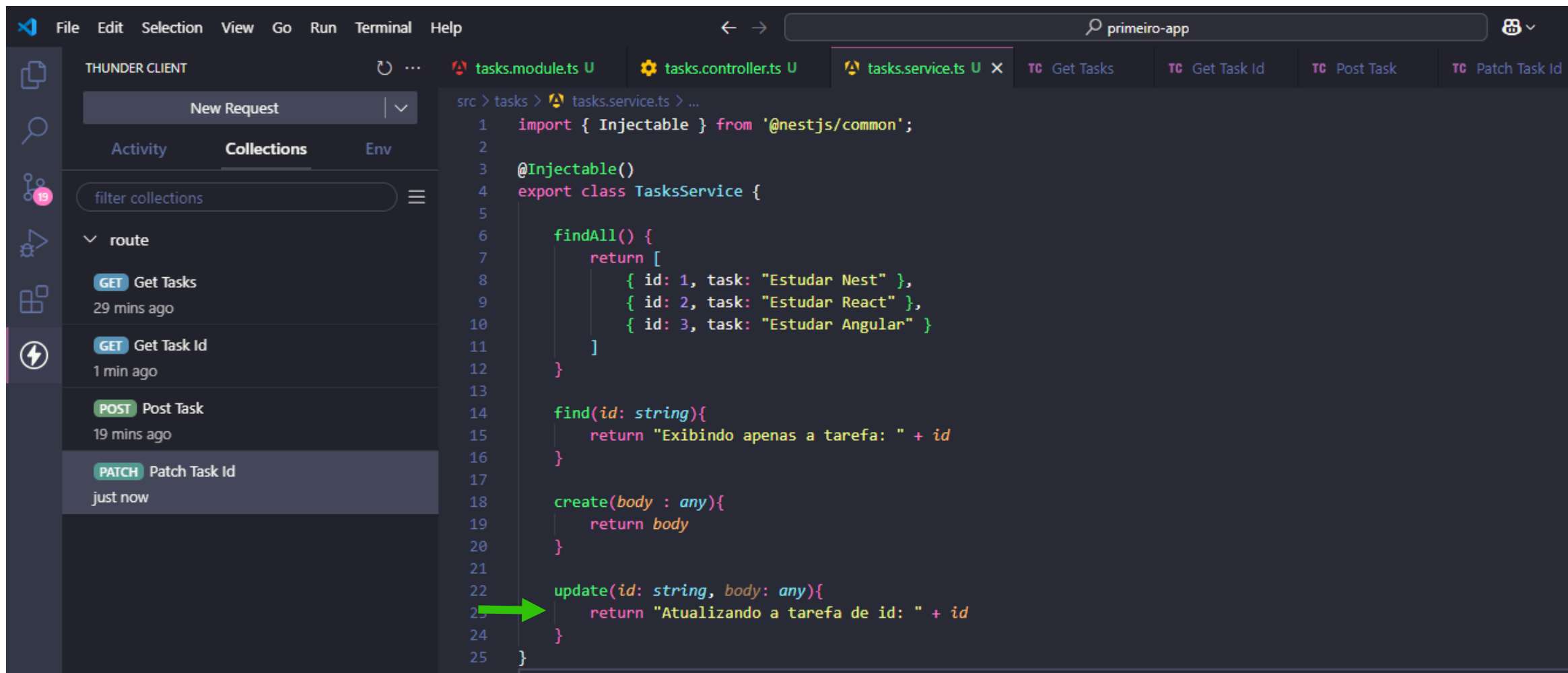
Response details:

- Status: 200 OK
- Size: 21 Bytes
- Time: 21 ms
- Response body: 1 Atualizando a tarefa!

Terminal output:

```
id: 1
body: { name: 'Nest.js', description: 'Aprendendo sobre o patch!' }
```

Rotas com Patch



The image shows a screenshot of a development environment (VS Code) with a REST client (Thunder Client) and a TypeScript service file.

Thunder Client (Left Panel):

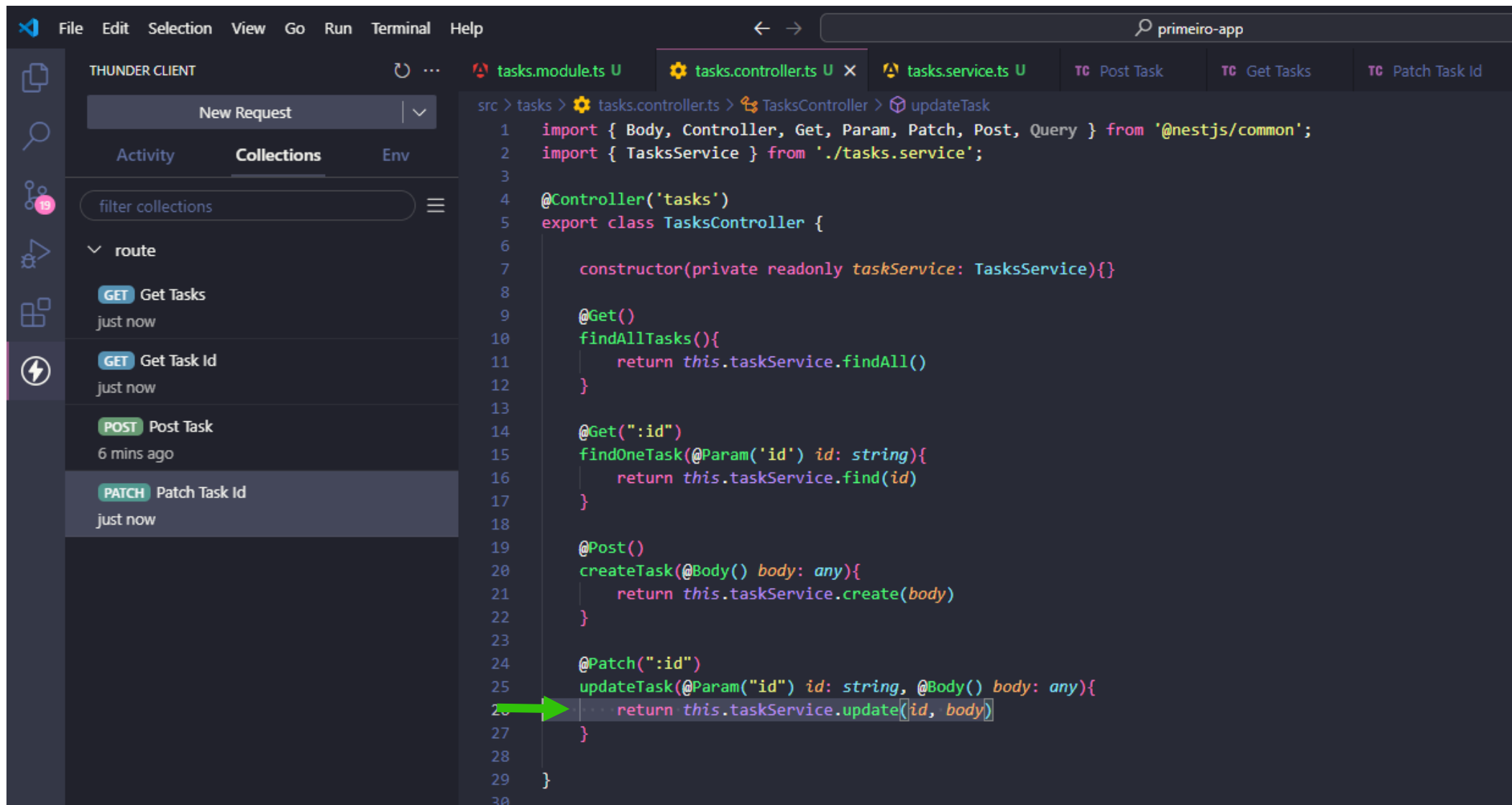
- Activity: Collections, Env
- filter collections
- route
- GET Get Tasks (29 mins ago)
- GET Get Task Id (1 min ago)
- POST Post Task (19 mins ago)
- PATCH Patch Task Id (just now)**

tasks.service.ts (Right Panel):

```
src > tasks > tasks.service.ts > ...
1  import { Injectable } from '@nestjs/common';
2
3  @Injectable()
4  export class TasksService {
5
6      findAll() {
7          return [
8              { id: 1, task: "Estudar Nest" },
9              { id: 2, task: "Estudar React" },
10             { id: 3, task: "Estudar Angular" }
11          ]
12      }
13
14      find(id: string){
15          return "Exibindo apenas a tarefa: " + id
16      }
17
18      create(body : any){
19          return body
20      }
21
22      update(id: string, body: any){
23          return "Atualizando a tarefa de id: " + id
24      }
25  }
```

A green arrow points to the `update` method in the `tasks.service.ts` file.

Rotas com Patch



The screenshot displays the Visual Studio Code editor with a REST client interface on the left and a TypeScript controller file on the right.

Left Panel (REST Client):

- THUNDER CLIENT** tab is active.
- New Request** button is visible.
- Activity** and **Collections** tabs are shown.
- filter collections** input field is present.
- route** dropdown is expanded, showing a list of routes:

Method	Route	Time
GET	Get Tasks	just now
GET	Get Task Id	just now
POST	Post Task	6 mins ago
PATCH	Patch Task Id	just now

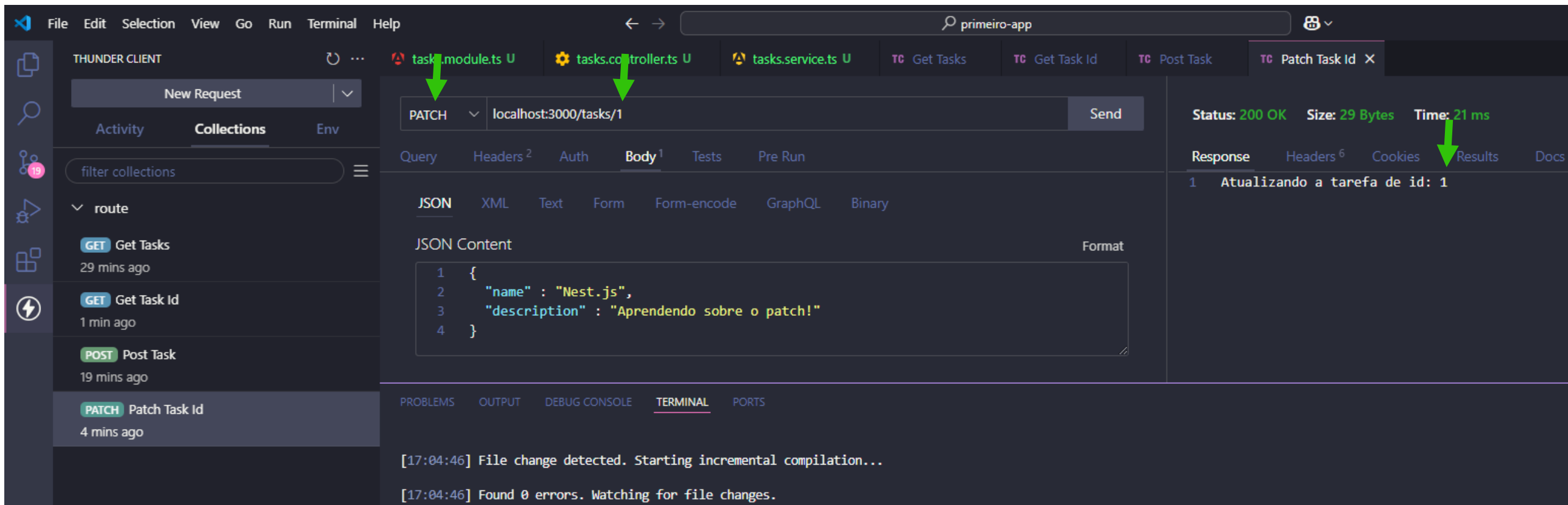
Right Panel (Code Editor):

- tasks.module.ts U** tab is active.
- tasks.controller.ts U** tab is active.
- tasks.service.ts U** tab is active.
- TC Post Task** tab is active.
- TC Get Tasks** tab is active.
- TC Patch Task Id** tab is active.

The code in the **tasks.controller.ts** file is as follows:

```
1 import { Body, Controller, Get, Param, Patch, Post, Query } from '@nestjs/common';
2 import { TasksService } from './tasks.service';
3
4 @Controller('tasks')
5 export class TasksController {
6
7     constructor(private readonly taskService: TasksService){}
8
9     @Get()
10    findAllTasks(){
11        return this.taskService.findAll()
12    }
13
14    @Get(":id")
15    findOneTask(@Param('id') id: string){
16        return this.taskService.find(id)
17    }
18
19    @Post()
20    createTask(@Body() body: any){
21        return this.taskService.create(body)
22    }
23
24    @Patch(":id")
25    updateTask(@Param("id") id: string, @Body() body: any){
26        return this.taskService.update(id, body)
27    }
28
29 }
```

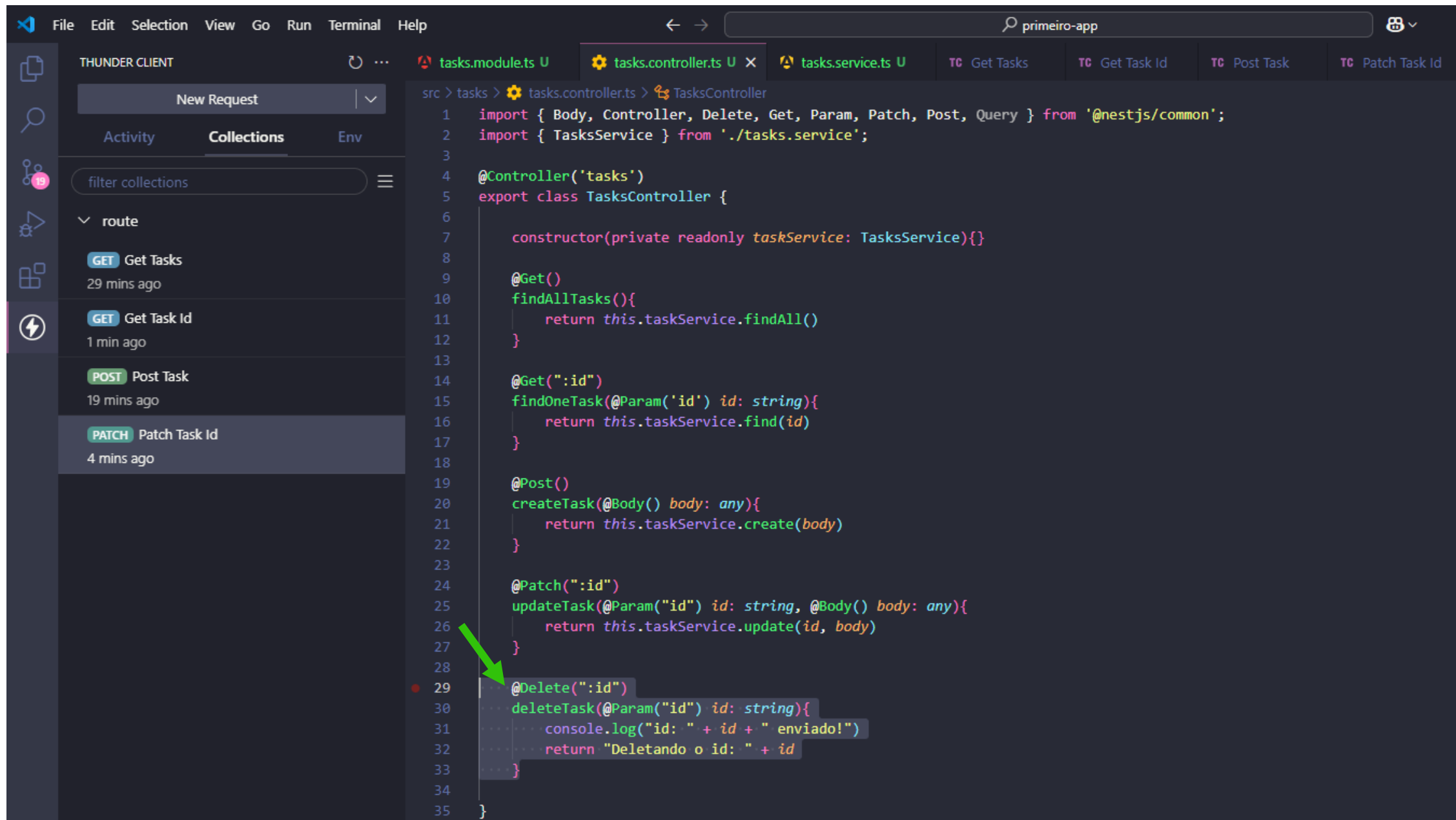
Rotas com Patch



The screenshot displays the Thunder Client interface with a PATCH request configured to `localhost:3000/tasks/1`. The request body is a JSON object: `{ "name": "Nest.js", "description": "Aprendendo sobre o patch!" }`. The response status is `200 OK`, with a size of `29 Bytes` and a time of `21 ms`. The response body contains the message `Atualizando a tarefa de id: 1`. The terminal at the bottom shows the following output:

```
[17:04:46] File change detected. Starting incremental compilation...  
[17:04:46] Found 0 errors. Watching for file changes.
```

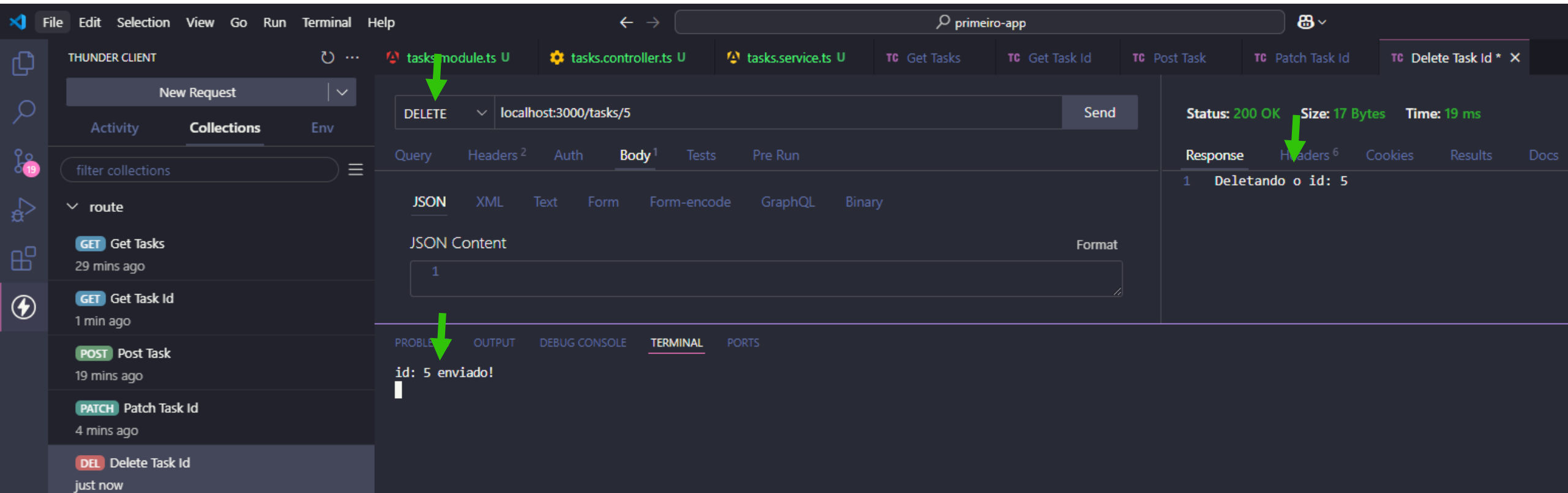
Rotas com Delete



The screenshot displays the Thunder Client interface with a REST API collection named 'primeiro-app'. The 'route' section lists several endpoints: 'GET Get Tasks' (29 mins ago), 'GET Get Task Id' (1 min ago), 'POST Post Task' (19 mins ago), and 'PATCH Patch Task Id' (4 mins ago). The 'PATCH Patch Task Id' route is highlighted. The code editor shows the implementation of the 'TasksController' in 'tasks.controller.ts'. The controller uses the '@nestjs/common' and './tasks.service' imports. It defines a constructor and several methods: 'findAllTasks()', 'findOneTask()', 'createTask()', 'updateTask()', and a new 'deleteTask()' method. A green arrow points to the 'deleteTask()' method, which is highlighted in the code editor.

```
src > tasks > tasks.controller.ts > TasksController
1  import { Body, Controller, Delete, Get, Param, Patch, Post, Query } from '@nestjs/common';
2  import { TaskService } from './tasks.service';
3
4  @Controller('tasks')
5  export class TasksController {
6
7      constructor(private readonly taskService: TaskService){}
8
9      @Get()
10     findAllTasks(){
11         return this.taskService.findAll();
12     }
13
14     @Get(":id")
15     findOneTask(@Param('id') id: string){
16         return this.taskService.find(id)
17     }
18
19     @Post()
20     createTask(@Body() body: any){
21         return this.taskService.create(body)
22     }
23
24     @Patch(":id")
25     updateTask(@Param("id") id: string, @Body() body: any){
26         return this.taskService.update(id, body)
27     }
28
29     @Delete(":id")
30     deleteTask(@Param("id") id: string){
31         console.log("id: " + id + " enviado!")
32         return "Deletando o id: " + id
33     }
34
35 }
```

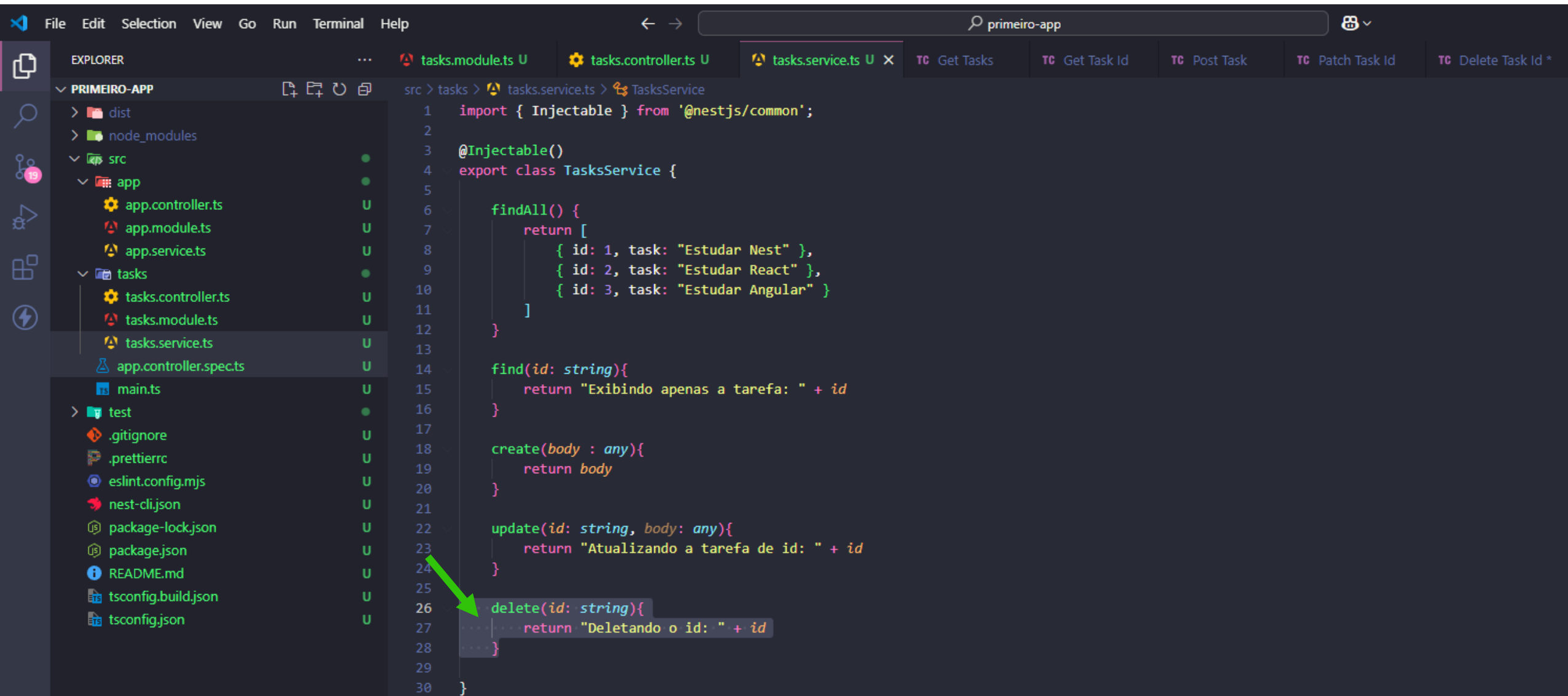
Rotas com Delete



The screenshot displays the Thunder Client interface with a DELETE request configured to `localhost:3000/tasks/5`. The request body is set to JSON with the value `1`. The response status is `200 OK`, with a size of `17 Bytes` and a time of `19 ms`. The response body contains the text `Deletando o id: 5`. The terminal at the bottom shows the message `id: 5 enviado!`.

Thunder Client interface showing a DELETE request to `localhost:3000/tasks/5` with a JSON body containing `1`. The response is `200 OK` with a size of `17 Bytes` and a time of `19 ms`. The response body contains the text `Deletando o id: 5`. The terminal shows the message `id: 5 enviado!`.

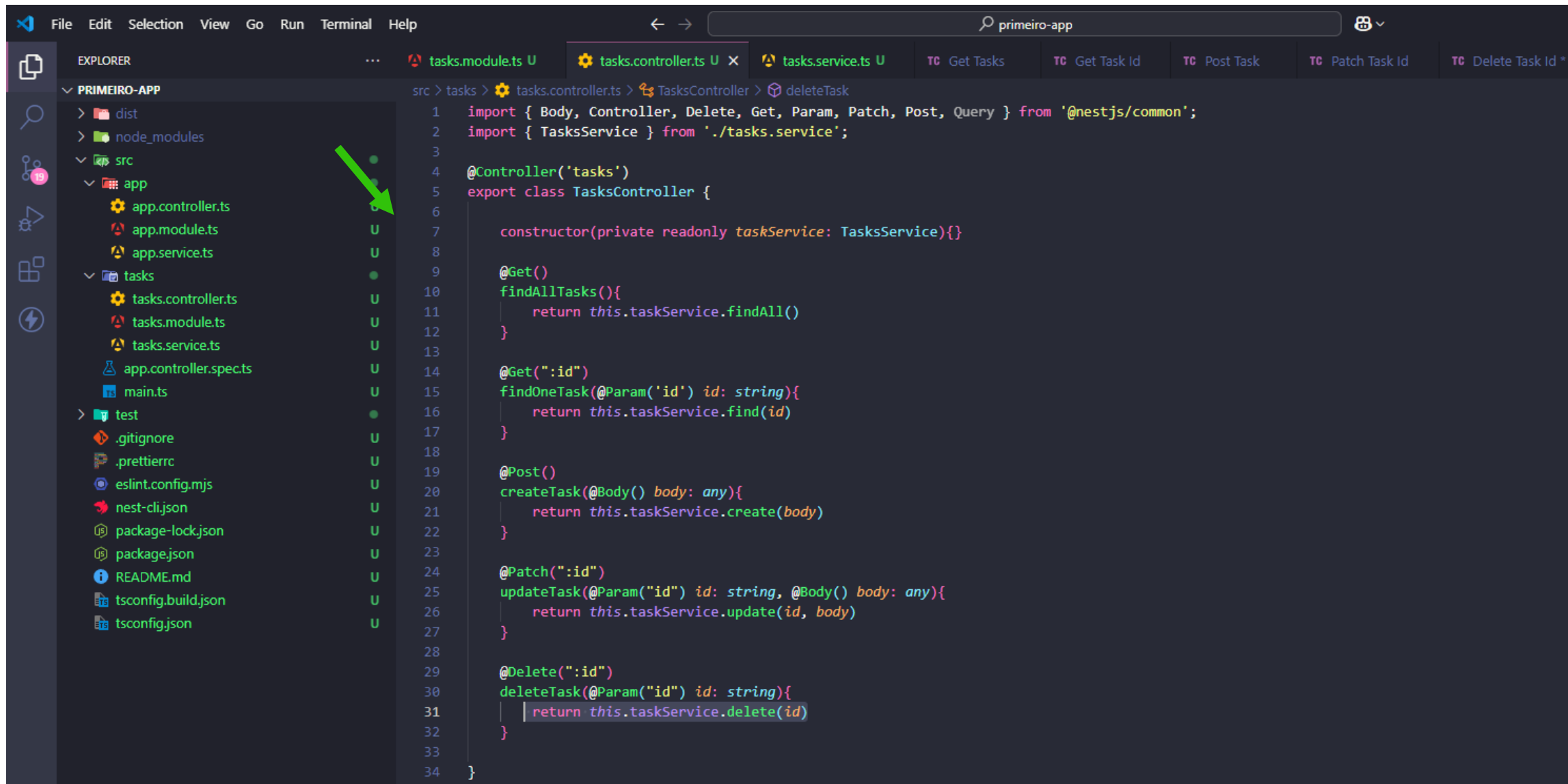
Rotas com Delete



The screenshot shows the Visual Studio Code editor with a project named "primeiro-app". The Explorer sidebar on the left displays the file structure, including the "tasks" directory. The "tasks.service.ts" file is open in the editor, showing the implementation of the `delete` method. A green arrow points to the `delete` method definition.

```
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class TasksService {
5
6   findAll() {
7     return [
8       { id: 1, task: "Estudar Nest" },
9       { id: 2, task: "Estudar React" },
10      { id: 3, task: "Estudar Angular" }
11    ]
12  }
13
14  find(id: string){
15    return "Exibindo apenas a tarefa: " + id
16  }
17
18  create(body : any){
19    return body
20  }
21
22  update(id: string, body: any){
23    return "Atualizando a tarefa de id: " + id
24  }
25
26  delete(id: string){
27    return "Deletando o id: " + id
28  }
29
30 }
```

Rotas com Delete



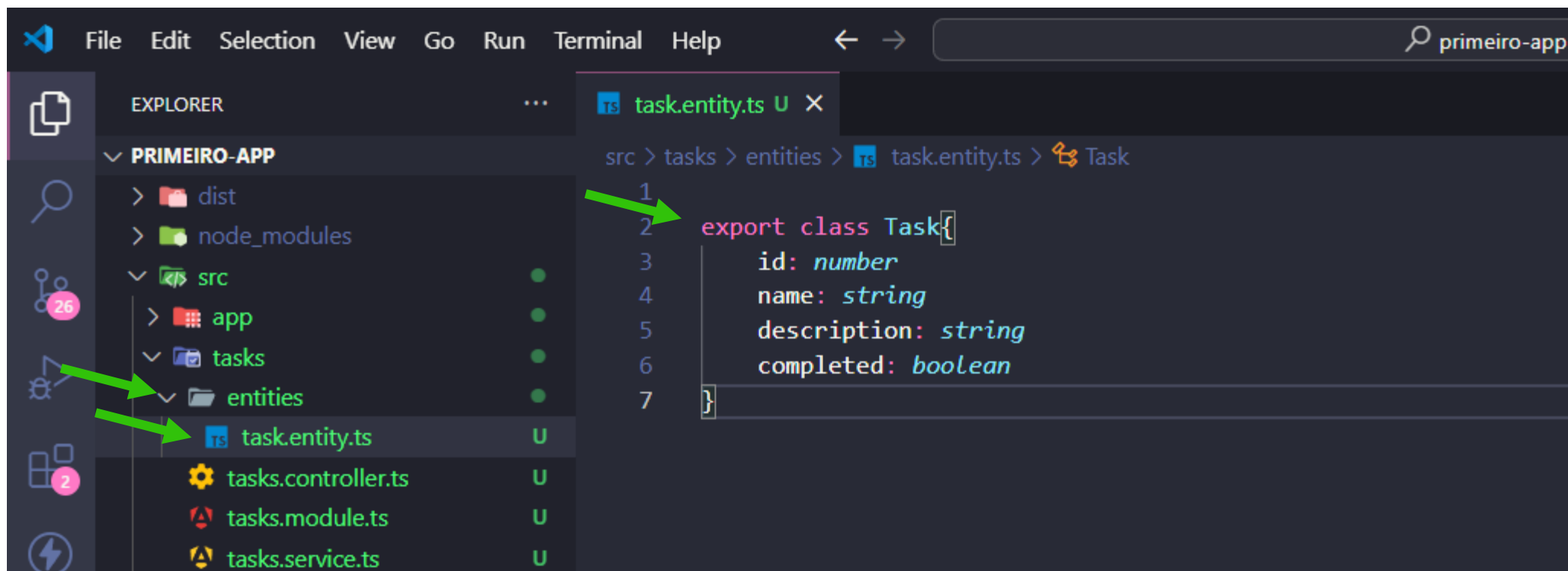
The screenshot shows the Visual Studio Code editor with a project named 'primeiro-app'. The Explorer on the left shows the file structure, with a green arrow pointing to 'tasks.controller.ts' under the 'tasks' directory. The main editor displays the code for 'tasks.controller.ts', which includes imports for NestJS decorators and the 'TasksService'. The controller has several routes: 'findAllTasks' (GET), 'findOneTask' (GET), 'createTask' (POST), 'updateTask' (PATCH), and 'deleteTask' (DELETE). The 'deleteTask' method is highlighted with a green box.

```

src > tasks > tasks.controller.ts > TasksController > deleteTask
1  import { Body, Controller, Delete, Get, Param, Patch, Post, Query } from '@nestjs/common';
2  import { TasksService } from '../tasks.service';
3
4  @Controller('tasks')
5  export class TasksController {
6
7      constructor(private readonly taskService: TasksService){}
8
9      @Get()
10     findAllTasks(){
11         return this.taskService.findAll()
12     }
13
14     @Get(":id")
15     findOneTask(@Param('id') id: string){
16         return this.taskService.find(id)
17     }
18
19     @Post()
20     createTask(@Body() body: any){
21         return this.taskService.create(body)
22     }
23
24     @Patch(":id")
25     updateTask(@Param("id") id: string, @Body() body: any){
26         return this.taskService.update(id, body)
27     }
28
29     @Delete(":id")
30     deleteTask(@Param("id") id: string){
31         return this.taskService.delete(id)
32     }
33
34 }

```

Listando Itens



File Edit Selection View Go Run Terminal Help

primeiro-app

EXPLORER

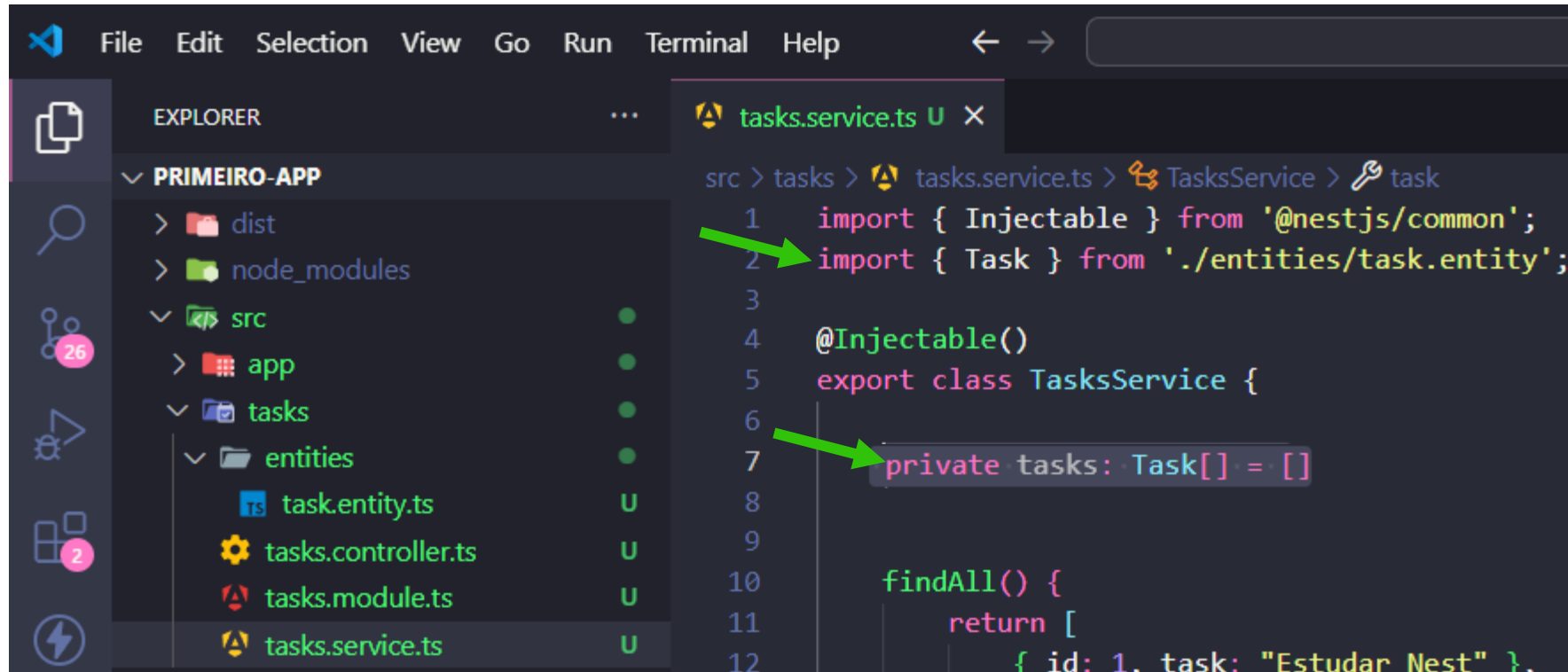
PRIMEIRO-APP

- dist
- node_modules
- src
 - app
 - tasks
 - entities
 - task.entity.ts
 - tasks.controller.ts
 - tasks.module.ts
 - tasks.service.ts

task.entity.ts

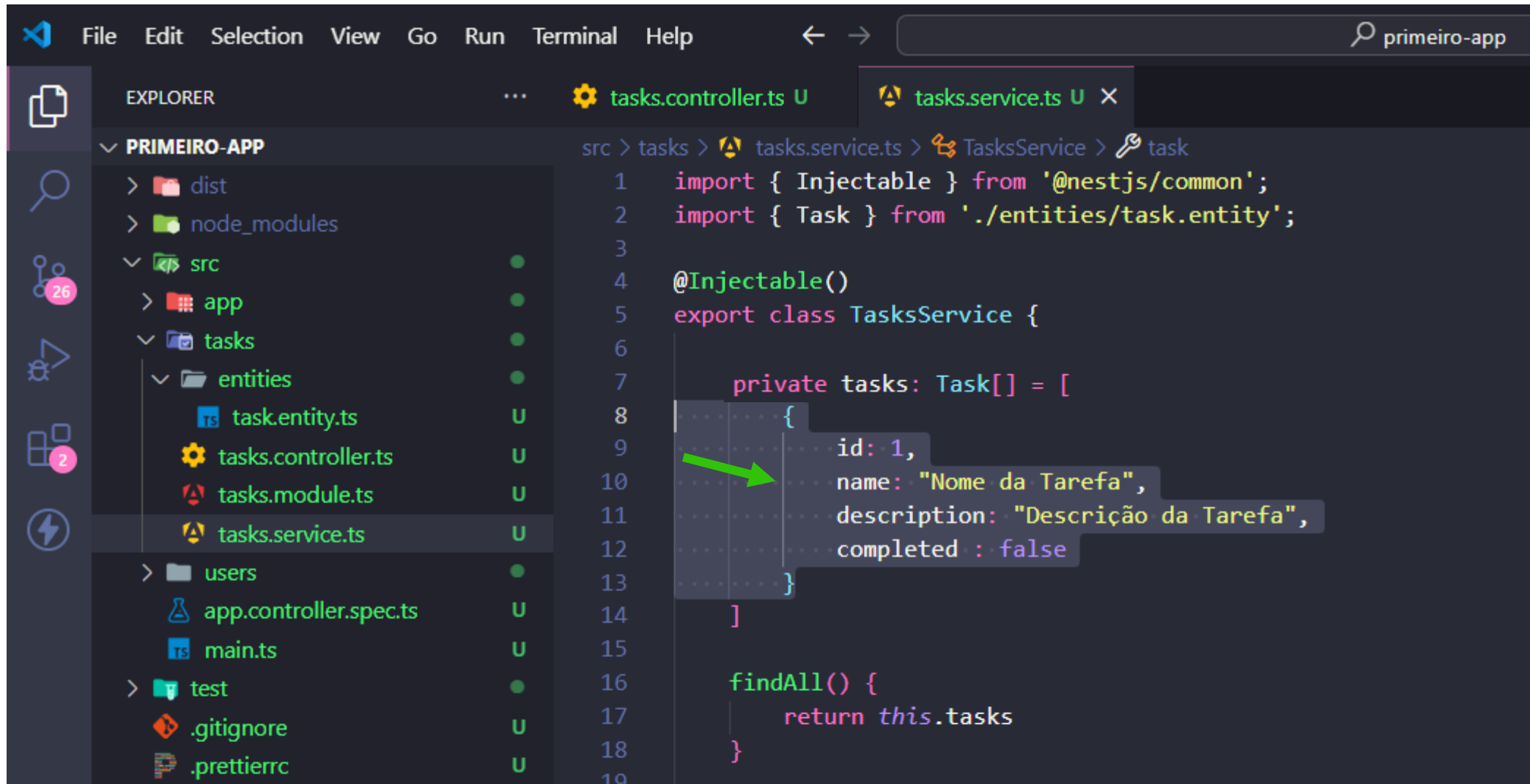
```
1  
2 export class Task {  
3     id: number  
4     name: string  
5     description: string  
6     completed: boolean  
7 }
```

Listando Itens



```
File Edit Selection View Go Run Terminal Help  
EXPLORER  
PRIMEIRO-APP  
  > dist  
  > node_modules  
  > src  
    > app  
    > tasks  
      > entities  
        task.entity.ts  
        tasks.controller.ts  
        tasks.module.ts  
        tasks.service.ts  
tasks.service.ts U X  
src > tasks > tasks.service.ts > TasksService > task  
1 import { Injectable } from '@nestjs/common';  
2 import { Task } from './entities/task.entity';  
3  
4 @Injectable()  
5 export class TasksService {  
6  
7   private tasks: Task[] = [];  
8  
9  
10  findAll() {  
11    return [  
12      { id: 1, task: "Estudar Nest" },
```

Listando Itens



```
File Edit Selection View Go Run Terminal Help
```

EXPLORER

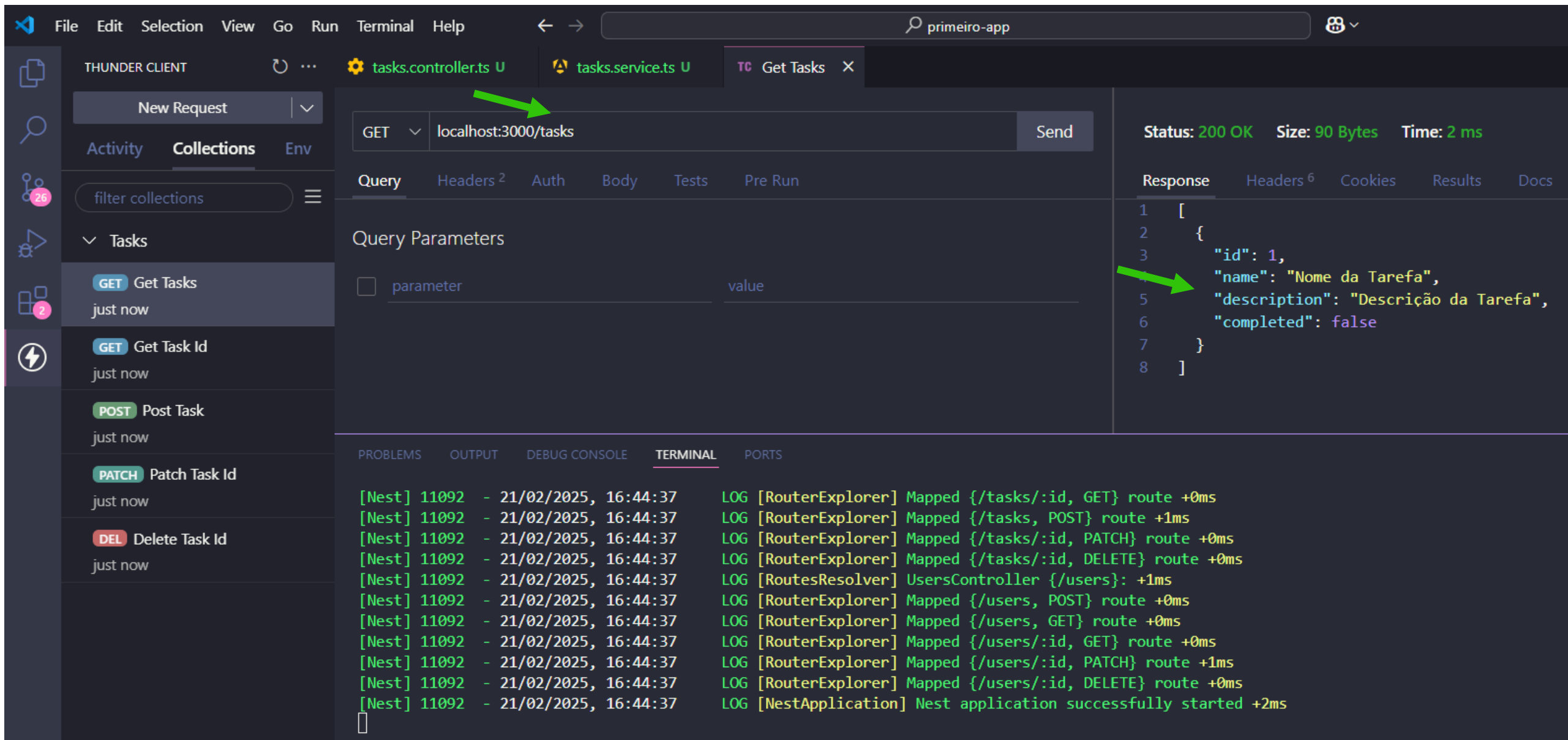
PRIMEIRO-APP

- dist
- node_modules
- src
 - app
 - tasks
 - entities
 - task.entity.ts
 - tasks.controller.ts
 - tasks.module.ts
 - tasks.service.ts
- users
- app.controller.spec.ts
- main.ts
- test
- .gitignore
- .prettierrc

src > tasks > tasks.service.ts > TasksService > task

```
1 import { Injectable } from '@nestjs/common';
2 import { Task } from '../entities/task.entity';
3
4 @Injectable()
5 export class TasksService {
6
7     private tasks: Task[] = [
8         {
9             id: 1,
10             name: "Nome da Tarefa",
11             description: "Descrição da Tarefa",
12             completed: false
13         }
14     ]
15
16     findAll() {
17         return this.tasks
18     }
19 }
```

Listando Itens



The screenshot displays the Thunder Client interface with a REST client request configured to `localhost:3000/tasks` using the `GET` method. The response status is `200 OK` with a size of `90 Bytes` and a time of `2 ms`. The response body is a JSON array containing one task object.

Request:

```
GET localhost:3000/tasks
```

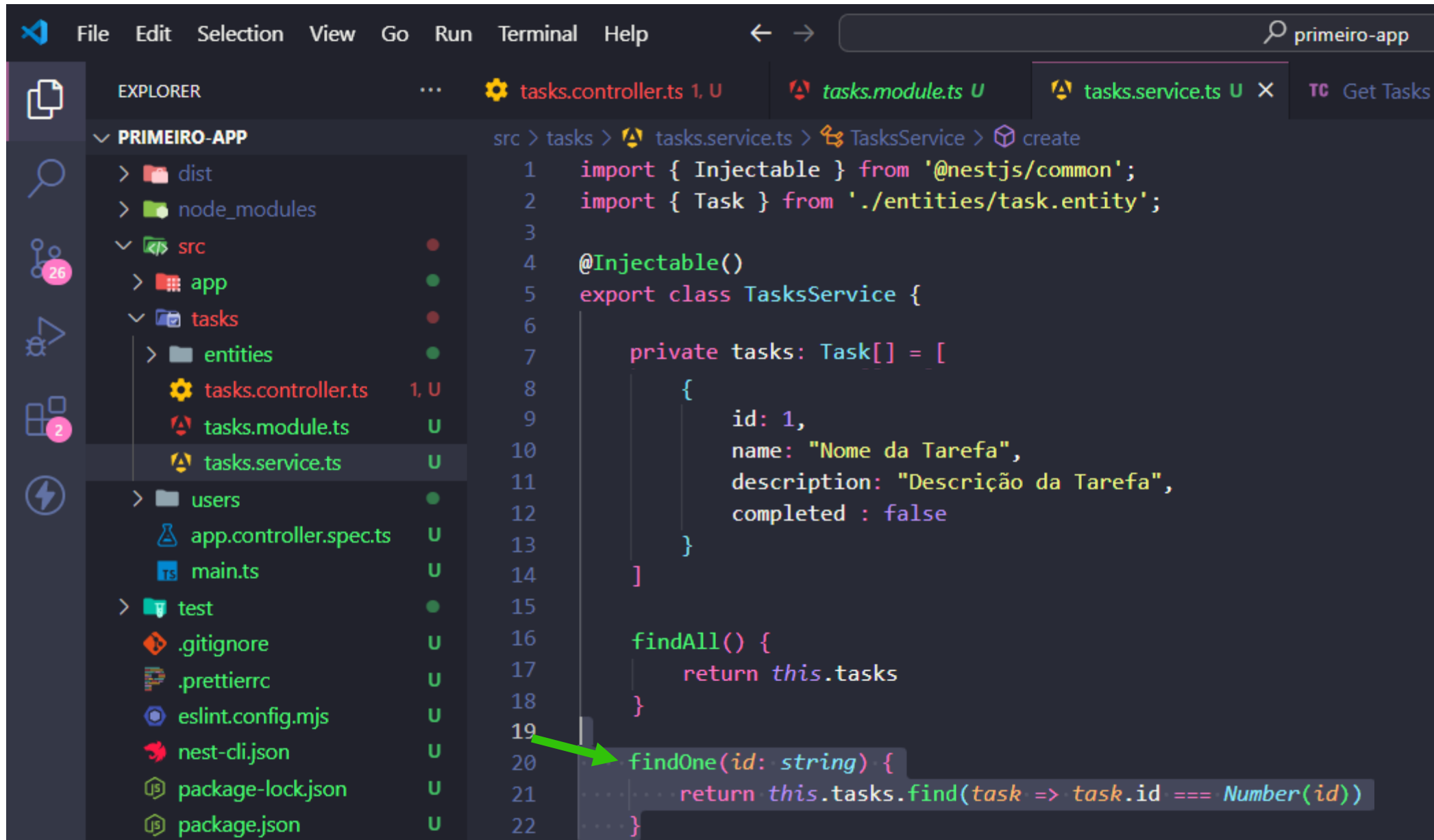
Response:

```
[
  {
    "id": 1,
    "name": "Nome da Tarefa",
    "description": "Descrição da Tarefa",
    "completed": false
  }
]
```

The terminal output shows the application startup logs:

```
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/tasks/:id, GET} route +0ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/tasks, POST} route +1ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/tasks/:id, PATCH} route +0ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/tasks/:id, DELETE} route +0ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RoutesResolver] UsersController {/users}: +1ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/users, POST} route +0ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/users, GET} route +0ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/users/:id, GET} route +0ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/users/:id, PATCH} route +1ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [RouterExplorer] Mapped {/users/:id, DELETE} route +0ms
[Nest] 11092 - 21/02/2025, 16:44:37 LOG [NestApplication] Nest application successfully started +2ms
```

Listando Itens



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

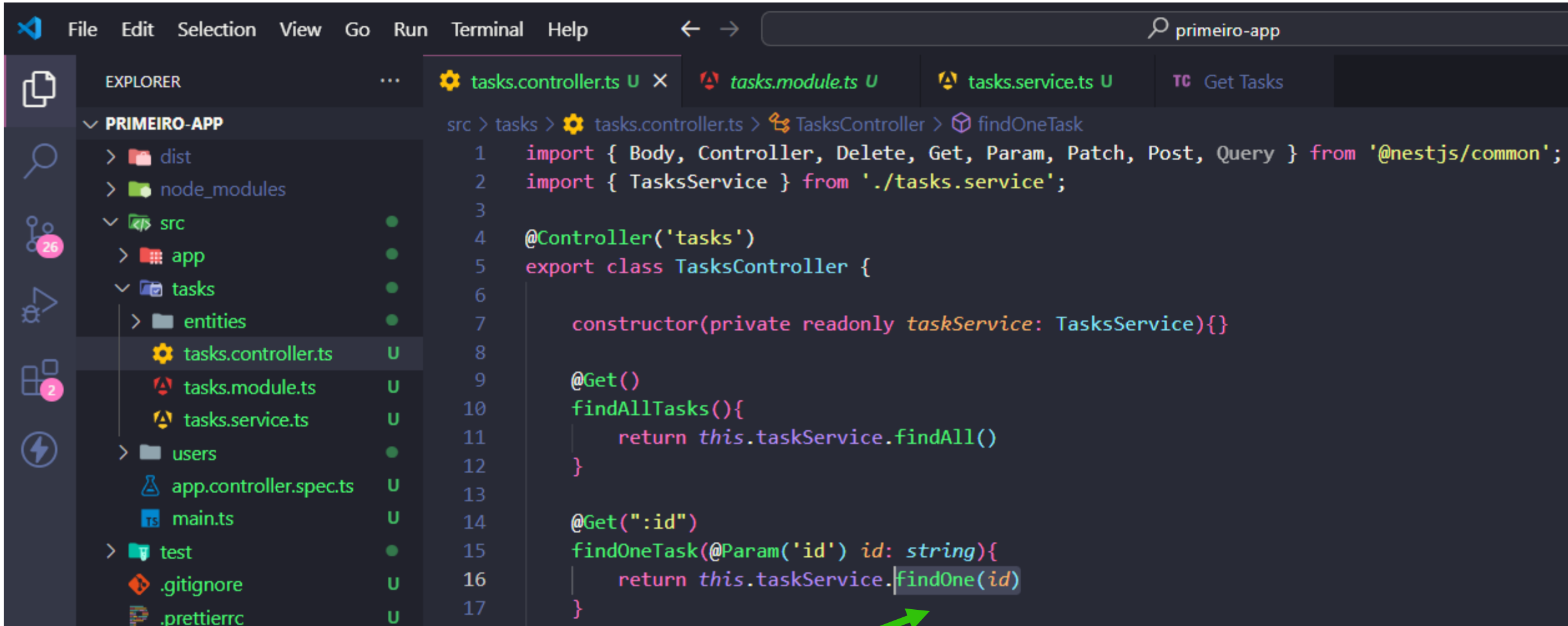
- Explorer Panel:** Displays the file structure of the project 'PRIMEIRO-APP'. The 'tasks' folder is expanded, showing 'entities', 'tasks.controller.ts', 'tasks.module.ts', and 'tasks.service.ts'.
- Editor Panel:** Shows the code for 'tasks.service.ts'. The breadcrumb navigation at the top indicates the current path: 'src > tasks > tasks.service.ts > TasksService > create'.
- Code Content:**

```

1  import { Injectable } from '@nestjs/common';
2  import { Task } from '../entities/task.entity';
3
4  @Injectable()
5  export class TasksService {
6
7      private tasks: Task[] = [
8          {
9              id: 1,
10             name: "Nome da Tarefa",
11             description: "Descrição da Tarefa",
12             completed: false
13         }
14     ]
15
16     findAll() {
17         return this.tasks
18     }
19
20     findOne(id: string) {
21         return this.tasks.find(task => task.id === Number(id))
22     }

```
- Annotations:** A green arrow points to the 'findOne' method definition on line 20.

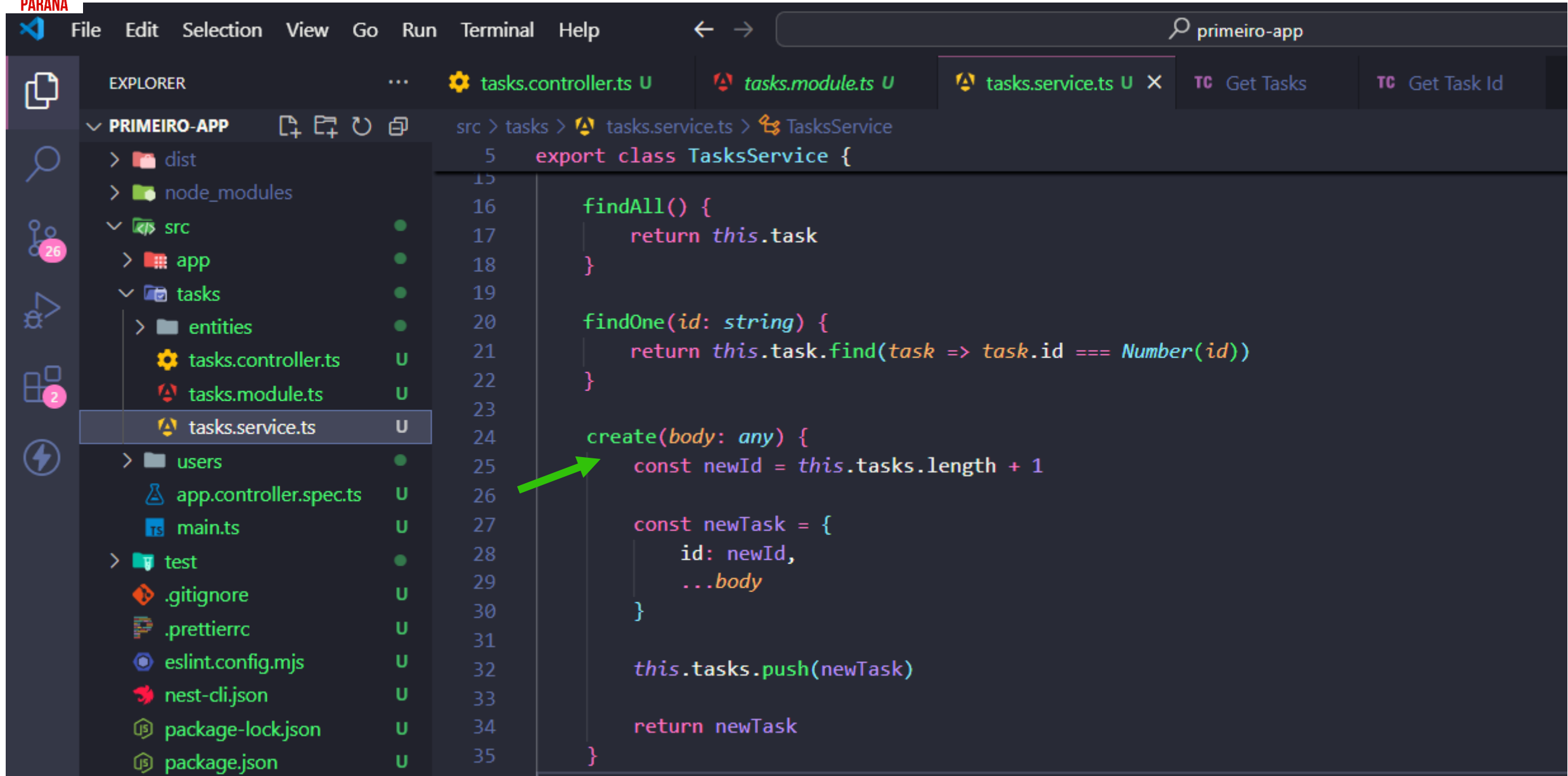
Listando Itens



The screenshot shows the Visual Studio Code interface with a project named 'primeiro-app'. The Explorer sidebar on the left displays the file structure, including folders like 'dist', 'node_modules', 'src', 'app', 'tasks', 'entities', 'users', and files like 'tasks.controller.ts', 'tasks.module.ts', 'tasks.service.ts', 'app.controller.spec.ts', 'main.ts', 'test', '.gitignore', and '.prettierrc'. The main editor area shows the 'tasks.controller.ts' file, which is part of the 'tasks' module. The code defines a 'TasksController' class with a constructor that takes a 'taskService' and two methods: 'findAllTasks()' and 'findOneTask()'. The 'findOneTask()' method is highlighted with a green arrow pointing to the 'findOne' call.

```
src > tasks > tasks.controller.ts > TasksController > findOneTask
1  import { Body, Controller, Delete, Get, Param, Patch, Post, Query } from '@nestjs/common';
2  import { TaskService } from './tasks.service';
3
4  @Controller('tasks')
5  export class TasksController {
6
7      constructor(private readonly taskService: TaskService){}
8
9      @Get()
10     findAllTasks(){
11         return this.taskService.findAll()
12     }
13
14     @Get(":id")
15     findOneTask(@Param('id') id: string){
16         return this.taskService.findOne(id)
17     }
18 }
```

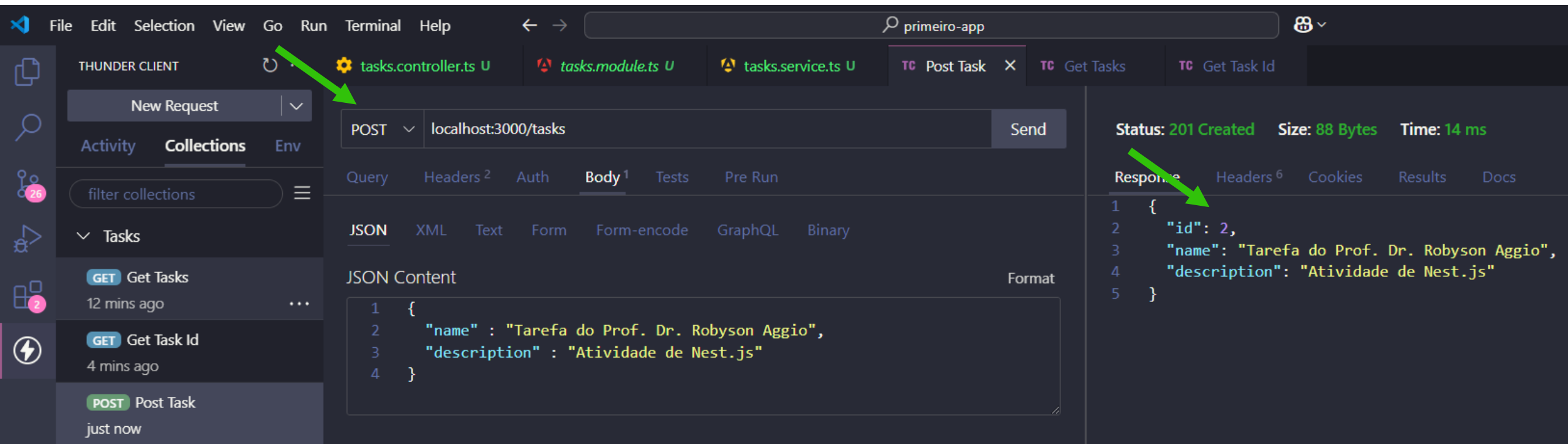
Listando Itens



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the file structure of a project named 'PRIMEIRO-APP'. The file 'tasks.service.ts' is selected and highlighted. The main editor area shows the content of 'tasks.service.ts', which defines a 'TasksService' class with methods 'findAll()', 'findOne()', and 'create()'. A green arrow points to the 'create' method, specifically to the line 'const newId = this.tasks.length + 1'.

```
5  export class TasksService {
15
16      findAll() {
17          return this.task
18      }
19
20      findOne(id: string) {
21          return this.task.find(task => task.id === Number(id))
22      }
23
24      create(body: any) {
25          const newId = this.tasks.length + 1
26
27          const newTask = {
28              id: newId,
29              ...body
30          }
31
32          this.tasks.push(newTask)
33
34          return newTask
35      }
36  }
```

Listando Itens



The screenshot displays the Thunder Client interface. The top bar shows the menu (File, Edit, Selection, View, Go, Run, Terminal, Help) and the current workspace (primeiro-app). The left sidebar contains a 'THUNDER CLIENT' section with a 'New Request' button and a 'Collections' tab. Below this, a list of tasks is shown: 'GET Get Tasks' (12 mins ago), 'GET Get Task Id' (4 mins ago), and 'POST Post Task' (just now). A green arrow points from the 'POST Post Task' entry to the main editor area.

The main editor area shows a REST client request configuration. The method is 'POST' and the URL is 'localhost:3000/tasks'. The 'Body' tab is selected, showing a JSON content editor with the following content:

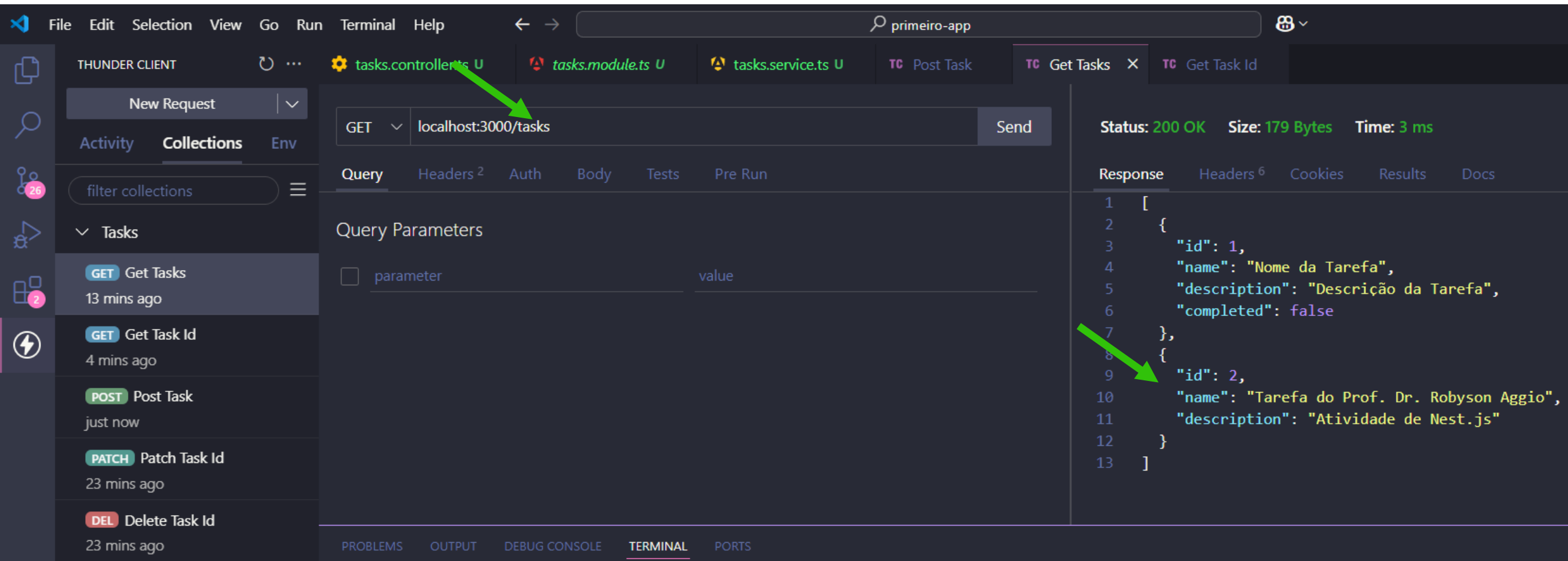
```
1 {
2   "name" : "Tarefa do Prof. Dr. Robyson Aggio",
3   "description" : "Atividade de Nest.js"
4 }
```

The right sidebar shows the response details. The status is '201 Created', the size is '88 Bytes', and the time is '14 ms'. The 'Response' tab is selected, showing the following JSON content:

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

A green arrow points from the 'Response' tab to the JSON content.

Listando Itens



The screenshot shows the Thunder Client interface. The top bar includes the menu (File, Edit, Selection, View, Go, Run, Terminal, Help) and a search bar with the text "primeiro-app". Below the menu, there are tabs for "tasks.controllers.ts U", "tasks.module.ts U", "tasks.service.ts U", "TC Post Task", "TC Get Tasks", and "TC Get Task Id". The "TC Get Tasks" tab is active.

The left sidebar shows the "Collections" view with a "filter collections" input and a list of tasks:

- GET Get Tasks (13 mins ago)
- GET Get Task Id (4 mins ago)
- POST Post Task (just now)
- PATCH Patch Task Id (23 mins ago)
- DEL Delete Task Id (23 mins ago)

The main area displays the details of the selected "GET" request to "localhost:3000/tasks". The "Query" tab is active, showing "Query Parameters" with a table:

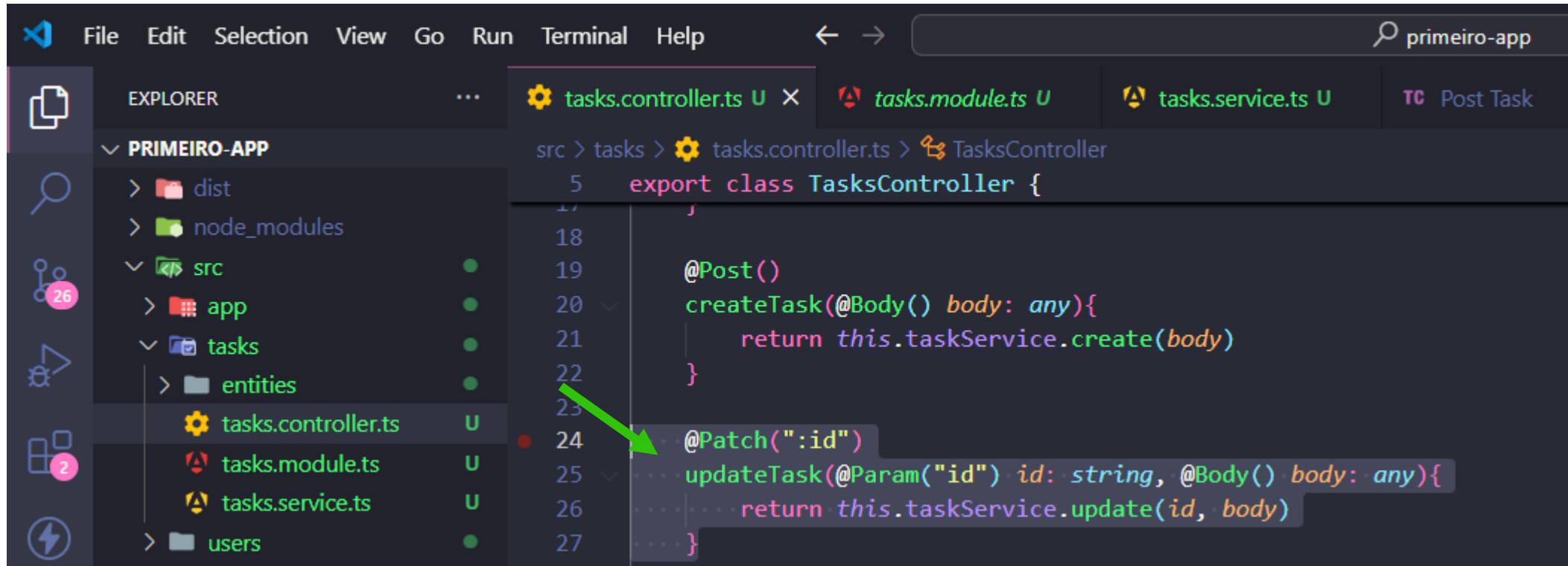
parameter	value
☐	

The "Response" tab is also active, showing the status "200 OK", size "179 Bytes", and time "3 ms". The response body is a JSON array:

```
[
  {
    "id": 1,
    "name": "Nome da Tarefa",
    "description": "Descrição da Tarefa",
    "completed": false
  },
  {
    "id": 2,
    "name": "Tarefa do Prof. Dr. Robyson Aggio",
    "description": "Atividade de Nest.js"
  }
]
```

Two green arrows are present: one pointing to the "localhost:3000/tasks" URL in the request bar, and another pointing to the second object in the JSON response array.

Atualizando Itens

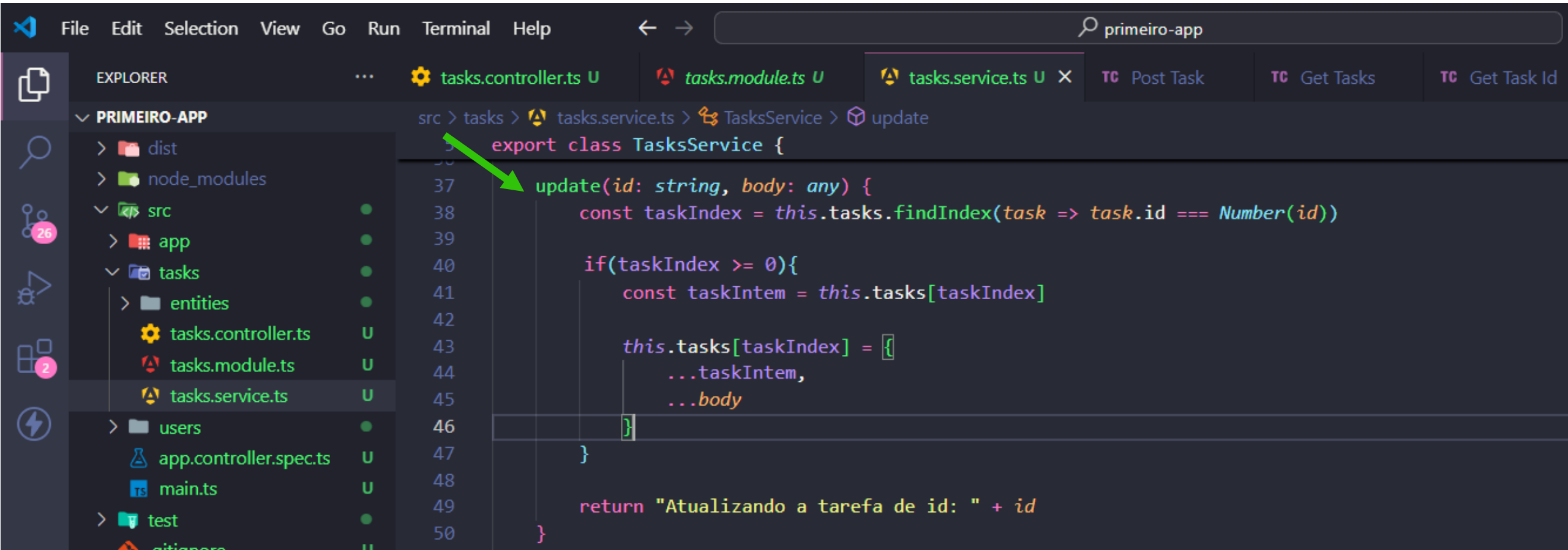


The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure for 'PRIMEIRO-APP', including folders like 'dist', 'node_modules', 'src', 'app', 'tasks', 'entities', and 'users'. The 'tasks' folder is expanded, showing files 'tasks.controller.ts', 'tasks.module.ts', and 'tasks.service.ts'. The main editor area displays the 'tasks.controller.ts' file, which contains the following TypeScript code:

```
5 export class TasksController {  
17  
18  
19     @Post()  
20     createTask(@Body() body: any){  
21         return this.taskService.create(body)  
22     }  
23  
24     @Patch(":id")  
25     updateTask(@Param("id") id: string, @Body() body: any){  
26         return this.taskService.update(id, body)  
27     }  
28 }
```

A green arrow points to the `@Patch(":id")` decorator on line 24, indicating the endpoint for updating tasks.

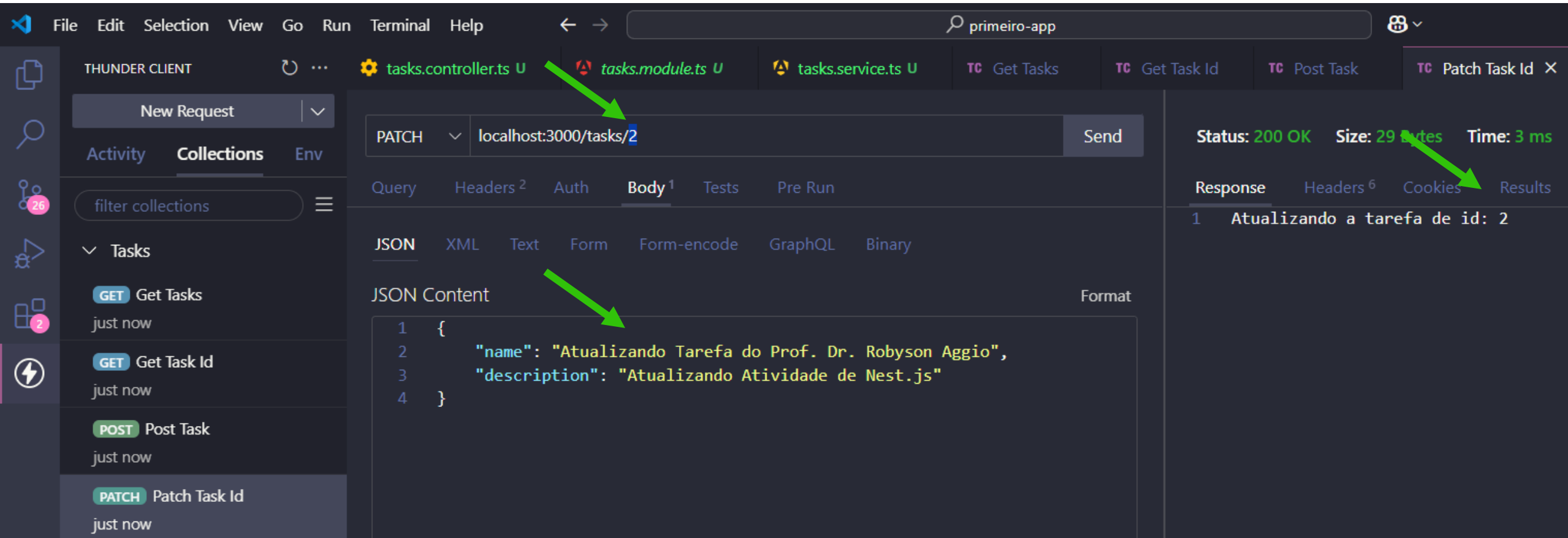
Atualizando Itens



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure for 'PRIMEIRO-APP', with the 'tasks' folder expanded and 'tasks.service.ts' selected. The breadcrumb navigation at the top indicates the current file path: 'src > tasks > tasks.service.ts > TasksService > update'. The main editor area shows the TypeScript code for the 'update' method, with a green arrow pointing to the method signature. The code includes a 'findIndex' call, an 'if' statement to check for a valid task index, and an object spread to update the task's body.

```
export class TasksService {  
  37 update(id: string, body: any) {  
    38     const taskIndex = this.tasks.findIndex(task => task.id === Number(id))  
    39  
    40     if(taskIndex >= 0){  
    41         const taskIntem = this.tasks[taskIndex]  
    42  
    43         this.tasks[taskIndex] = {  
    44             ...taskIntem,  
    45             ...body  
    46         }  
    47     }  
    48  
    49     return "Atualizando a tarefa de id: " + id  
    50 }  
}
```

Atualizando Itens



The screenshot displays the Thunder Client interface with a PATCH request configured. The URL is `localhost:3000/tasks/2`. The request body is a JSON object: `{ "name": "Atualizando Tarefa do Prof. Dr. Robyson Aggio", "description": "Atualizando Atividade de Nest.js" }`. The response status is `200 OK`, with a size of `29 bytes` and a time of `3 ms`. The response body is `1 Atualizando a tarefa de id: 2`.

Thunder Client

File Edit Selection View Go Run Terminal Help

primeiro-app

tasks.controller.ts U tasks.module.ts U tasks.service.ts U TC Get Tasks TC Get Task Id TC Post Task TC Patch Task Id X

New Request

Activity Collections Env

filter collections

Tasks

GET Get Tasks just now

GET Get Task Id just now

POST Post Task just now

PATCH Patch Task Id just now

PATCH localhost:3000/tasks/2 Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

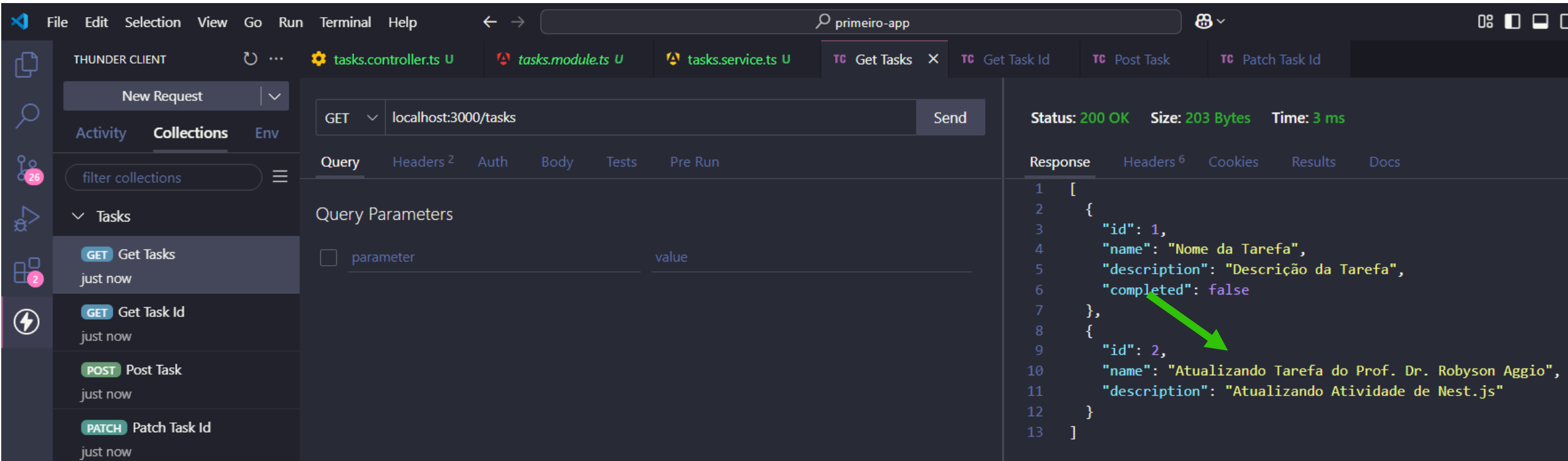
```
1 {
2   "name": "Atualizando Tarefa do Prof. Dr. Robyson Aggio",
3   "description": "Atualizando Atividade de Nest.js"
4 }
```

Status: 200 OK Size: 29 bytes Time: 3 ms

Response Headers 6 Cookies Results

1 Atualizando a tarefa de id: 2

Atualizando Itens



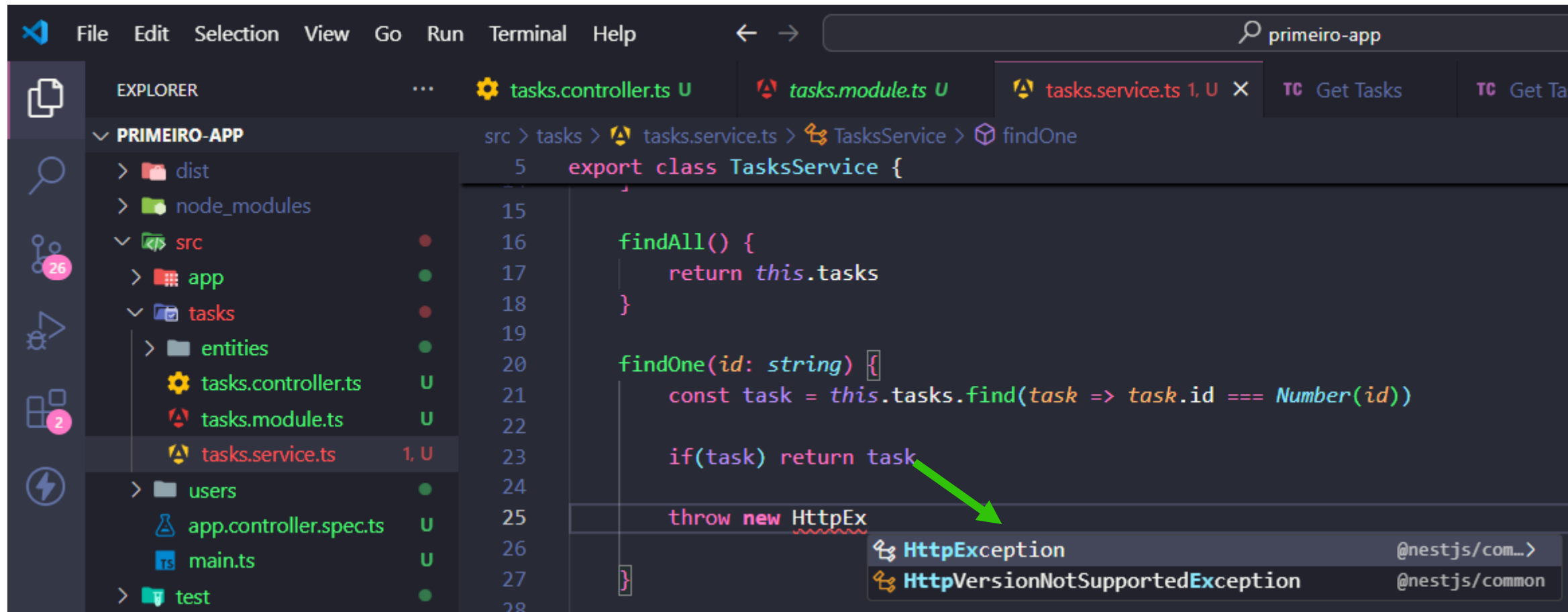
The screenshot shows the Thunder Client interface with a REST client request and response. The request is a GET request to `localhost:3000/tasks`. The response is a 200 OK status with a JSON body containing an array of tasks. A green arrow points to the second task in the array, which is the one being updated.

Thunder Client Interface:

- Top Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help. Search bar: `primeiro-app`.
- Left Sidebar:** THUNDER CLIENT, New Request, Activity, Collections, Env, filter collections, Tasks, GET Get Tasks, GET Get Task Id, POST Post Task, PATCH Patch Task Id.
- Request Panel:** GET `localhost:3000/tasks`. Send button.
- Query Parameters:** parameter, value.
- Response Panel:** Status: 200 OK, Size: 203 Bytes, Time: 3 ms. Response body:

```
[
  {
    "id": 1,
    "name": "Nome da Tarefa",
    "description": "Descrição da Tarefa",
    "completed": false
  },
  {
    "id": 2,
    "name": "Atualizando Tarefa do Prof. Dr. Robyson Aggio",
    "description": "Atualizando Atividade de Nest.js"
  }
]
```

Tratando erros



The screenshot shows the Visual Studio Code editor with a project named 'primeiro-app'. The Explorer panel on the left shows the file structure, with 'tasks.service.ts' selected under the 'tasks' folder. The main editor displays the code for 'tasks.service.ts', which includes a 'findOne' method. A red squiggly line under 'HttpEx' indicates an error. A green arrow points to the error message dropdown, which lists 'HttpException' and 'HttpVersionNotSupportedException'.

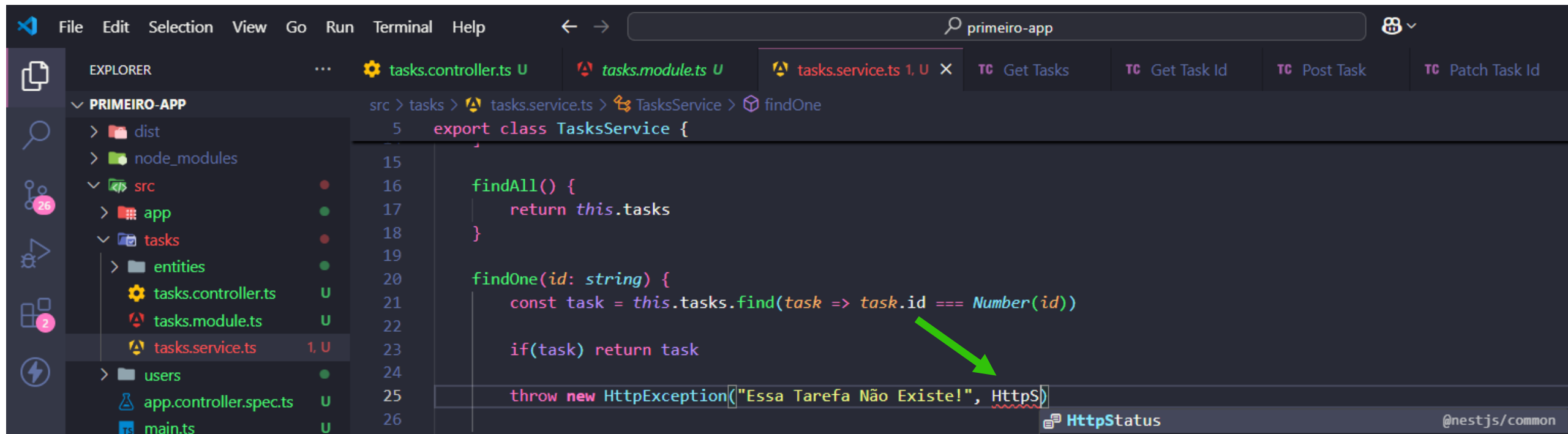
```

5  export class TaskService {
15
16  findAll() {
17    return this.tasks
18  }
19
20  findOne(id: string) {
21    const task = this.tasks.find(task => task.id === Number(id))
22
23    if(task) return task
24
25    throw new HttpEx
26
27  }
28

```

HttpException @nestjs/com...
HttpVersionNotSupportedException @nestjs/common

Tratando erros

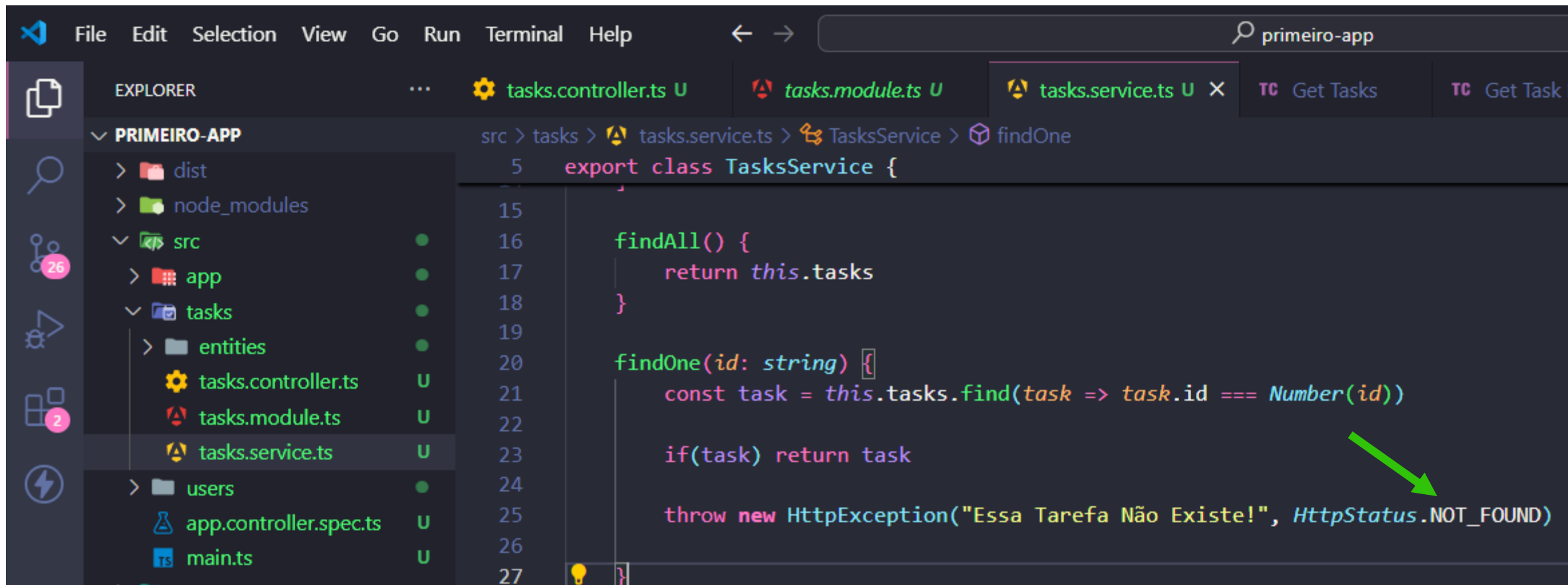


The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays the project structure for 'PRIMEIRO-APP', including folders like 'dist', 'node_modules', 'src', 'app', 'tasks', 'entities', and 'users'. The 'tasks' folder is expanded, showing files 'tasks.controller.ts', 'tasks.module.ts', and 'tasks.service.ts'. The 'tasks.service.ts' file is selected and open in the editor. The breadcrumb navigation at the top indicates the current file path: 'src > tasks > tasks.service.ts > TasksService > findOne'. The editor displays the following TypeScript code:

```
5 export class TasksService {  
15  
16   findAll() {  
17     return this.tasks  
18   }  
19  
20   findOne(id: string) {  
21     const task = this.tasks.find(task => task.id === Number(id))  
22  
23     if(task) return task  
24  
25     throw new HttpException("Essa Tarefa Não Existe!", HttpStatus) 26  
26   }
```

A green arrow points to the `HttpStatus` property access in the `throw new HttpException` line, which is underlined in red, indicating an error. A tooltip for `HttpStatus` is visible at the bottom right, showing the import path `@nestjs/common`.

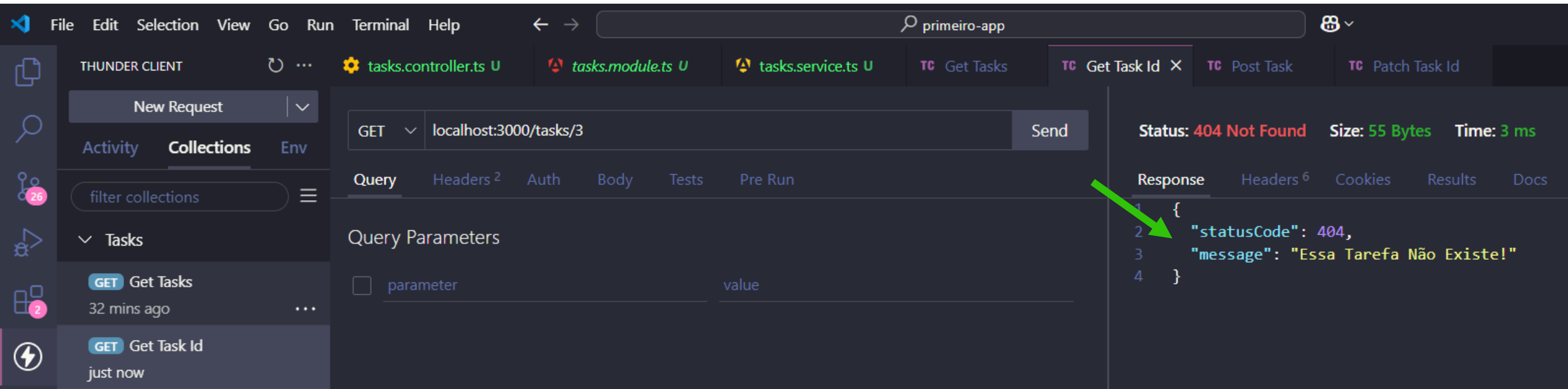
Tratando erros



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', 'app', 'tasks', 'entities', and 'users'. The 'tasks' folder is expanded, showing files 'tasks.controller.ts', 'tasks.module.ts', and 'tasks.service.ts'. The 'tasks.service.ts' file is open in the editor, showing the 'TasksService' class with methods 'findAll()' and 'findOne(id: string)'. The 'findOne' method contains a call to 'this.tasks.find' with a callback function. A green arrow points to the 'Number(id)' argument in the callback, which is highlighted with a yellow background, indicating a TypeScript error. The error message at the bottom of the editor reads: 'Argument of type 'string' is not assignable to parameter of type 'number''.

```
5 export class TasksService {  
15  
16     findAll() {  
17         return this.tasks  
18     }  
19  
20     findOne(id: string) {  
21         const task = this.tasks.find(task => task.id === Number(id))  
22  
23         if(task) return task  
24  
25         throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)  
26  
27     }  
}
```

Tratando erros



The screenshot shows the Thunder Client interface. The top bar includes the Thunder Client logo, menu items (File, Edit, Selection, View, Go, Run, Terminal, Help), a search bar with 'primeiro-app', and a user profile icon. Below the top bar, there's a tab bar with several tabs: 'tasks.controller.ts U', 'tasks.module.ts U', 'tasks.service.ts U', 'TC Get Tasks', 'TC Get Task Id X', 'TC Post Task', and 'TC Patch Task Id'. The 'TC Get Task Id' tab is selected. The main area shows a request details view for a GET request to 'localhost:3000/tasks/3'. The 'Query' tab is active, showing 'Query Parameters' with a table with columns 'parameter' and 'value'. The 'Response' tab is also visible, showing the response details: 'Status: 404 Not Found', 'Size: 55 Bytes', and 'Time: 3 ms'. The response body is displayed as JSON:

```
{  "statusCode": 404,  "message": "Essa Tarefa Não Existe!"}
```

. A green arrow points from the 'Response' tab to the JSON body.

THUNDER CLIENT

File Edit Selection View Go Run Terminal Help

primeiro-app

tasks.controller.ts U tasks.module.ts U tasks.service.ts U TC Get Tasks TC Get Task Id X TC Post Task TC Patch Task Id

New Request

Activity Collections Env

filter collections

Tasks

GET Get Tasks 32 mins ago

GET Get Task Id just now

GET localhost:3000/tasks/3 Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

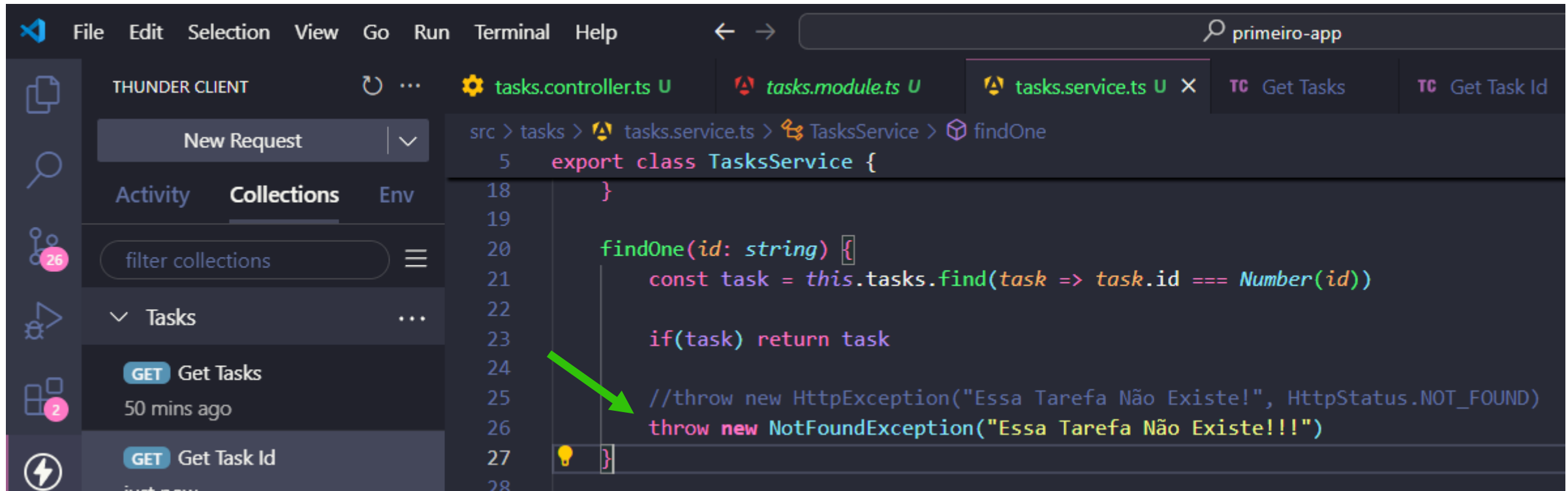
parameter	value

Status: 404 Not Found Size: 55 Bytes Time: 3 ms

Response Headers 6 Cookies Results Docs

```
{  "statusCode": 404,  "message": "Essa Tarefa Não Existe!"}
```

Tratando erros

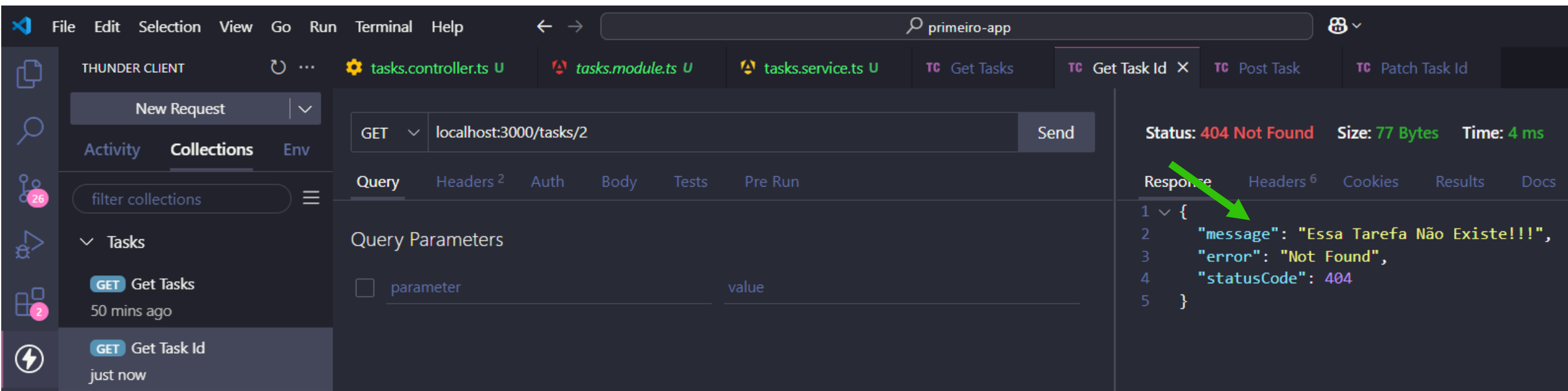


The screenshot shows the Visual Studio Code editor with a TypeScript file named `tasks.service.ts` open. The file is part of a project named `primeiro-app`. The editor displays the following code:

```
src > tasks > tasks.service.ts > TasksService > findOne
5  export class TasksService {
18      }
19
20      findOne(id: string) {
21          const task = this.tasks.find(task => task.id === Number(id))
22
23          if(task) return task
24
25          //throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)
26          throw new NotFoundException("Essa Tarefa Não Existe!!!")
27      }
28  }
```

A green arrow points to the line `throw new NotFoundException("Essa Tarefa Não Existe!!!")`, which is highlighted in yellow. This line is the source of a TypeScript error, indicated by a yellow lightbulb icon in the left margin. The error message is "Property 'NotFoundException' does not exist on type 'typeof import(\"../tasks\")'".

Tratando erros



The screenshot shows the Thunder Client interface. The top bar includes the menu (File, Edit, Selection, View, Go, Run, Terminal, Help), a search bar with 'primeiro-app', and a workspace tab bar with 'tasks.controller.ts', 'tasks.module.ts', 'tasks.service.ts', and several task-related tabs. The left sidebar shows a 'Collections' view with a 'Tasks' collection containing 'Get Tasks' and 'Get Task Id'. The main panel displays a GET request to 'localhost:3000/tasks/2'. The 'Response' tab is active, showing a 404 Not Found status with a message: 'Essa Tarefa Não Existe!!!'. A green arrow points to the 'message' field in the JSON response.

THUNDER CLIENT

File Edit Selection View Go Run Terminal Help

primeiro-app

tasks.controller.ts tasks.module.ts tasks.service.ts TC Get Tasks TC Get Task Id X TC Post Task TC Patch Task Id

New Request

Activity Collections Env

filter collections

Tasks

GET Get Tasks 50 mins ago

GET Get Task Id just now

GET localhost:3000/tasks/2 Send

Query Headers² Auth Body Tests Pre Run

Query Parameters

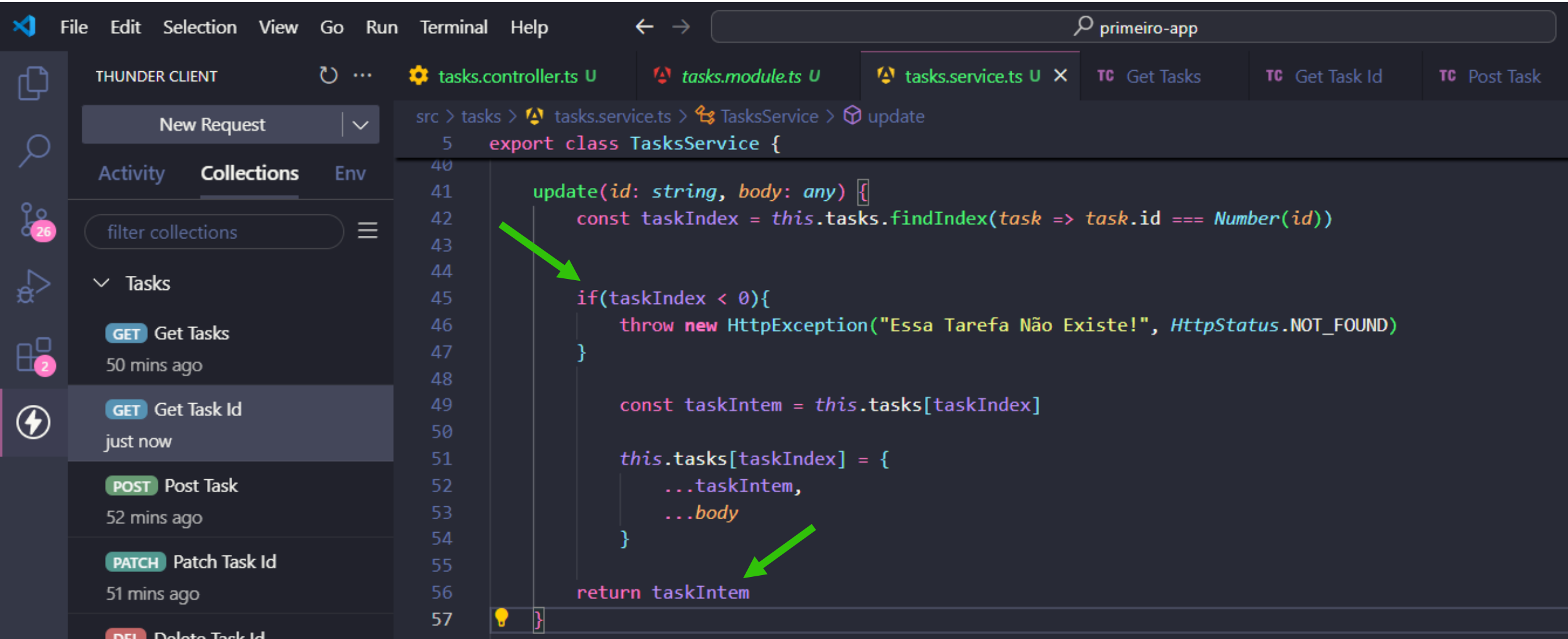
parameter value

Status: 404 Not Found Size: 77 Bytes Time: 4 ms

Response Headers⁶ Cookies Results Docs

```
1 {
2   "message": "Essa Tarefa Não Existe!!!",
3   "error": "Not Found",
4   "statusCode": 404
5 }
```

Tratando erros

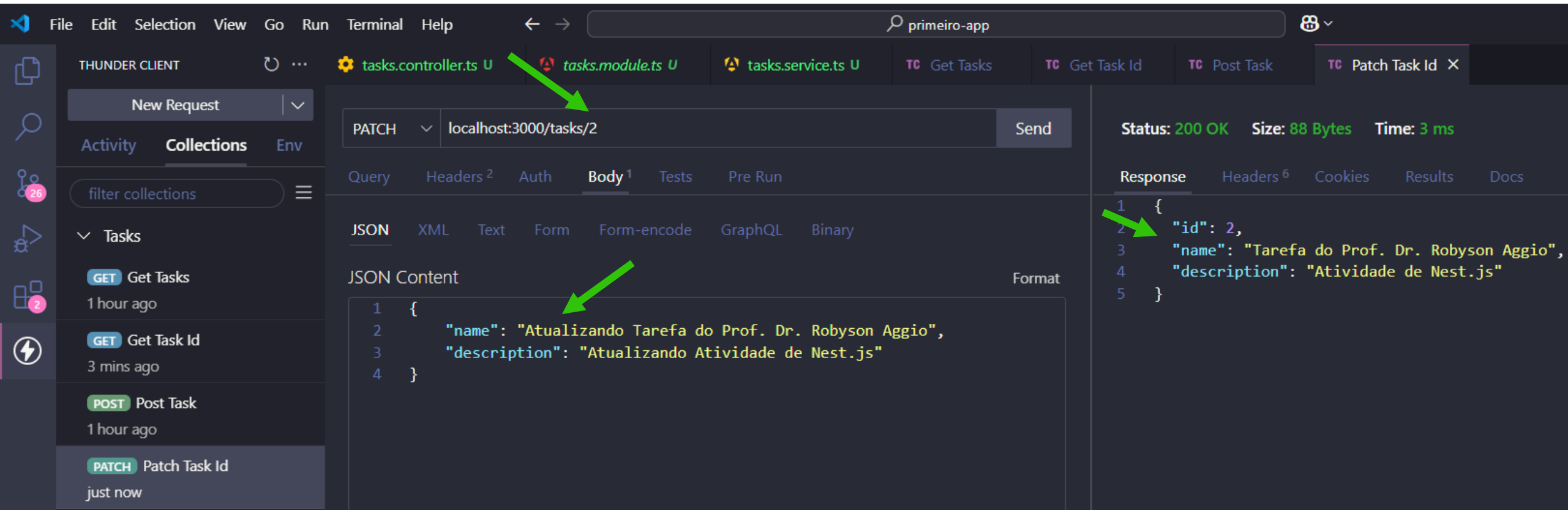


The screenshot shows the Visual Studio Code editor with the file `tasks.service.ts` open. The editor displays the following TypeScript code:

```
src > tasks > tasks.service.ts > TasksService > update
5  export class TasksService {
40
41  update(id: string, body: any) {
42      const taskIndex = this.tasks.findIndex(task => task.id === Number(id))
43
44      if(taskIndex < 0){
45          throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)
46      }
47
48      const taskIntem = this.tasks[taskIndex]
49
50      this.tasks[taskIndex] = {
51          ...taskIntem,
52          ...body
53      }
54
55      return taskIntem
56  }
57 }
```

Two green arrows highlight the error handling logic: one points to the `if(taskIndex < 0){` condition, and the other points to the `throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)` statement.

Tratando erros



THUNDER CLIENT

File Edit Selection View Go Run Terminal Help

primeiro-app

tasks.controller.ts U tasks.module.ts U tasks.service.ts U TC Get Tasks TC Get Task Id TC Post Task TC Patch Task Id X

New Request

Activity Collections Env

filter collections

Tasks

GET Get Tasks 1 hour ago

GET Get Task Id 3 mins ago

POST Post Task 1 hour ago

PATCH Patch Task Id just now

PATCH localhost:3000/tasks/2 Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content

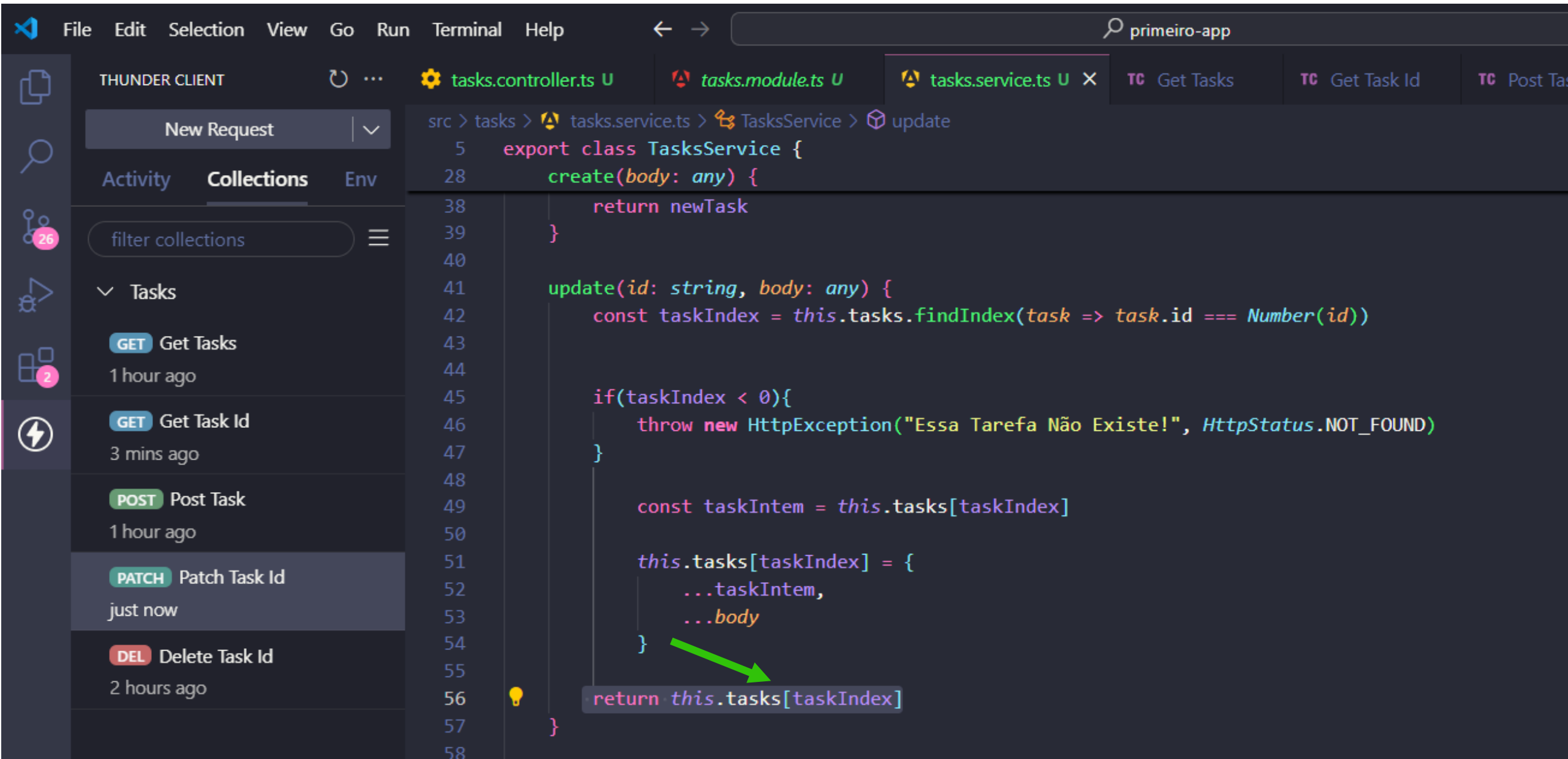
```
1 {
2   "name": "Atualizando Tarefa do Prof. Dr. Robyson Aggio",
3   "description": "Atualizando Atividade de Nest.js"
4 }
```

Status: 200 OK Size: 88 Bytes Time: 3 ms

Response Headers 6 Cookies Results Docs

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

Tratando erros

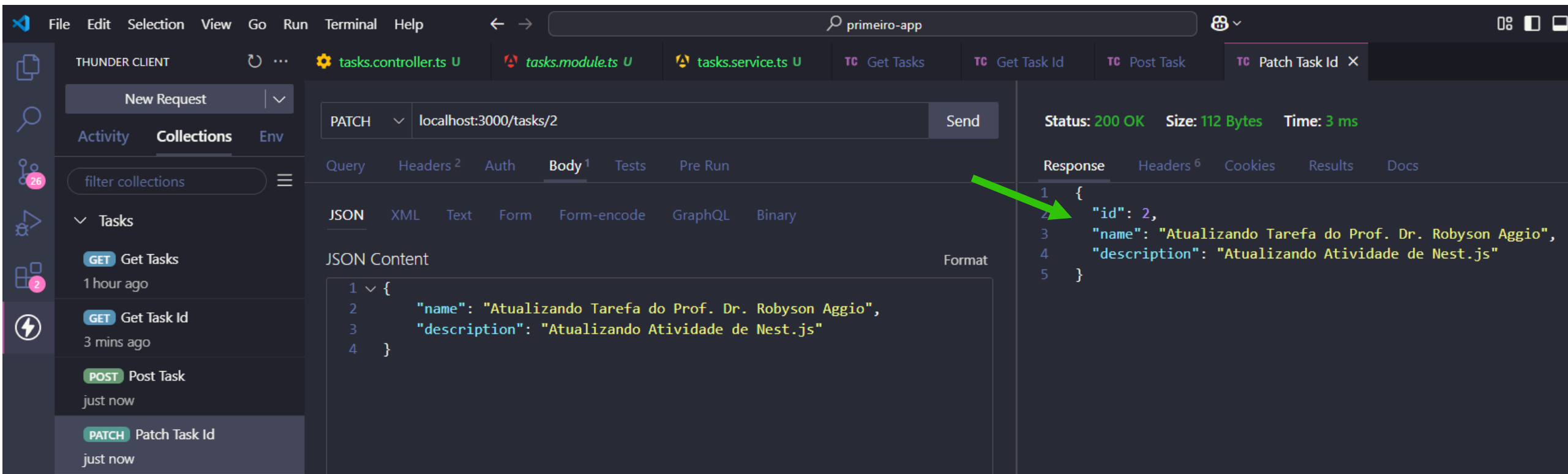


The screenshot shows the Visual Studio Code editor with a project named "primeiro-app". The left sidebar displays the "THUNDER CLIENT" with a list of API endpoints under the "Tasks" collection. The main editor area shows the file "tasks.service.ts" with the following code:

```
src > tasks > tasks.service.ts > TasksService > update
5  export class TasksService {
28  create(body: any) {
38      return newTask
39  }
40
41  update(id: string, body: any) {
42      const taskIndex = this.tasks.findIndex(task => task.id === Number(id))
43
44
45      if(taskIndex < 0){
46          throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)
47      }
48
49      const taskIntem = this.tasks[taskIndex]
50
51      this.tasks[taskIndex] = {
52          ...taskIntem,
53          ...body
54      }
55
56      return this.tasks[taskIndex]
57  }
58
```

A green arrow points to the closing curly brace of the `update` method on line 54, indicating a syntax error. A lightbulb icon is visible in the left margin next to line 56.

Tratando erros



THUNDER CLIENT

File Edit Selection View Go Run Terminal Help

primeiro-app

tasks.controller.ts U tasks.module.ts U tasks.service.ts U TC Get Tasks TC Get Task Id TC Post Task TC Patch Task Id X

New Request

Activity Collections Env

filter collections

Tasks

GET Get Tasks 1 hour ago

GET Get Task Id 3 mins ago

POST Post Task just now

PATCH Patch Task Id just now

PATCH localhost:3000/tasks/2 Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

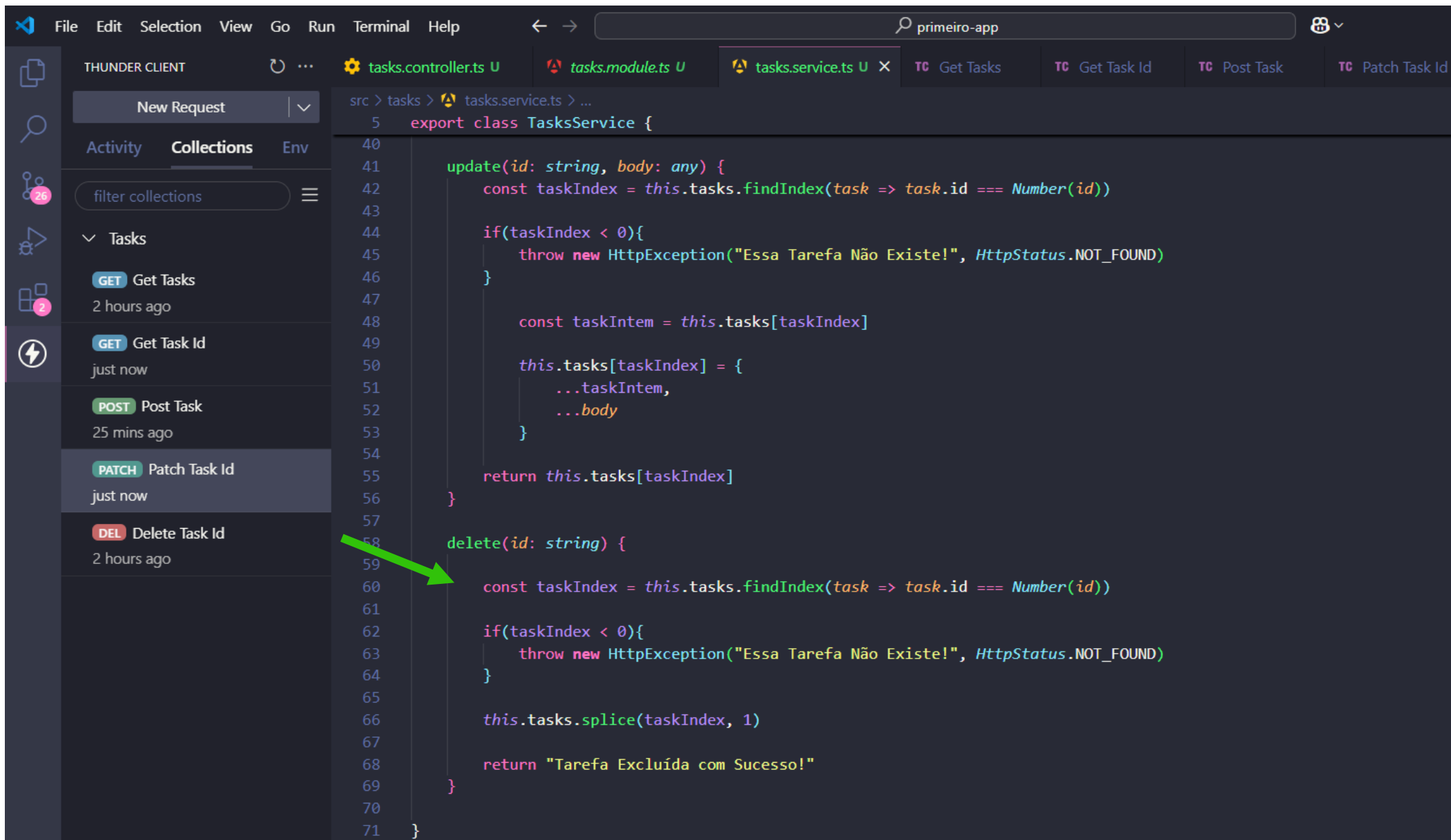
```
1 {  
2   "name": "Atualizando Tarefa do Prof. Dr. Robyson Aggio",  
3   "description": "Atualizando Atividade de Nest.js"  
4 }
```

Status: 200 OK Size: 112 Bytes Time: 3 ms

Response Headers 6 Cookies Results Docs

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

Delete

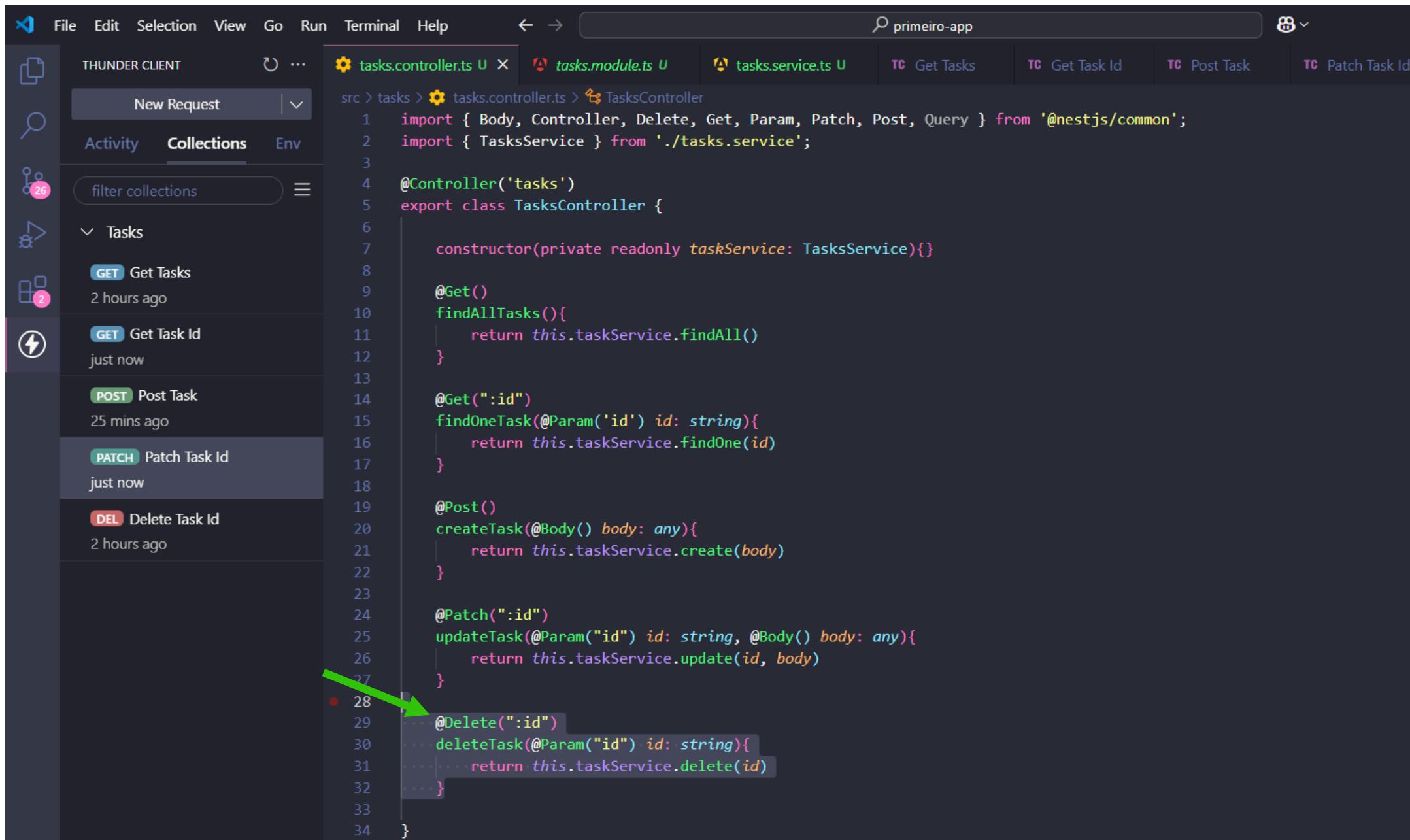


The screenshot shows the VS Code editor with the `tasks.service.ts` file open. The THUNDER CLIENT sidebar on the left displays a list of requests. The 'Delete Task Id' request, marked with a red 'DEL' icon, is selected. A green arrow points from this request to the `delete` method in the `TasksService` class.

```

src > tasks > tasks.service.ts > ...
5  export class TasksService {
40
41      update(id: string, body: any) {
42          const taskIndex = this.tasks.findIndex(task => task.id === Number(id))
43
44          if(taskIndex < 0){
45              throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)
46          }
47
48          const taskIntem = this.tasks[taskIndex]
49
50          this.tasks[taskIndex] = {
51              ...taskIntem,
52              ...body
53          }
54
55          return this.tasks[taskIndex]
56      }
57
58      delete(id: string) {
59
60          const taskIndex = this.tasks.findIndex(task => task.id === Number(id))
61
62          if(taskIndex < 0){
63              throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)
64          }
65
66          this.tasks.splice(taskIndex, 1)
67
68          return "Tarefa Excluída com Sucesso!"
69      }
70
71  }
  
```

Delete



The image shows a development environment with a REST client and a TypeScript controller. The REST client on the left lists several requests, with 'DELETE Task Id' highlighted. The TypeScript code on the right implements the controller logic, including a deleteTask method that is highlighted with a green arrow pointing to the @Delete decorator.

THUNDER CLIENT

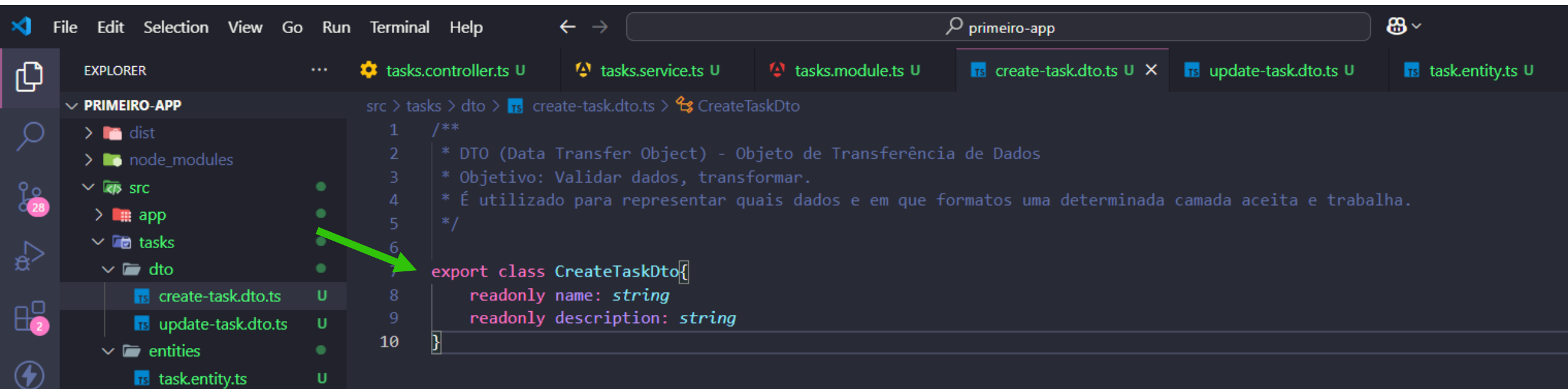
- New Request
- Activity
- Collections
- Env
- filter collections
- Tasks
 - GET Get Tasks (2 hours ago)
 - GET Get Task Id (just now)
 - POST Post Task (25 mins ago)
 - PATCH Patch Task Id (just now)
 - DELETE Delete Task Id (2 hours ago)**

tasks.controller.ts

```

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

Criando DTO

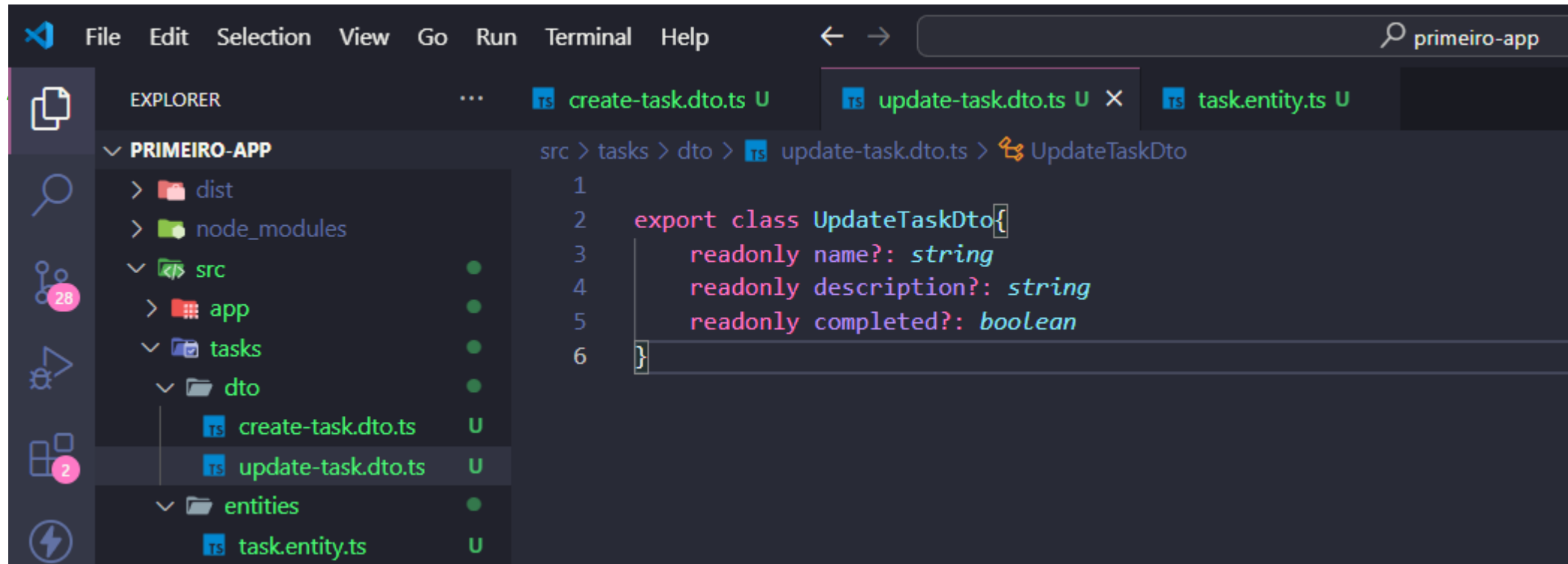


The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left and the Editor window on the right. The Explorer sidebar shows the project structure for 'PRIMEIRO-APP', with the 'dto' folder under 'tasks' selected. The Editor window shows the 'create-task.dto.ts' file, which contains the following code:

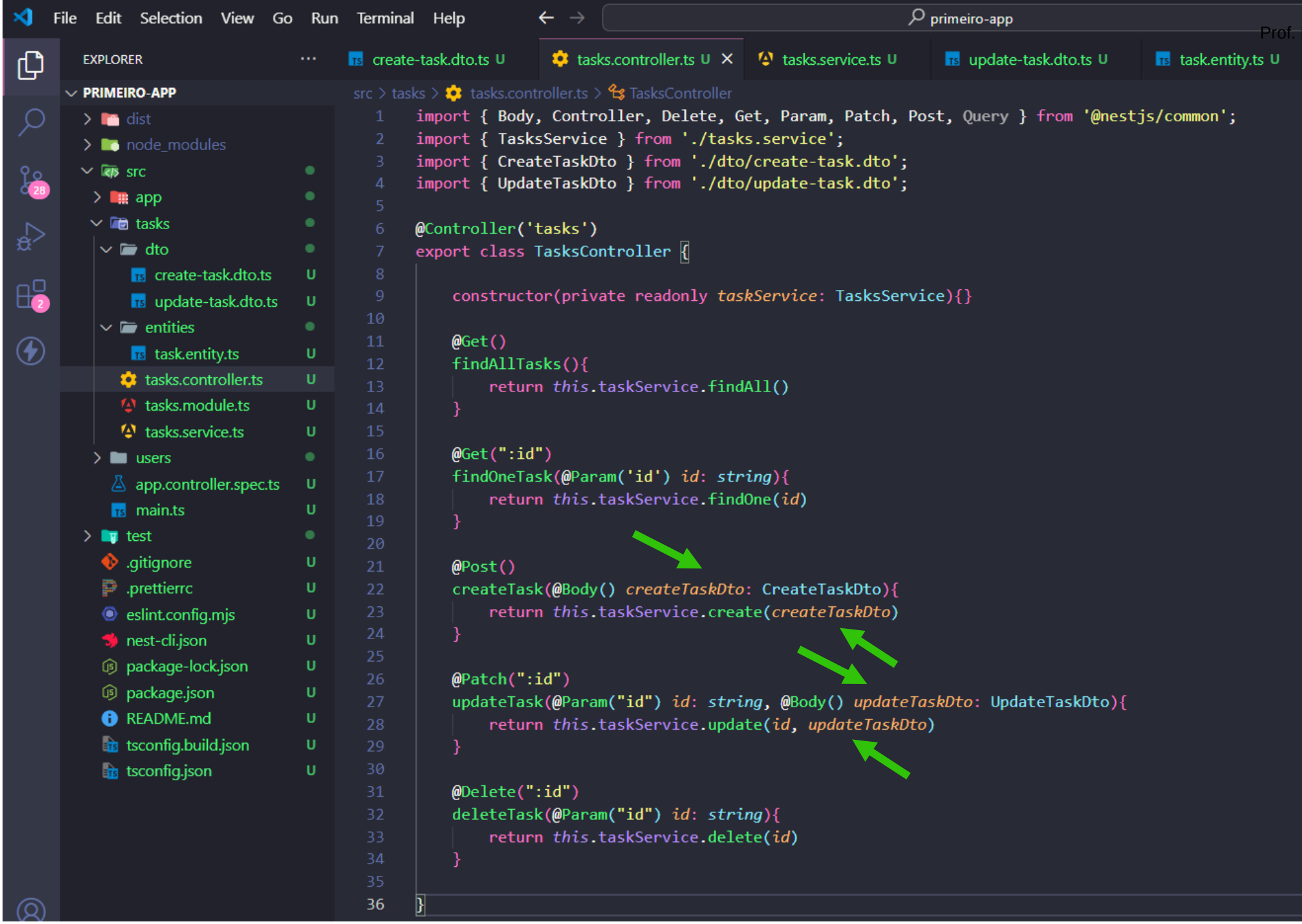
```
1  /**
2   * DTO (Data Transfer Object) - Objeto de Transferência de Dados
3   * Objetivo: Validar dados, transformar.
4   * É utilizado para representar quais dados e em que formatos uma determinada camada aceita e trabalha.
5   */
6
7  export class CreateTaskDto {
8      readonly name: string
9      readonly description: string
10 }
```

A green arrow points from the 'dto' folder in the Explorer sidebar to the 'CreateTaskDto' class definition in the Editor window.

Atualizando DTO



```
File Edit Selection View Go Run Terminal Help
EXPLORER
PRIMEIRO-APP
  dist
  node_modules
  src
    app
    tasks
      dto
        create-task.dto.ts
        update-task.dto.ts
    entities
      task.entity.ts
src > tasks > dto > update-task.dto.ts > UpdateTaskDto
1
2 export class UpdateTaskDto{
3   readonly name?: string
4   readonly description?: string
5   readonly completed?: boolean
6 }
```



```
File Edit Selection View Go Run Terminal Help
primeiro-app

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 > tasks > tasks.controller.ts > TasksController
1 import { Body, Controller, Delete, Get, Param, Patch, Post, Query } 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') id: string){
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") id: string, @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
36 }
```

File Edit Selection View Go Run Terminal Help

primeiro-app

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 > tasks > tasks.service.ts > TasksService > delete

```
7 export class TasksService {  
29  
30 create(createTaskDto: CreateTaskDto) {  
31     const newId = this.tasks.length + 1  
32  
33     const newTask = {  
34         id: newId,  
35         ...createTaskDto,  
36         completed: false  
37     }  
38  
39     this.tasks.push(newTask)  
40  
41     return newTask  
42 }  
43  
44 update(id: string, updateTaskDto: UpdateTaskDto) {  
45     const taskIndex = this.tasks.findIndex(task => task.id === Number(id))  
46  
47     if(taskIndex < 0){  
48         throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)  
49     }  
50  
51     const taskIntem = this.tasks[taskIndex]  
52  
53     this.tasks[taskIndex] = {  
54         ...taskIntem,  
55         ...updateTaskDto  
56     }  
57  
58     return this.tasks[taskIndex]  
59 }
```

Atividade

- 1) Dentro de cada camada: Guests, Users e Teachers, implemente os verbos: Post, Patch e Delete**
- 2) Crie a pasta entities para cada camada e a respectiva Entidade.**
- 3) Crie a pasta dto para cada camada e dentro os arquivos: create*Dto e update*Dto**
- 4) Teste todos os verbos HTTP (Get, Post, Patch, Delete)**