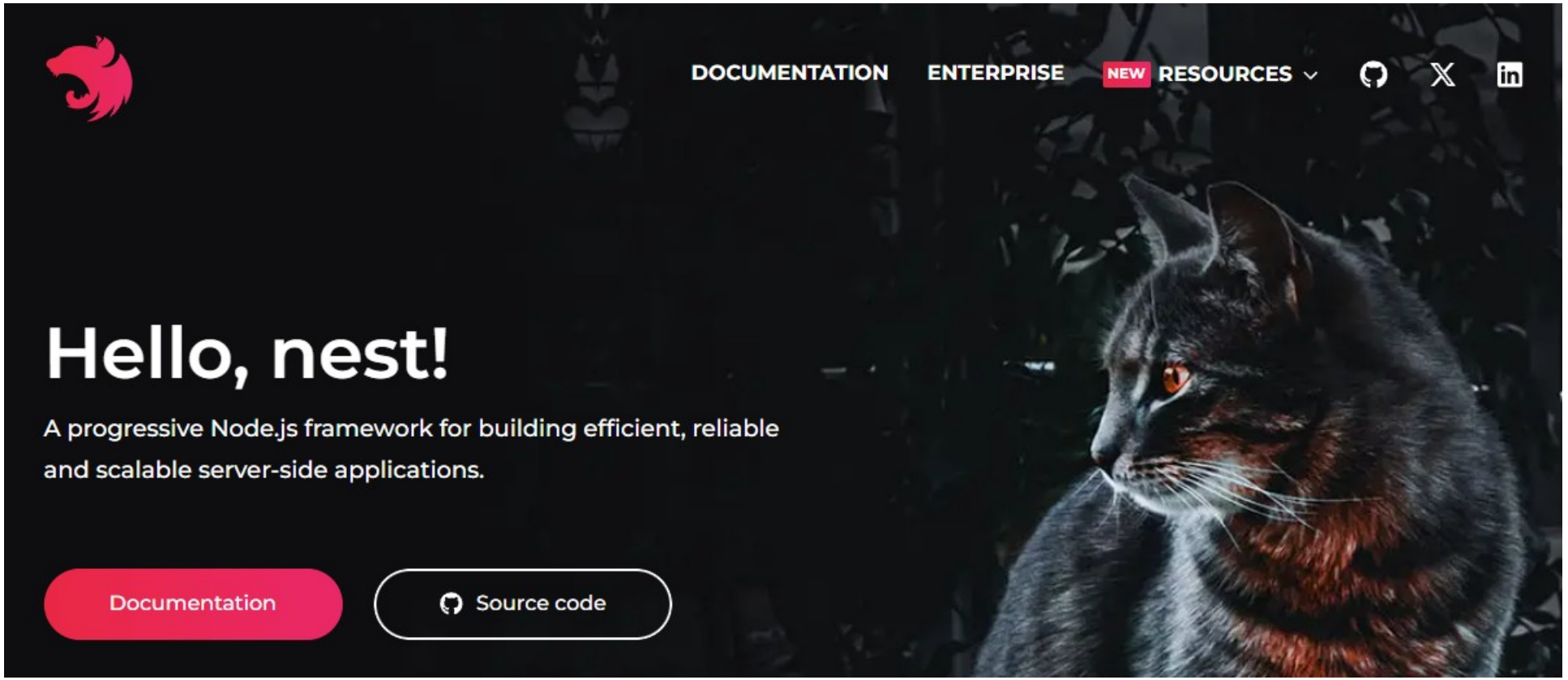
INSTITUTO FEDERAL
PARANÁ

Nest.js

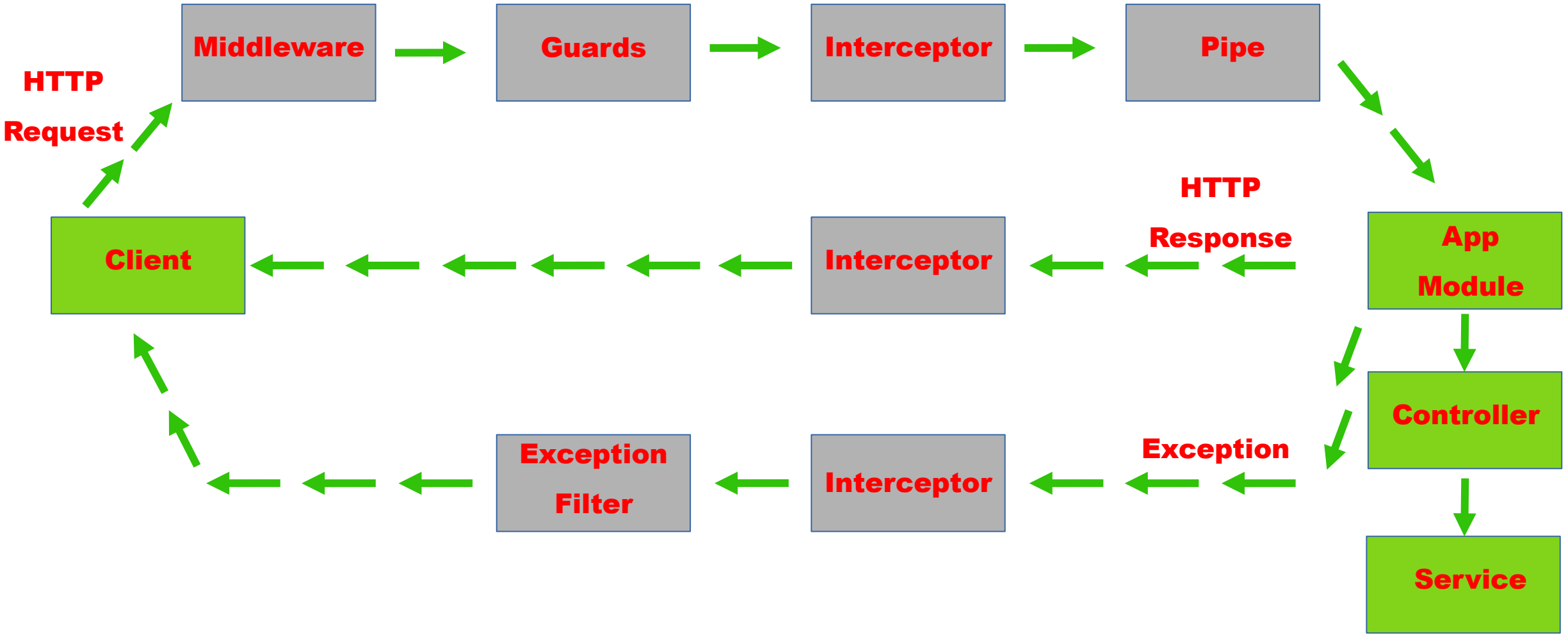


Prof. Dr. Robyson Aggio

www.aggioirati.com.br

Prof. Dr. Robyson Aggio

Estrutura



Prisma

Prisma is an open-source ORM for Node.js and TypeScript. It is used as an **alternative** to writing plain SQL, or using another database access tool such as SQL query builders (like knex.js) or ORMs (like TypeORM and Sequelize). Prisma currently supports PostgreSQL, MySQL, SQL Server, SQLite, MongoDB and CockroachDB (Preview).

While Prisma can be used with plain JavaScript, it embraces TypeScript and provides a level to type-safety that goes beyond the guarantees other ORMs in the TypeScript ecosystem. You can find an in-depth comparison of the type-safety guarantees of Prisma and TypeORM here.

> **NOTE**
> If you want to get a quick overview of how Prisma works, you can follow the Quickstart or read the Introduction in the documentation. There also are ready-to-run examples for REST and GraphQL in the `prisma-examples` repo.

Getting started

In this recipe, you'll learn how to get started with NestJS and Prisma from scratch. You are going to build a sample NestJS application with a REST API that can read and write data in a database.

For the purpose of this guide, you'll use a SQLite database to save the overhead of setting up a database server. Note that you can still follow this guide, even if you're using PostgreSQL or MySQL – you'll get extra instructions for using these databases at the right places.

> **NOTE**
> If you already have an existing project and consider migrating to Prisma, you can follow the guide for adding Prisma to an existing project. If you are migrating from TypeORM, you can read the guide Migrating from TypeORM to Prisma.

Create your NestJS project

To get started, install the NestJS CLI and create your app skeleton with the following commands:

Design and Development tips in your inbox. Every weekday.

ADS VIA CARBON

Prisma

Prof. Dr. Robyson Aggio

Prisma | NestJS - A progressive

docs.nestjs.com/recipes/prisma

Search COURSES ENTERPRISE NEW DEVTOOLS DEPLOY WITH MAU

Set up Prisma

Start by installing the Prisma CLI as a development dependency in your project:

```
$ cd hello-prisma
$ npm install prisma --save-dev
```

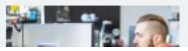
In the following steps, we'll be utilizing the Prisma CLI. As a best practice, it's recommended to invoke the CLI locally by prefixing it with `npx`:

```
$ npx prisma
```

▶ Expand if you're using Yarn

Now create your initial Prisma setup using the `init` command of the Prisma CLI:

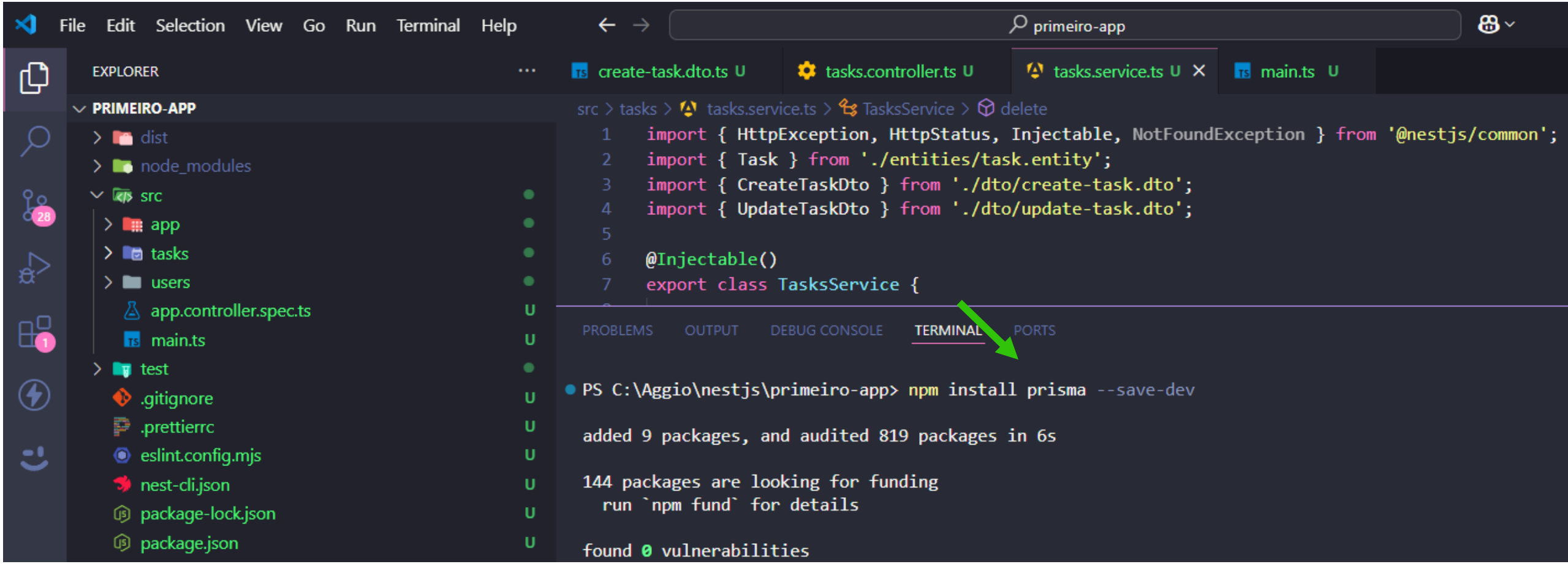
```
$ npx prisma init
```

INSTITUTO FEDERAL
PARANÁ

Prisma

Prof. Dr. Robyson Aggio



Install and generate Prisma Client

Prisma Client is a type-safe database client that's *generated* from your Prisma model definition. Because of this approach, Prisma Client can expose CRUD operations that are *tailored* specifically to your models.

To install Prisma Client in your project, run the following command in your terminal:

```
$ npm install @prisma/client
```

Note that during installation, Prisma automatically invokes the `prisma generate` command for you. In the future, you need to run this command after *every* change to your Prisma models to update your generated Prisma Client.

> **NOTE**
> The `prisma generate` command reads your Prisma schema and updates the generated Prisma Client library inside `node_modules/@prisma/client`.

Use Prisma Client in your NestJS services

You're now able to send database queries with Prisma Client. If you want to learn more about building queries with Prisma Client, check out the API documentation.

When setting up your NestJS application, you'll want to abstract away the Prisma Client API for database queries within a service. To get started, you can create a new `PrismaService` that takes care of instantiating `PrismaClient` and connecting to your database.

Inside the `src` directory, create a new file called `prisma.service.ts` and add the following code to it:

```
import { Injectable, OnModuleInit } from '@nestjs/common';
import { PrismaClient } from '@prisma/client';

@Injectable()
```

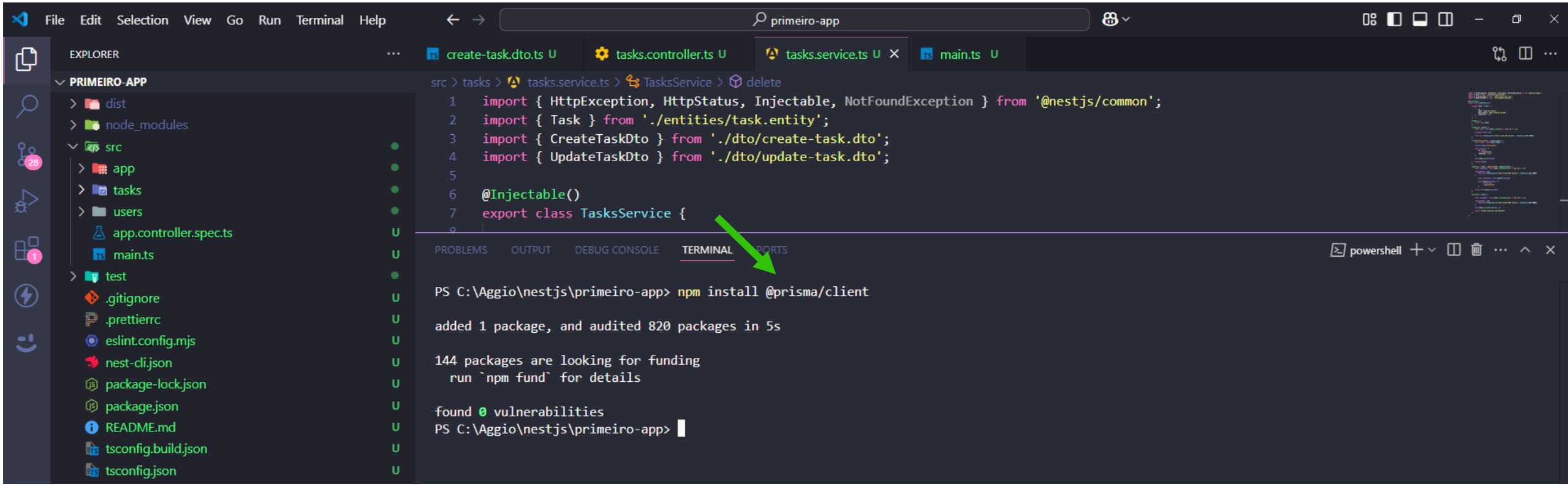

Design and Development tips in your inbox. Every weekday.

ADS VIA CARBON

INSTITUTO FEDERAL
PARANÁ

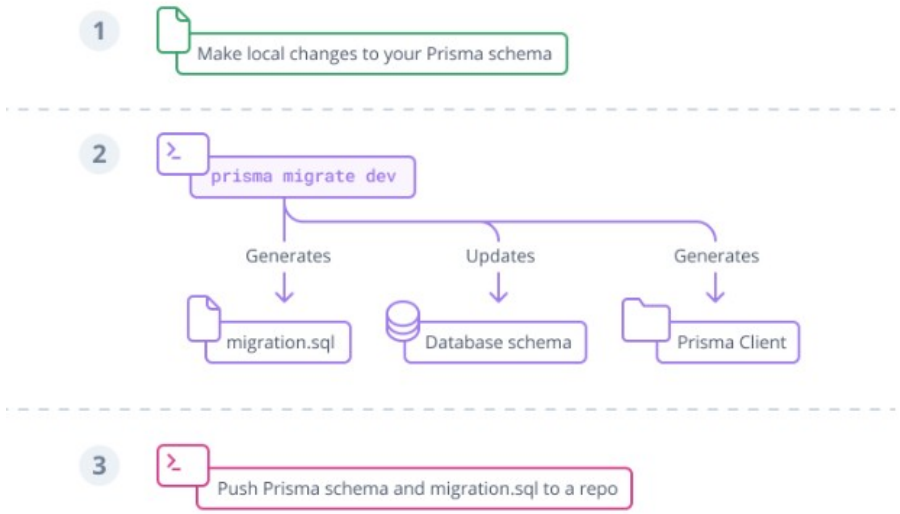
Prisma

Prof. Dr. Robyson Aggio



Prisma

Prof. Dr. Robyson Aggio



Prisma ORM makes developers productive

Prisma ORM's main goal is to make application developers more productive when working with databases. Considering the tradeoff between productivity and control again, this is how Prisma ORM fits in:

Prisma

Prof. Dr. Robyson Aggio

Prisma | NestJS - A progressive | Building REST APIs with Prisma

prisma.io/docs/orm/overview/prisma-in-your-stack/rest

Prisma /docs | Get Started | Products | Guides | Examples | Search CTRL K | Log In

Want real-time updates from your database without manual polling?

ORM

OVERVIEW
- Introduction
- Prisma ORM in your stack
  - REST
  - GraphQL
  - Fullstack
  - Is Prisma ORM an ORM?
- Databases
- Beyond Prisma ORM

PRISMA SCHEMA
- Overview
- Data model
- Introspection
- PostgreSQL extensions Preview

PRISMA CLIENT
- Setup & configuration
- Queries
- Write your own SQL

/ ORM / Overview / Prisma ORM in your stack

REST

When building REST APIs, Prisma Client can be used inside your *route controllers* to send databases queries.

ON THIS PAGE

Supported libraries

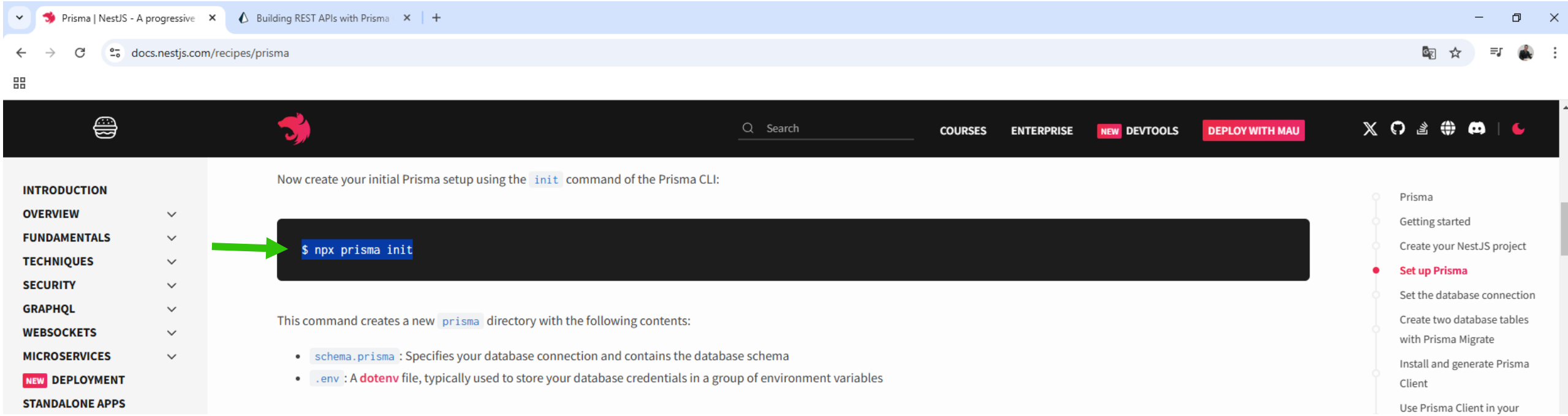
As Prisma Client is "only" responsible for sending queries to your database, it can be combined with any HTTP server library or web framework of your choice.

Here's a non-exhaustive list of libraries and frameworks you can use with Prisma ORM:

- Express
- koa
- hapi
- Fastify
- Sails
- AdonisJs
- NestJS

Prof. Dr. Robyson Aggio

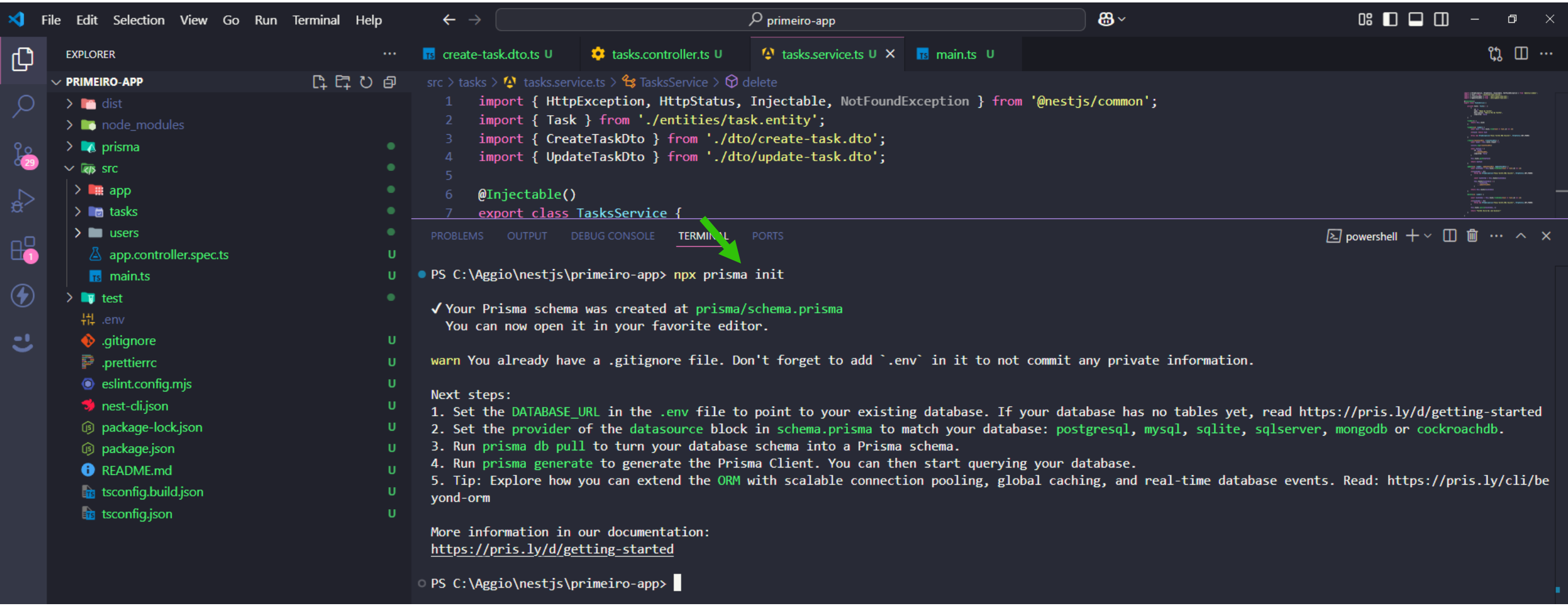
Prisma



INSTITUTO FEDERAL
PARANÁ

Prisma

Prof. Dr. Robyson Aggio



Prisma

Prof. Dr. Robyson Aggio

Prisma | NestJS - A progressive | Building REST APIs with Prisma

docs.nestjs.com/recipes/prisma

Set the database connection

Your database connection is configured in the `datasource` block in your `schema.prisma` file. By default it's set to `postgresql`, but since you're using a SQLite database in this guide you need to adjust the `provider` field of the `datasource` block to `sqlite`:

```
datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}
```

Now, open up `.env` and adjust the `DATABASE_URL` environment variable to look as follows:

```
DATABASE_URL="file:./dev.db"
```

Make sure you have a **ConfigModule** configured, otherwise the `DATABASE_URL` variable will not be picked up from `.env`.

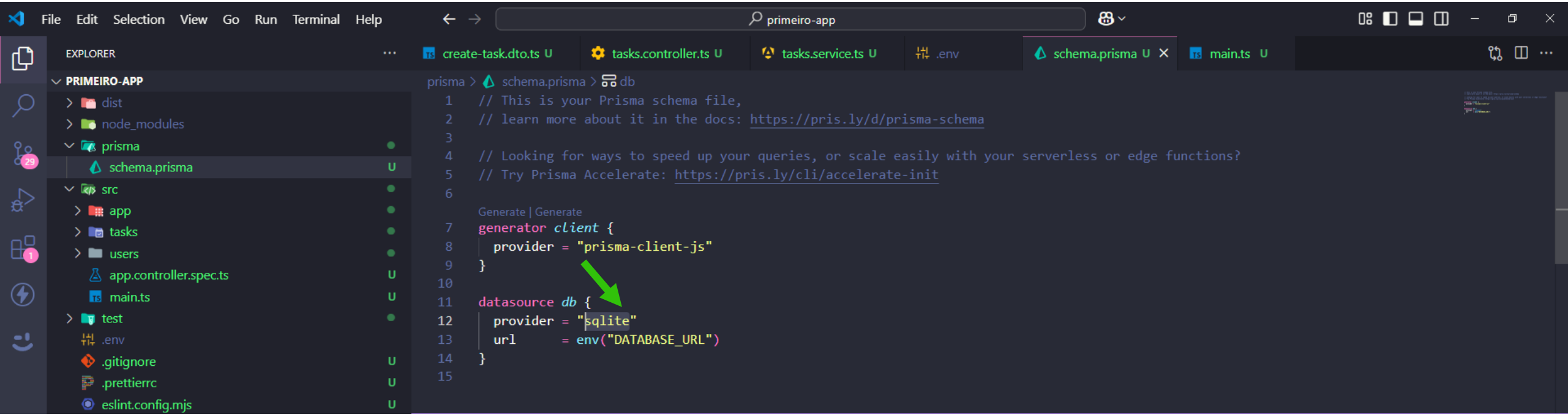
SQLite databases are simple files; no server is required to use a SQLite database. So instead of configuring a connection URL with a *host* and *port*, you can just point it to a local file which in this case is called `dev.db`. This file will be created in the next step.

▶ Expand if you're using PostgreSQL, MySQL, MsSQL or Azure SQL

Design and

Prof. Dr. Robyson Aggio

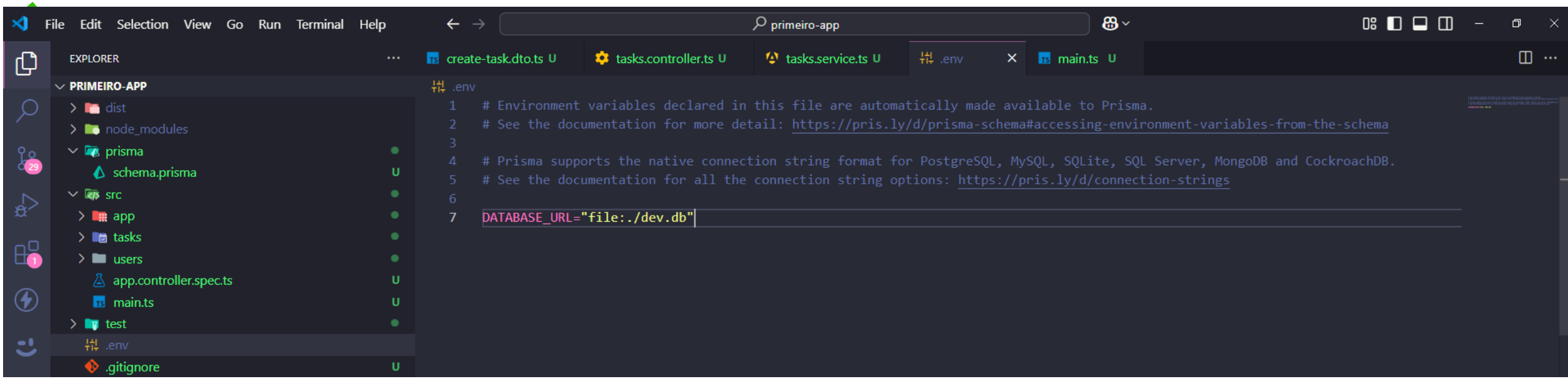
Prisma



INSTITUTO FEDERAL
PARANÁ

Prisma

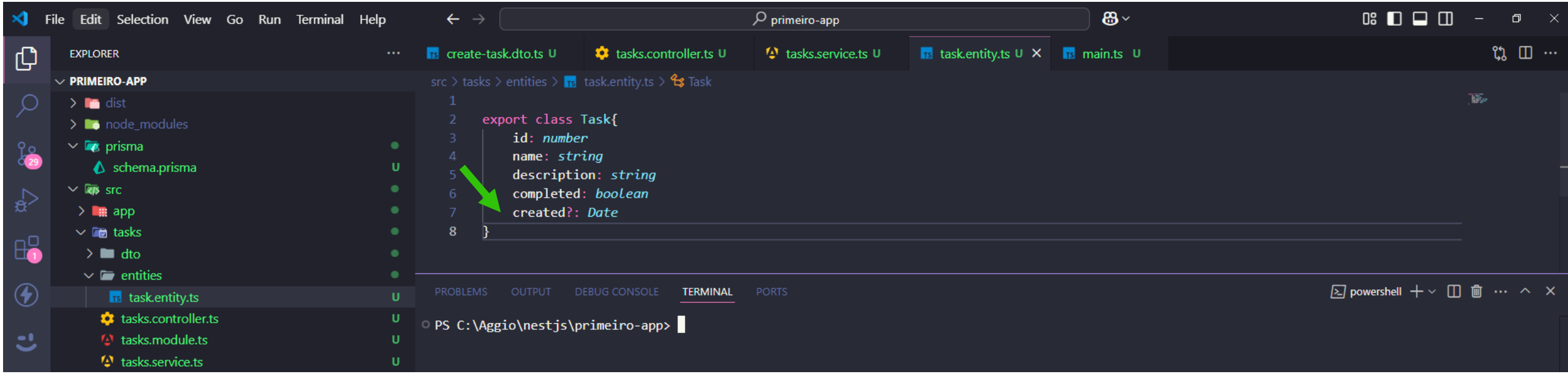
Prof. Dr. Robyson Aggio



INSTITUTO FEDERAL
PARANÁ

Prisma

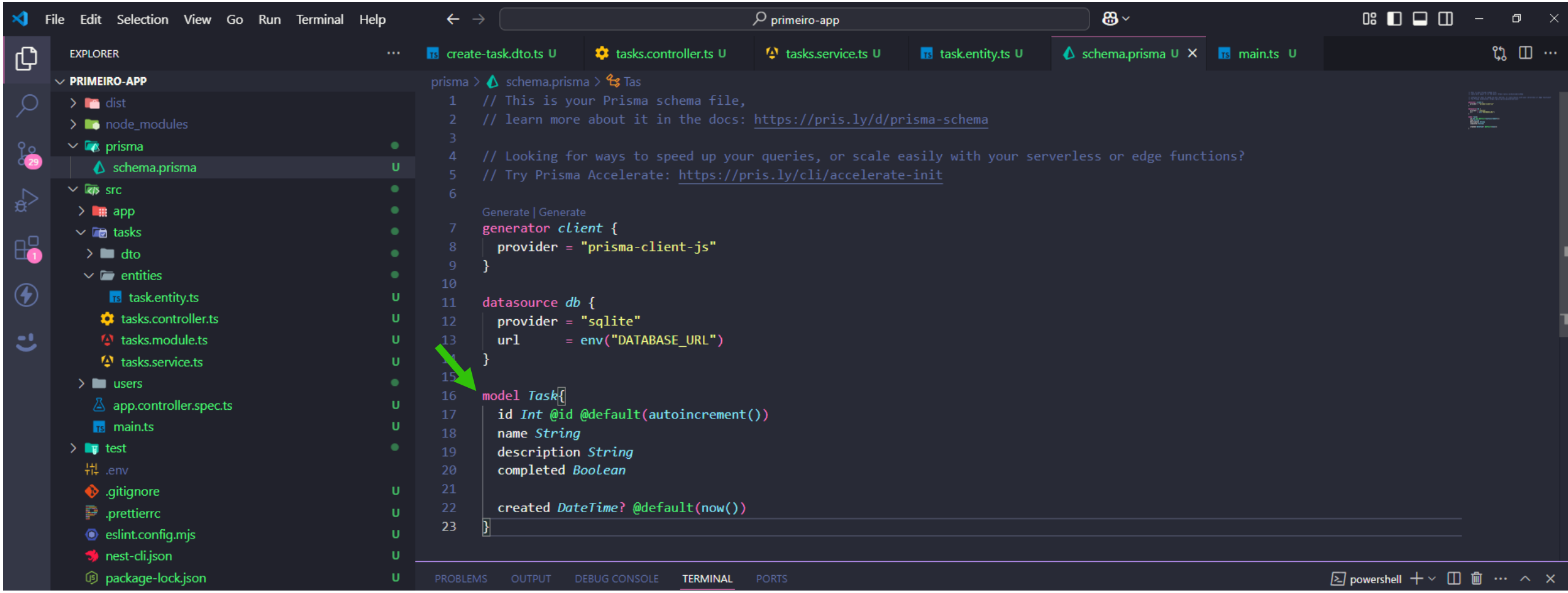
Prof. Dr. Robyson Aggio



INSTITUTO FEDERAL
PARANÁ

Prisma

Prof. Dr. Robyson Aggio



Create two database tables with Prisma Migrate

In this section, you'll create two new tables in your database using Prisma Migrate. Prisma Migrate generates SQL migration files for your declarative data model definition in the Prisma schema. These migration files are fully customizable so that you can configure any additional features of the underlying database or include additional commands, e.g. for seeding.

Add the following two models to your `schema.prisma` file:

```
model User {
  id    Int     @default(autoincrement()) @id
  email String  @unique
  name  String?
  posts Post[]
}

model Post {
  id        Int      @default(autoincrement()) @id
  title     String
  content   String?
  published Boolean? @default(false)
  author    User?    @relation(fields: [authorId], references: [id])
  authorId  Int?
}
```

With your Prisma models in place, you can generate your SQL migration files and run them against the database. Run the following commands in your terminal:

```
$ npx prisma migrate dev --name init
```


Design and Development tips in your inbox. Every

Prisma

Selection View Go Run Terminal Help

create-task.dto.ts U | tasks.controller.ts U | tasks.service.ts U | task.entity.ts U | schema.prisma U | main.ts U

prisma > schema.prisma > Tas

```
datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model Task{
  id Int @id @default(autoincrement())
  name String
  description String
  completed Boolean

  created DateTime? @default(now())
}
```

PRIMEIRO-APP

- dist
- node_modules
- prisma
  - migrations
  - dev.db U
  - dev.db-journal U
  - schema.prisma U
- src
  - app
  - tasks
    - dto
    - entities
      - task.entity.ts U
    - tasks.controller.ts U
    - tasks.module.ts U
    - tasks.service.ts U
  - users
  - app.controller.spec.ts U
  - main.ts U
- test
- .env
- .gitignore U
- .prettierrc U
- eslint.config.mjs U
- nest-cli.json U
- package-lock.json U
- package.json U
- README.md U
- tsconfig.build.json U
- tsconfig.json U

OUTLINE

TIMELINE

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

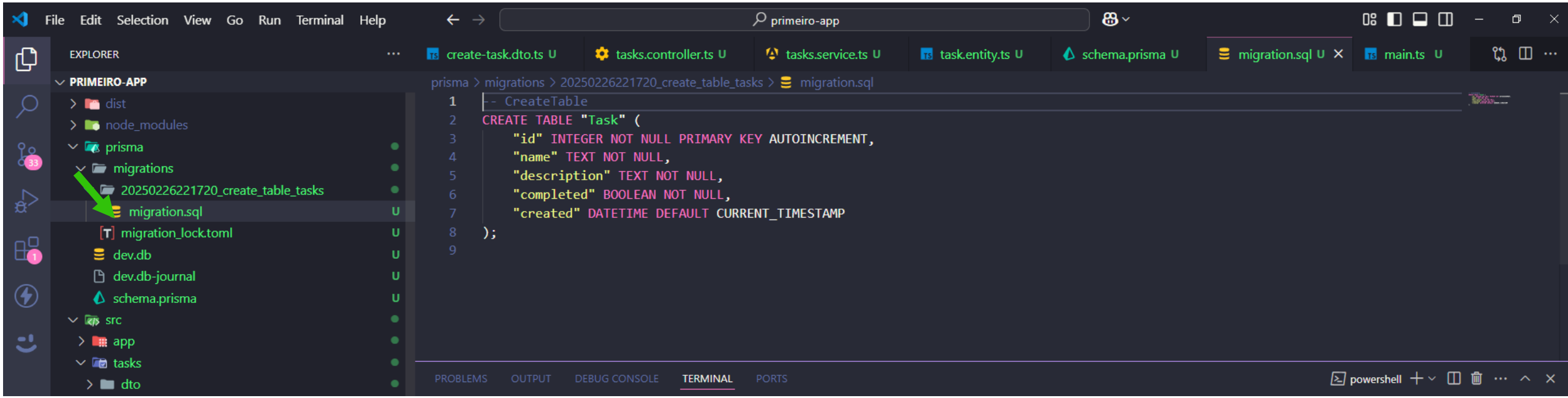
powershell

```
PS C:\Aggio\nestjs\primeiro-app> npx prisma migrate dev
Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Datasource "db": SQLite database "dev.db" at "file:./dev.db"

SQLite database dev.db created at file:./dev.db

√ Enter a name for the new migration: ... create-table-tasks
Applying migration `20250226221720_create_table_tasks`

The following migration(s) have been created and applied from new schema changes:

migrations/
  └─ 20250226221720_create_table_tasks/
    └─ migration.sql

Your database is now in sync with your schema.

✓ Generated Prisma Client (v6.4.1) to .\node_modules\@prisma\client in 46ms

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

INSTITUTO FEDERAL
PARANÁ

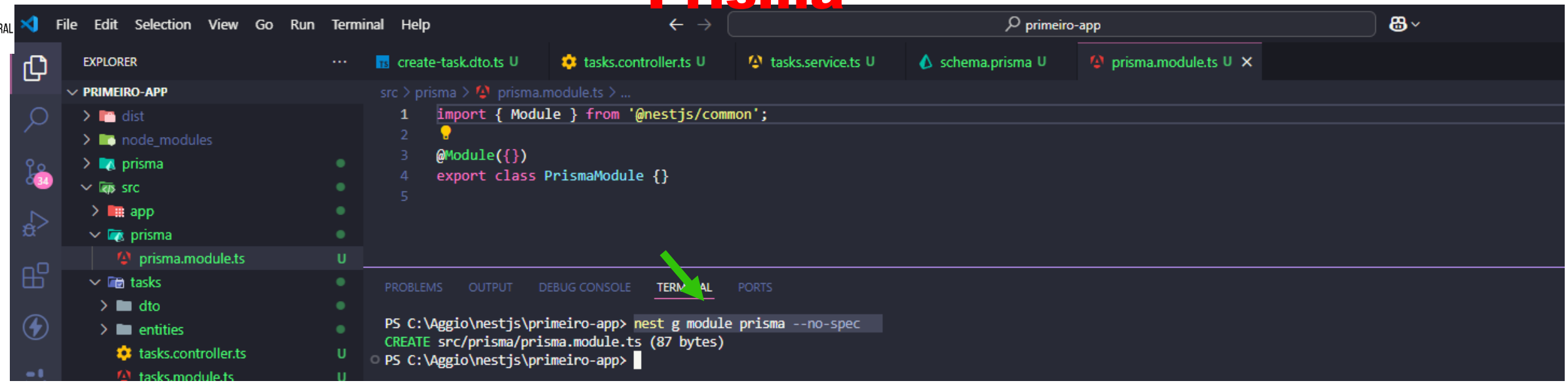
Prisma

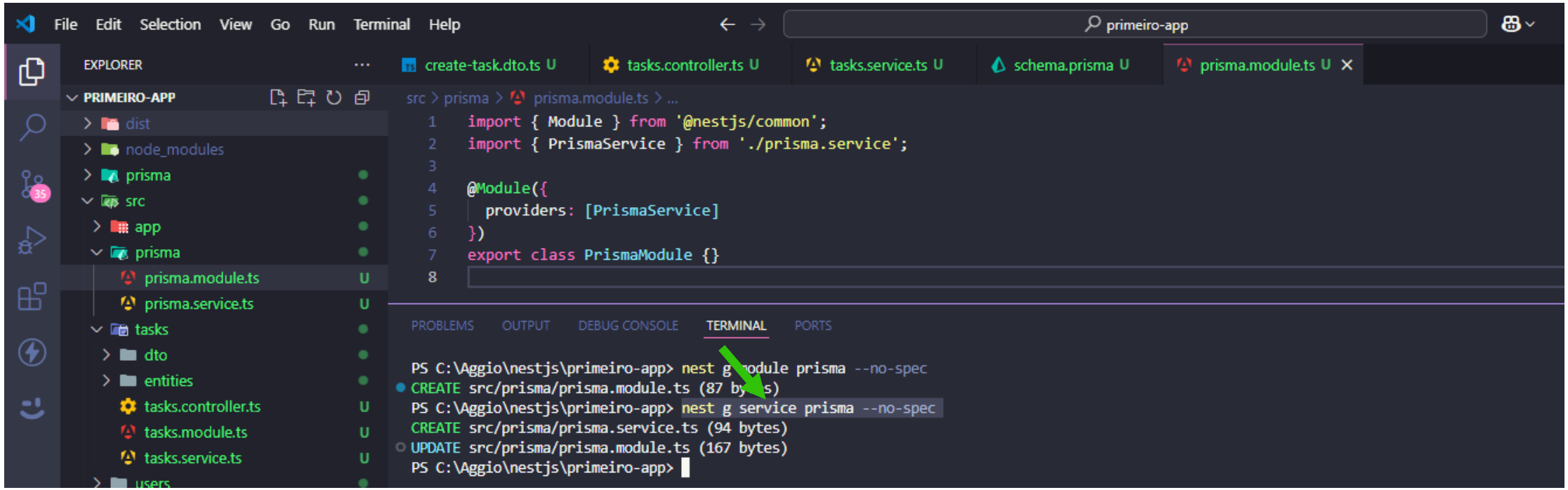
Prof. Dr. Robyson Aggio



Prisma

Prof. Dr. Robyson Aggio

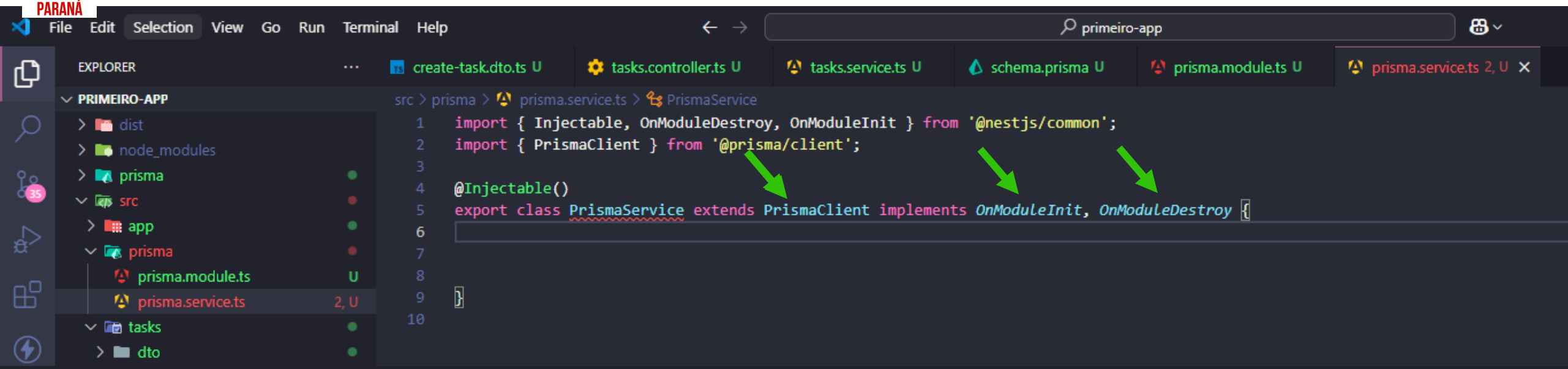


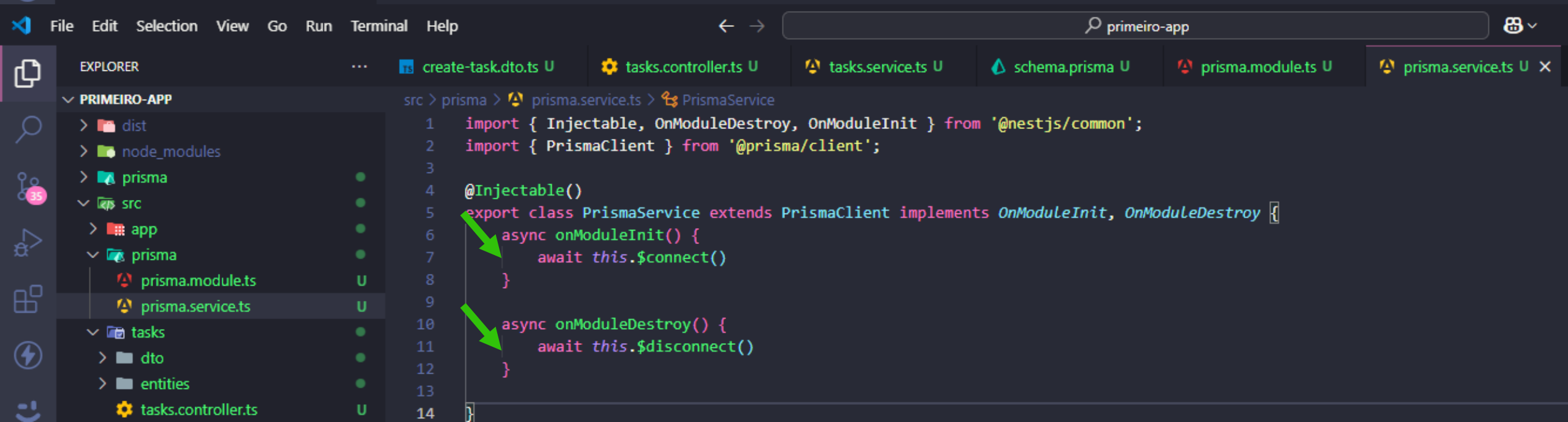


INSTITUTO FEDERAL
PARANÁ

Prisma

Prof. Dr. Robyson Aggio

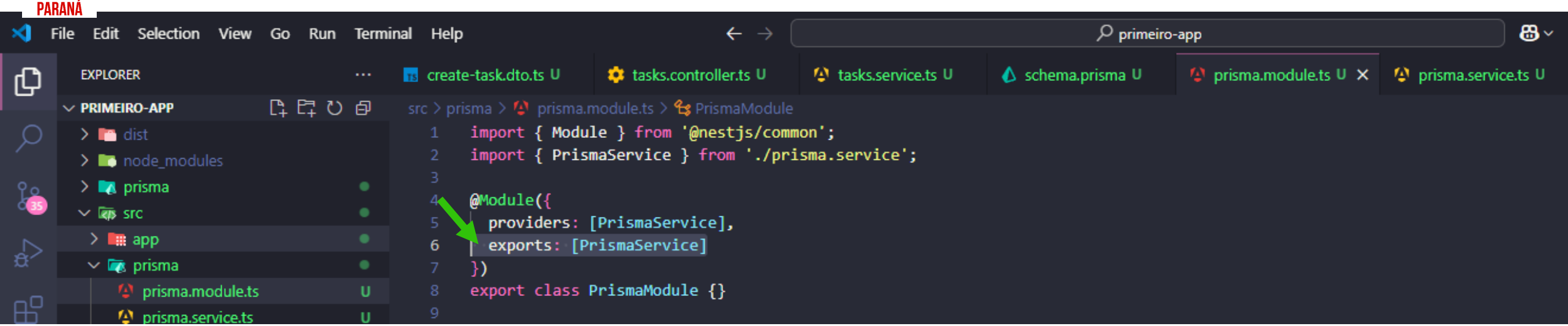


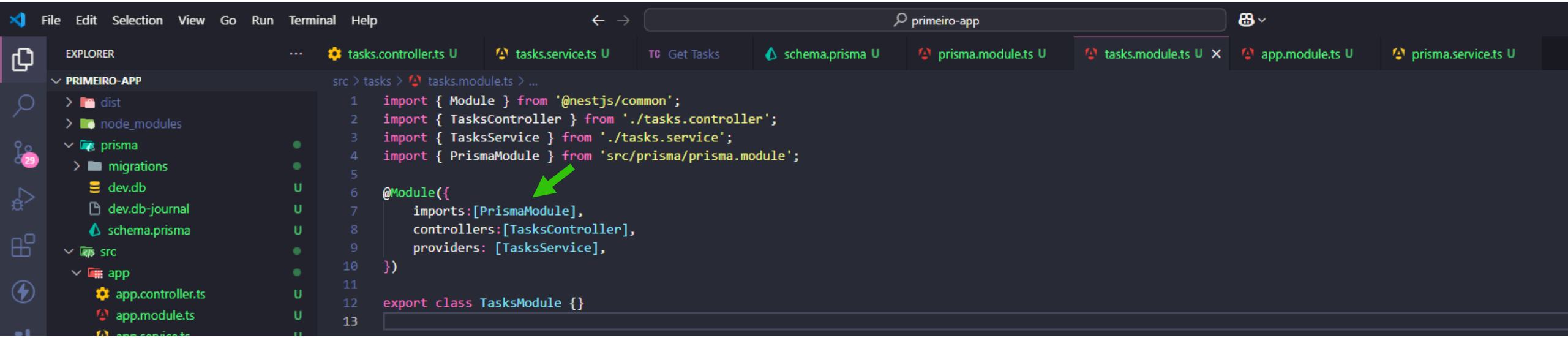


INSTITUTO FEDERAL
PARANÁ

Prisma

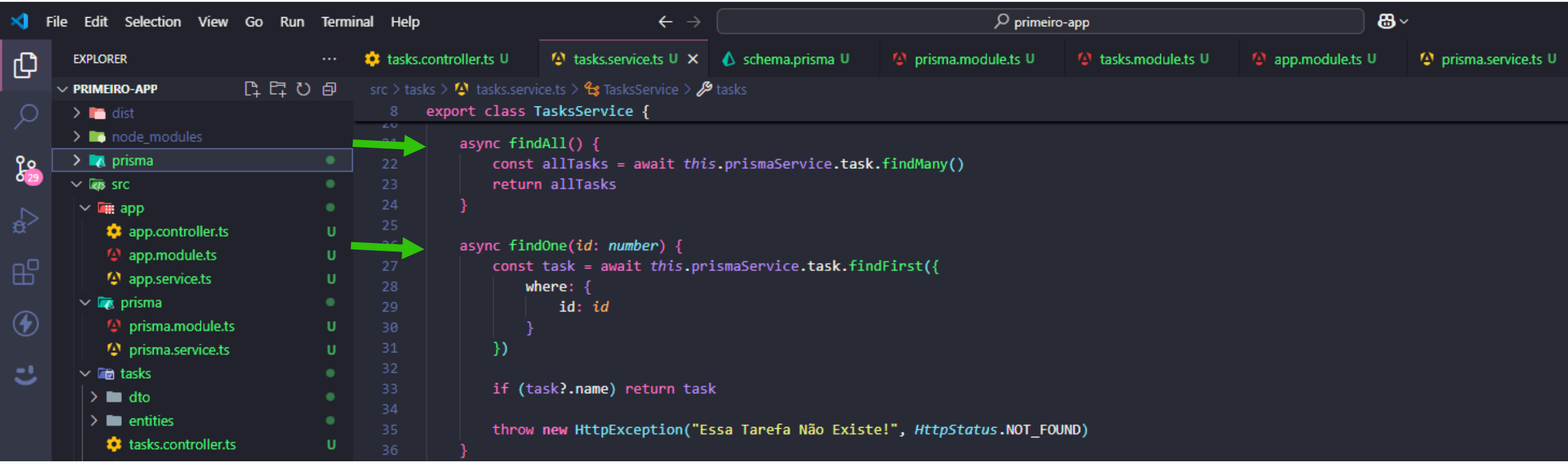
Prof. Dr. Robyson Aggio





Prof. Dr. Robyson Aggio

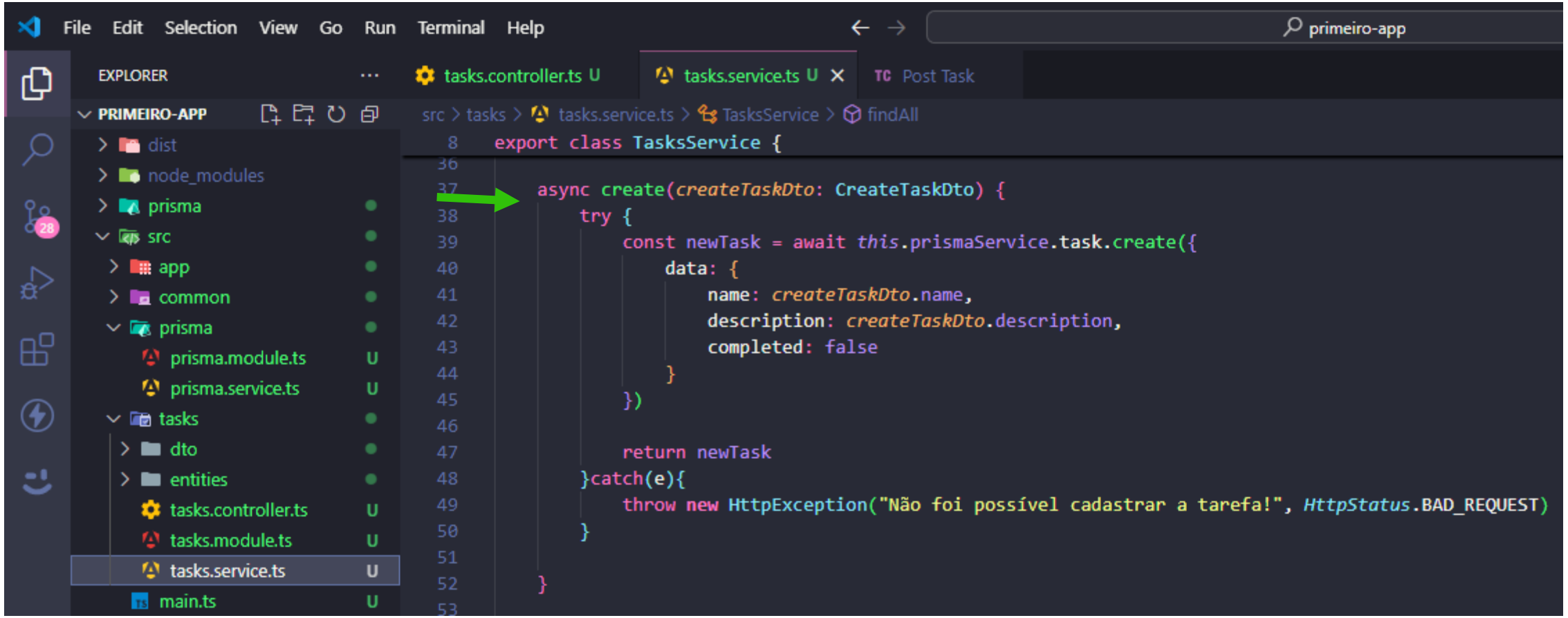
Get



INSTITUTO FEDERAL
PARANÁ

Post

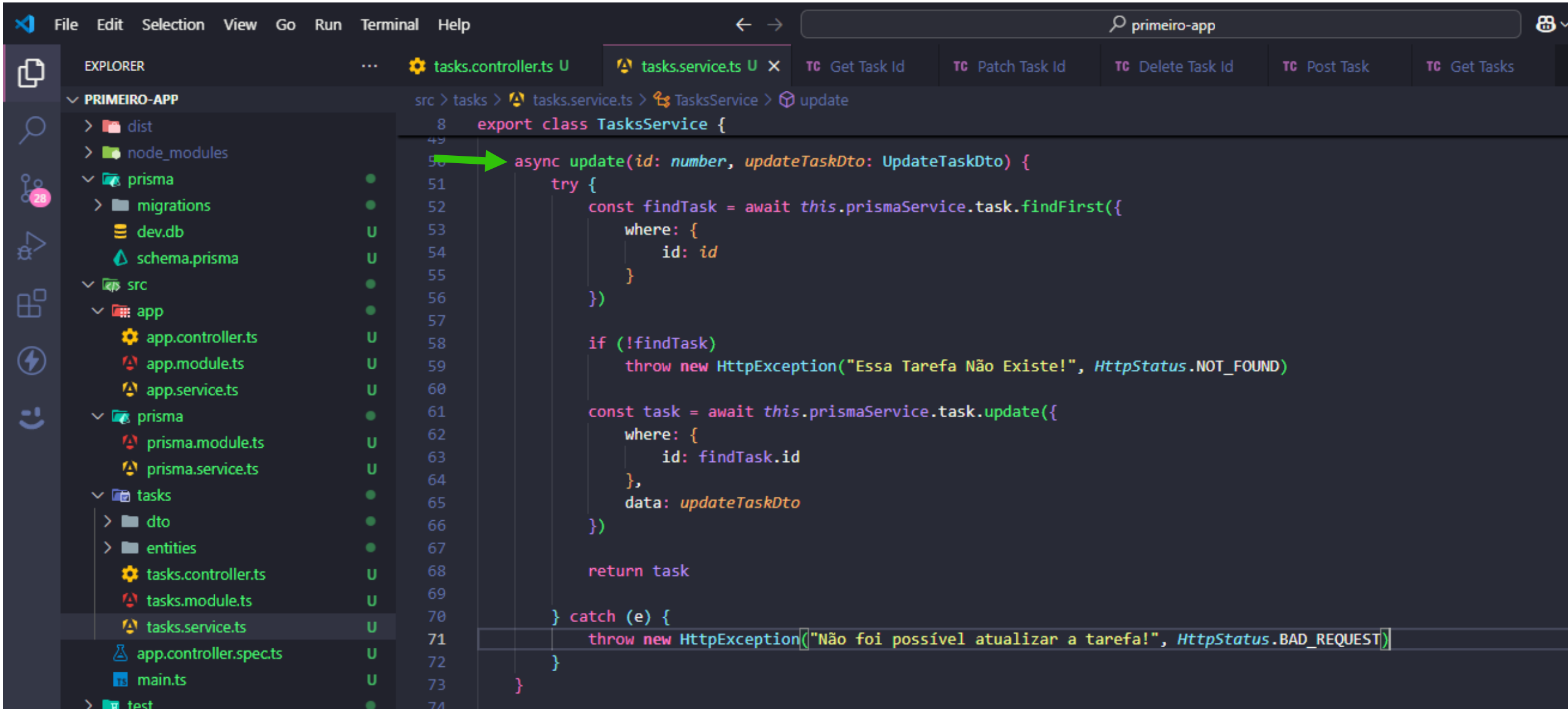
Prof. Dr. Robyson Aggio



INSTITUTO FEDERAL
PARANÁ

Patch

Prof. Dr. Robyson Aggio



INSTITUTO FEDERAL
PARANÁ

Delete

Prof. Dr. Robyson Aggio

File Edit Selection View Go Run Terminal Help

primeiro-app

EXPLORER

tasks.controller.ts U | tasks.service.ts U | Get Task Id | Patch Task Id | Delete Task Id | Post Task | Get Tasks

PRIMEIRO-APP

- dist
- node_modules
- prisma
  - migrations
  - dev.db U
  - schema.prisma U
- src
  - app
    - app.controller.ts U
    - app.module.ts U
    - app.service.ts U
  - prisma
    - prisma.module.ts U
    - prisma.service.ts U
  - tasks
    - dto
    - entities
    - tasks.controller.ts U
    - tasks.module.ts U
    - tasks.service.ts U
  - app.controller.spec.ts U
  - main.ts U
- test

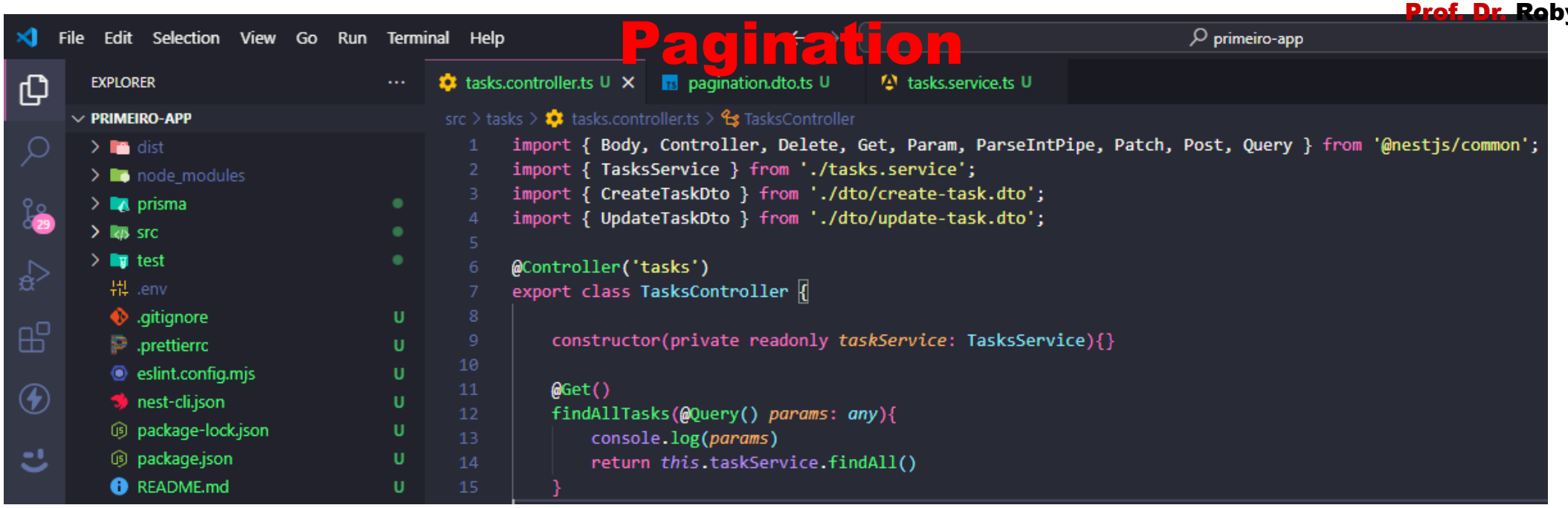
src > tasks > tasks.service.ts > TasksService > delete

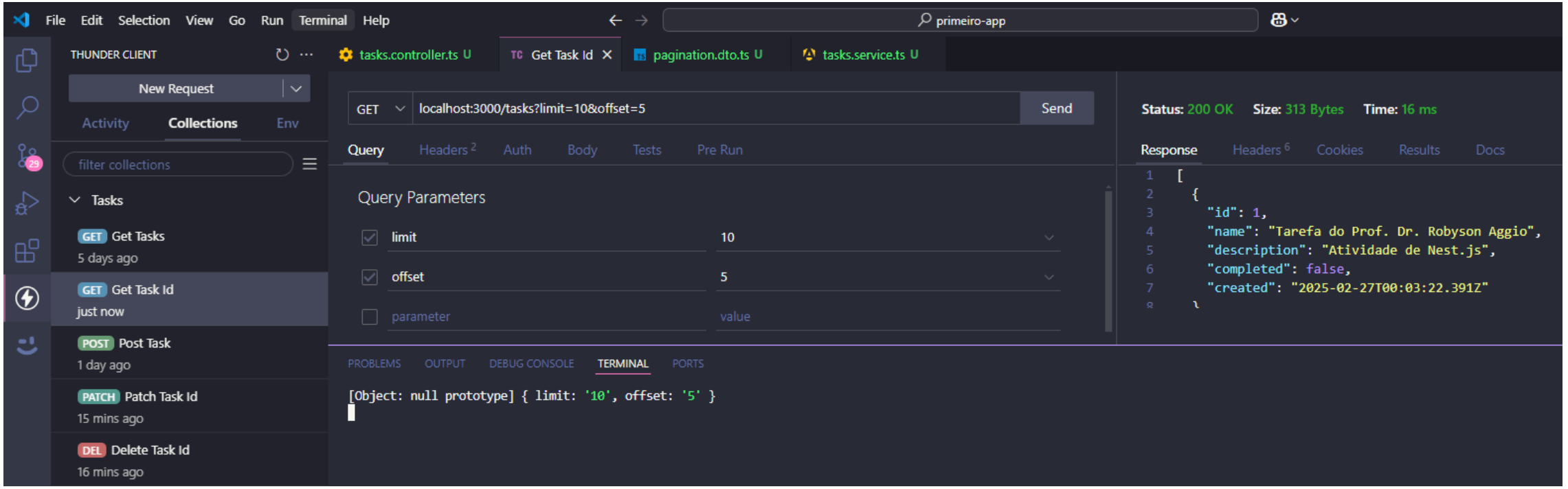
```
8   export class TasksService {
75      async delete(id: number) {
76          try {
77              const findTask = await this.prismaService.task.findFirst({
78                  where: {
79                      id: id
80                  }
81              })
82
83              if (!findTask)
84                  throw new HttpException("Essa Tarefa Não Existe!", HttpStatus.NOT_FOUND)
85
86              await this.prismaService.task.delete({
87                  where: {
88                      id: findTask.id
89                  }
90              })
91
92              return "Tarefa Excluída com Sucesso!"
93          } catch (e) {
94              throw new HttpException("Não foi possível deletar a tarefa!", HttpStatus.BAD_REQUEST)
95          }
96
97      }
98
99  }
```

INSTITUTO FEDERAL
PARANÁ

Prof. Dr. Robyson Aggio

Pagination





Pagination

Prof. Dr. Robyson Aggio

INSTITUTO FEDERAL PARANÁ

```
import { Type } from "class-transformer"
import { IsInt } from "class-validator"

export class PaginationDto{
    @IsInt()
    @Type(() => Number)
    limit : number
    @IsInt()
    @Type(() => Number)
    offset: number
}
```

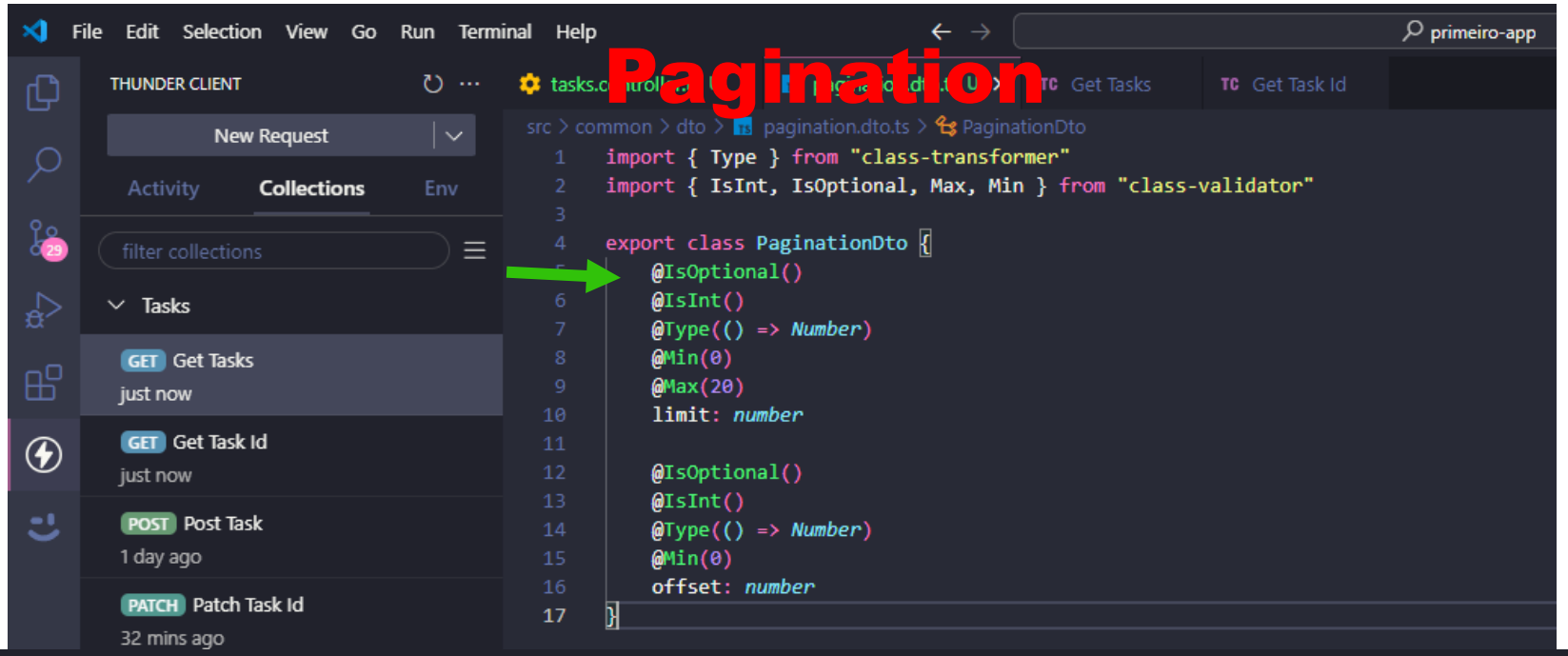
```
[
  {
    "id": 1,
    "name": "Tarefa do Prof. Dr. Robyson Aggio",
    "description": "Atividade de Nest.js",
    "completed": false,
    "created": "2025-02-27T00:03:22.391Z"
  }
```

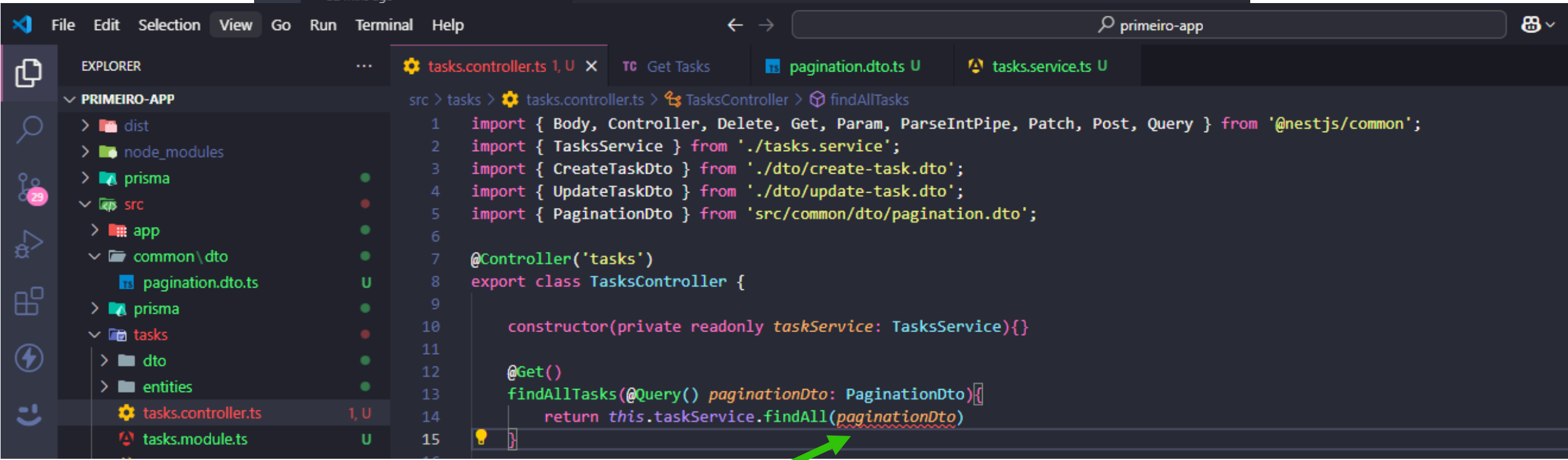
```
PaginationDto { limit: 10, offset: 5 }
```

INSTITUTO FEDERAL
PARANÁ

Pagination

Prof. Dr. Robyson Aggio

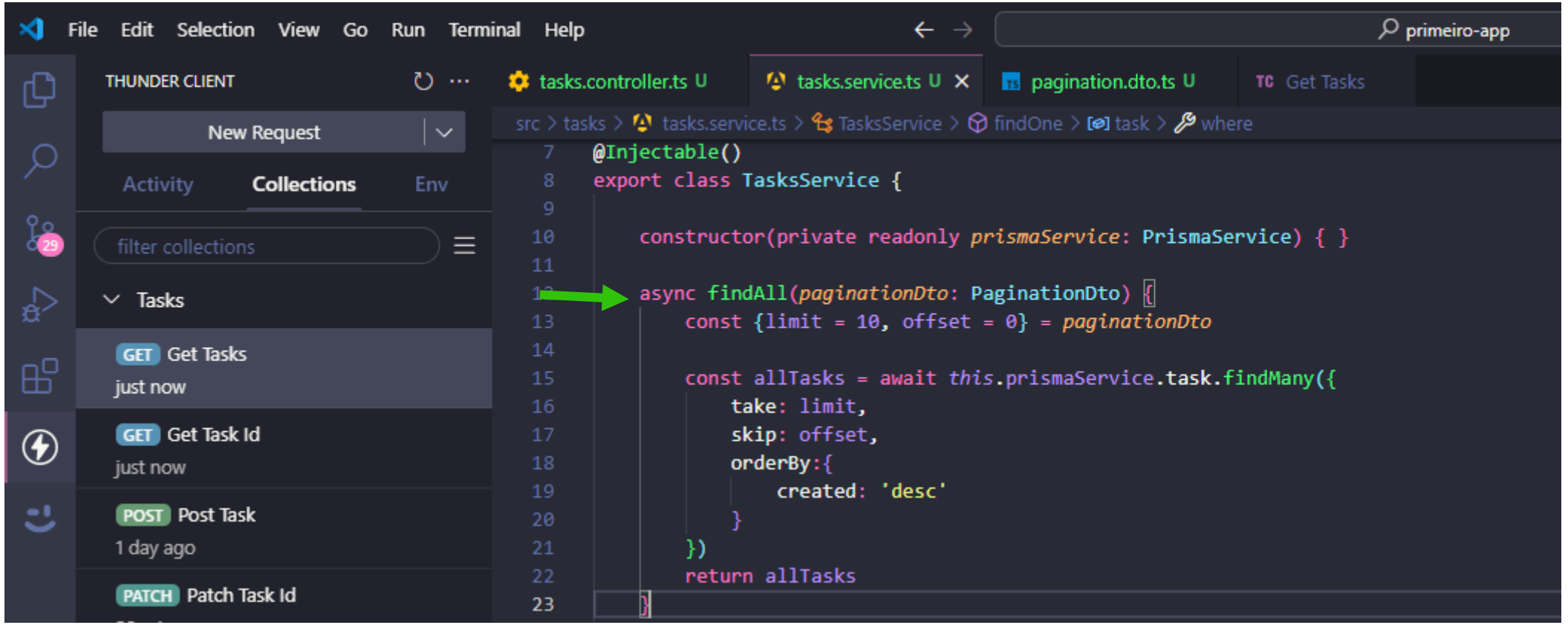




INSTITUTO FEDERAL
PARANÁ

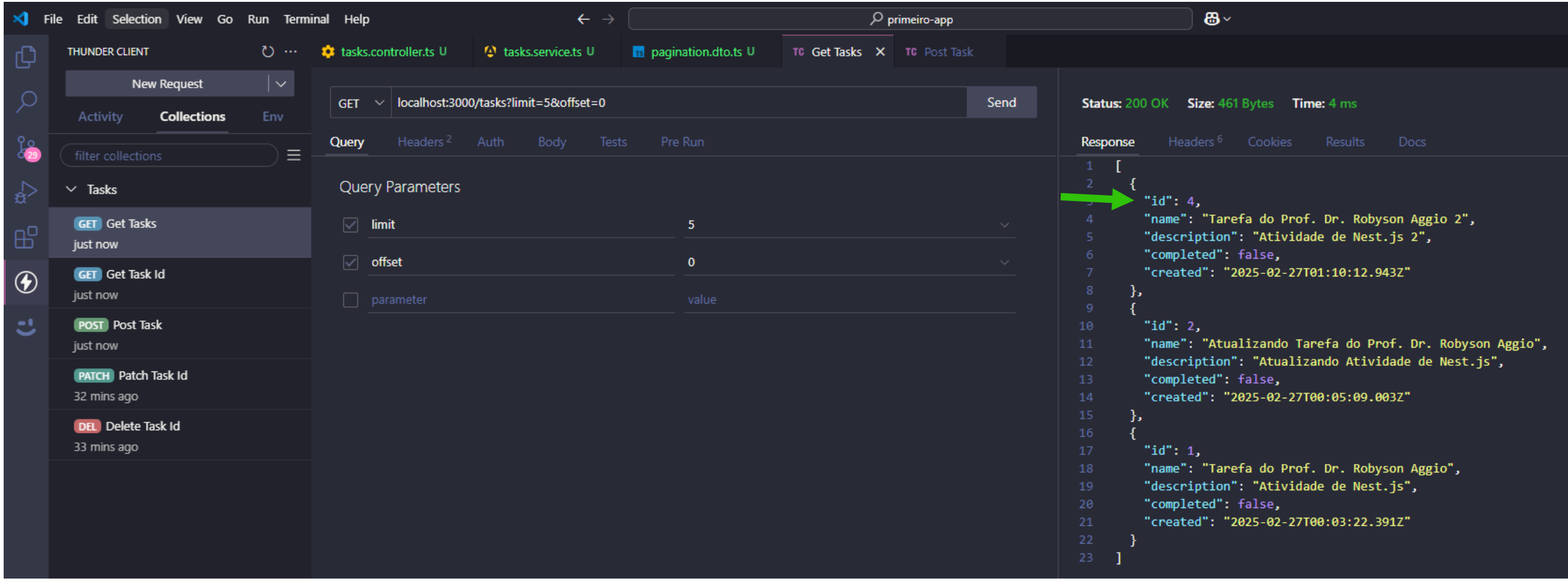
Pagination

Prof. Dr. Robyson Aggio



Testar no Thunder ou Postman

Prof. Dr. Robyson Aggio



Prof. Dr. Robyson Aggio

Atividade

1) Implemente o prisma

2) No arquivo schema.prisma, crie os models: Guest, Users, Teachers

3) Para cada model do item 2), implemente o prisma para os verbos (Get, Post, Patch e Delete)

4) Crie a camada: common / dto / pagination.dto.ts

4.1) crie o pagination.dto.ts

4.2) Utilize o paginationDto no verbo get, no método findAll()

5) Testar no Thunder Client ou Postman