INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

**TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO**
**Disciplina de Desenvolvimento Web II**
Aula 05: Adonis JS - Routes / Controller / Validator

---

*Gil Eduardo de Andrade*

Material TypeScrip - Básico
Material Docker (Adonis + MySQL + PhpMyAdmin)
Repositório - Aplicação

---

# Conceitos Preliminares

(https://adonisjs.com/)

## Introdução - AdonisJS

AdonisJS é um framework web robusto e moderno para Node.js, construído com TypeScript como linguagem principal. Ele oferece uma estrutura completa para o desenvolvimento de aplicações web full-stack e servidores de API JSON, com foco na ergonomia do desenvolvedor, estabilidade e performance.

### Características Principais

- TypeScript-first: totalmente escrito em TypeScript, proporcionando tipagem estática e um desenvolvimento mais seguro e produtivo;
- Estrutura MVC: segue o padrão Model-View-Controller, promovendo uma organização clara e escalável do código;
- Ecossistema: vem com um ecossistema de pacotes oficiais para tarefas comuns como autenticação, ORM, validação, envio de e-mails, etc;
- Frontend: apesar de focado no backend, também permite a integração com qualquer framework frontend (React, Vue, Angular, etc.) ou o uso de templates server-side com Edge.js;
- Segurança: oferece proteção contra ameaças comuns como CSRF, XSS e outros, através do pacote Shield;
- Ferramentas Modernas: utiliza tecnologias como Vite para bundling de assets, SWC para compilação de TypeScript e HMR (Hot Module Replacement) para um desenvolvimento mais ágil.

### Filosofia do Framework

A filosofia do AdonisJS é permitir que os desenvolvedores se concentrem na lógica de negócio e na entrega de valor, em vez de gastar tempo com decisões triviais de

configuração e arquitetura. O framework busca fornecer uma base sólida e bem documentada, com convenções inteligentes que guiam o desenvolvimento sem restringir a liberdade do desenvolvedor.

**Comparação com outros frameworks**

- Express.js: enquanto o Express é minimalista e não opinativo, o AdonisJS é um framework completo com uma estrutura bem definida, o que pode acelerar o desenvolvimento de aplicações complexas;
- NestJS: Ambos são frameworks TypeScript para Node.js, mas o NestJS é mais focado em arquitetura de microsserviços e utiliza uma abordagem mais orientada a anotações, enquanto o AdonisJS segue um padrão MVC mais tradicional, similar a frameworks como Laravel e Ruby on Rails.

**Quando usar AdonisJS?**

- Desenvolvimento de APIs RESTful e GraphQL robustas e escaláveis;
- Aplicações web full-stack com renderização no servidor;
- Projetos que exigem um alto nível de organização e manutenibilidade;
- Equipes que valorizam a produtividade e a ergonomia do desenvolvedor;
- Aplicações que necessitam de um ecossistema de pacotes coeso e bem integrado.

**Comandos Ace**

Ace é o framework de linha de comando do AdonisJS. Ele fornece uma série de comandos úteis para o desenvolvimento da aplicação.

Comandos Built-in:

*Inicia o servidor de desenvolvimento com Hot Module Replacement*

```shell
node ace serve --hmr
```

*Cria um novo Controller*

```shell
node ace make:controller
```

*Cria um novo Model*

```shell
node ace make:model
```

*Cria uma nova Migration*

```shell
node ace make:migration
```

*Lista todas as rotas da aplicação.*

```shell
node ace list:routes
```

**Desenvolvimento e Deploy**

*Servidor de Desenvolvimento*

```shell
node ace serve --hmr
```

*Build para Produção*

```shell
node ace build --production
```

**Variáveis de Ambiente**

As variáveis de ambiente da aplicação podem ser gerenciadas através do arquivo ***".env"***.

*Estrutura básica do arquivo ".env"*

```typescript
TZ=UTC
PORT=3333
HOST=localhost
LOG_LEVEL=info
```

```
APP_KEY=DZjNaoIIb3RW3ll-MbCeT1crQemSJEau
NODE_ENV=development
DB_HOST=127.0.0.1
DB_PORT=3306
DB_USER=root
DB_PASSWORD=root
DB_DATABASE=app
```

**Deploy em Produção**
O AdonisJS pode ser implantado em qualquer serviço de hospedagem que suporte Node.js, como Heroku, DigitalOcean, AWS, etc.

**Instalação e Configuração**
Ace - Script / Linha de Comandos

*Criando Projeto - Básico*

Shell

```
npm init adonisjs@latest hello-world
```

*Criando Projeto - Pacotes Adicionais*

Shell

```
# Com suporte ao MySQL
npm init adonisjs@latest hello-world -- --db=mysql

# Com suporte ao PostgreSQL e no Formato API
npm init adonisjs@latest hello-world -- --db=postgres
--kit=api

# No fomato API e com Kit Auth e Token de Acesso
npm init adonisjs@latest hello-world -- --kit=api
--auth-guard=access_tokens

# Kit Web
npm init adonisjs@latest -- -K=web
```

**Mais Informações:** https://docs.adonisjs.com/guides/getting-started/installation

**Estrutura de Diretórios**

- /app: contém a lógica de negócio da aplicação, como controllers, models, middleware, etc.
- /config: arquivos de configuração para os diferentes módulos do framework.
- /database: migrations e seeders para o seu banco de dados.
- /public: assets estáticos como imagens, CSS e JavaScript.
- /resources: Arquivos de frontend como templates Edge.js e código-fonte de
- assets.
- /start: arquivos de inicialização, como a definição de rotas e eventos.
- /tests: testes automatizados da sua aplicação.

**Arquivos de Configuração**

- adonisrc.ts: configurações do workspace e do runtime da aplicação.
- tsconfig.json: configurações do TypeScript.
- package.json: dependências do projeto e scripts npm.

**Sistema de Rotas**

O sistema de rotas do AdonisJS é responsável por mapear as URLs da sua aplicação para os respectivos controllers e actions. As rotas são definidas no arquivo *"start/routes.ts"*

```typescript
import router from '@adonisjs/core/services/router'

router.get('/', async () => {
      return 'Hello, world!'
})
```

**Métodos HTTP Suportados**

- router.get()
- router.post()
- router.put()
- router.patch()
- router.delete()

### Parâmetros de Rota

```typescript
router.get('/posts/:id', ({ params }) => {
      return `Exibindo o post com ID ${params.id}`
})
```

### Route Middleware

```typescript
router.get('/profile', [ ... ]).use(middleware.auth())
```

### Controllers

Os *controllers* são responsáveis por agrupar a lógica de manipulação de requisições relacionadas a um determinado recurso da sua aplicação.

```shell
node ace make:controller Post
```

*Estrutura Básica do Controller*

```typescript
import type { HttpContext } from '@adonisjs/core/http'

export default class PostsController {

      public async index({}: HttpContext) {
            // Lógica para listar posts
      }
      public async store({ request }: HttpContext) {
            // Lógica para criar um novo post
      }
}
```

### Resource Controllers

É possível criar um controller com todos os métodos RESTful padrão, utilizando a flag *--resource*.

```shell
node ace make:controller Post --resource
```

**Middleware**

Middleware são funções que são executadas antes ou depois de uma rota ou de um grupo de rotas. Eles são úteis para tarefas como autenticação, logging, etc.

*Comando - Criação de um Middleware*

```shell
node ace make:middleware Auth
```

*Estrutura Básica de um Middleware*

```typescript
import type { HttpContext } from '@adonisjs/core/http'
import type { NextFn } from '@adonisjs/core/types/http'

export default class AuthMiddleware {
    async handle({ response }: HttpContext, next: NextFn) {
        // Lógica do middleware
        await next()
    }
}
```

**Registro de Middleware**

O middleware pode ser registrado globalmente no arquivo start/kernel.ts ou aplicado a rotas específicas.

**Request e Response**

O objeto *request* contém todas as informações sobre a requisição HTTP, como o corpo da requisição, os parâmetros da rota, os headers, etc.

O objeto *response* é utilizado para enviar a resposta ao cliente. Você pode enviar respostas em diferentes formatos, como JSON, HTML, etc.

**Validação de Dados**
O AdonisJS utiliza a biblioteca VineJS para a validação de dados.

```shell
node ace make:validator Post
```

*Utilizando no Controller*

```typescript
import { HttpContext } from '@adonisjs/core/http'
import { createPostValidator } from '#validators/post_validator'

export default class PostsController {
      async store({ request }: HttpContext) {
            const payload =
                  await request.validateUsing(createPostValidator)
            // ...
      }
}
```

**Models**
Os modelos são classes que representam as tabelas do banco de dados.

```shell
node ace make:model Post
```

**Operações de CRUD**
O Lucid facilita a realização de operações de CRUD (Create, Read, Update, Delete):

```typescript
// Criar
const post = await Post.create({ title: 'Novo Post', content: '...' })

// Ler
const posts = await Post.all()
```

```
// Atualizar
const post = await Post.findOrFail(1)
post.title = 'Título Atualizado'
await post.save()

// Deletar
const post = await Post.findOrFail(1)
await post.delete()
```

**Migrations**

As migrações são utilizadas para gerenciar a evolução do schema do banco de dados de forma versionada.

Shell

```
node ace make:migration create_posts_table
```

TypeScript

```
import { BaseSchema } from "@adonisjs/lucid/schema";

export default class extends BaseSchema {
      protected tableName = "posts";
      public async up() {
            this.schema.createTable(this.tableName, (table) => {
                  table.increments("id");
                  table.string("title");
                  table.text("content");
                  table.timestamp("created_at");
                  table.timestamp("updated_at");
            });
      }

      public async down() {
            this.schema.dropTable(this.tableName);
```

```
        }
}
```

**Comandos de Migração**

- *migration:run* - para executar todas as migrações pendentes;
- *migration:rollback* - para reverter a última execução de migrações;
- *migration:refresh* - para reverter e executar todas novamente;
- *migration:reset* - para reverter todas as migrações;
- *migration:status* - para verificar o estado das migrações;

**Seeders**

Os seeders são utilizados para popular o seu banco de dados com dados iniciais ou de teste.

*Criar um Seeder*

Shell
```
node ace make:seeder Post
```

*Executar Seeders*

Shell
```
node ace db:seed
```

---

# CRIANDO UMA APLICAÇÃO SIMPLES

Passo a Passo

*Criando Projeto Aula*

Shell
```
npm init adonisjs@latest aula -- --db=mysql --kit=api
```

---

## 1. Criando Services *"/app/services/*.ts"*

**Comando**

*(no terminal de comando)*

Shell

```
node ace make:service courseService
```

*Arquivo: "/app/services/course_service.ts"*

TypeScript

```typescript
import logger from '@adonisjs/core/services/logger'

interface Course {
  id?: number
  name: string
  duration: number
}

const courses: Course[] = []
let currentId: number = 1

export class CourseService {
  getAllCourses() {
    logger.info('index')
    return courses
  }

  createCourse(courseData: Course) {
    logger.info('create')
    if (!courseData.name || !courseData.duration) {
      return null
    }
    const newCourse = { id: currentId++, ...courseData }
    courses.push(newCourse)
    return newCourse
  }
```

```
  getCourseById(id: number) {
    logger.info('show')
    return courses.find((course) => course.id === id)
    /*for(let i=0; i<courses.length; i++) {
      if(id === courses[i].id) {
        return courses[i]
      }
    }
    return undefined*/
  }
  updateCourse(id: number, courseData: Course) {
    logger.info('update')
    const index = courses.findIndex((course) => course.id === id)
    if (index !== -1) {
      courses[index] = { ...courses[index], ...courseData, id: id }
      return courses[index]
    }
    return null
  }

  deleteCourse(id: number) {
    logger.info('delete')
    const index = courses.findIndex((course) => course.id === id)
    if (index !== -1) {
      return courses.splice(index, 1)
    }
    return null
  }
}
```

---

**2. Criando Validadores (VineJs)** ***"/app/validators/*.ts"***

**Comandos**

*(no terminal de comando)*

*Instalação*

```shell
node ace add vinejs
```

*Criação*

```shell
node ace make:validator course
```

*Arquivo: "/app/validators/course.ts"*

```typescript
import vine from '@vinejs/vine'

// Valida a criação dos cursos (create)
export const createCourseValidator = vine.compile(
  vine.object({
    name: vine.string().trim().minLength(4),
    duration: vine.number().positive().withoutDecimals(),
  })
)

// Valida a atualização dos cursos (update)
export const updateCourseValidator = vine.compile(
  vine.object({
    name: vine.string().trim().minLength(4),
    duration: vine.number().positive().withoutDecimals(),
  })
)
```

*(https://vinejs.dev/docs/introduction)*

---

**3. Criando Controllers *"/app/controllers/*.ts"***

**Comando**

*(no terminal de comando)*

Shell

```
# Classe de Controle Vazia - Sem métodos
node ace make:controller CoursesController

# Classe de Controle Pré-formatada - Com métodos de CRUD
node ace make:controller CoursesController --resource
```

*Implementação*

TypeScript

```
import { inject } from '@adonisjs/core'
import type { HttpContext } from '@adonisjs/core/http'
import { CourseService } from '#services/course_service'
import { createCourseValidator, updateCourseValidator } from '#validators/course'

/* Decorador necessário para o contêiner resolver as dependências
  Utiliza os recursos de reflexão do TypeScript para detectar dependências
  de construtor em tempo de execução */
@inject()
export default class CoursesController {
  constructor(protected courseService: CourseService) {}

  async index({ response }: HttpContext) {
    const courses = await this.courseService.getAllCourses()
    return response.status(201).json({
      message: 'OK',
      data: courses,
    })
  }
  async create({}: HttpContext) {}

  async store({ response, request }: HttpContext) {
    const payload = await request.validateUsing(createCourseValidator)
    const course = await this.courseService.createCourse(payload)
    if (!course) {
      return response.status(422).json({
        message: 'ERROR',
      })
    }
    return response.status(201).json({
      message: 'OK',
```

```
      data: course,
    })
  }

  async show({ params, response }: HttpContext) {
    const course = await
this.courseService.getCourseById(Number(params.id))
    if (!course) {
      return response.status(404).json({
        message: 'NOT FOUND',
      })
    }
    return response.status(201).json({
      message: 'OK',
      data: course,
    })
  }

  async edit({ params }: HttpContext) {}

  async update({ params, request, response }: HttpContext) {
    const payload = await request.validateUsing(updateCourseValidator)
    const course = await
this.courseService.updateCourse(Number(params.id), payload)
    if (!course) {
      return response.status(422).json({
        message: 'NOT FOUND',
      })
    }
    return response.status(201).json({
      message: 'OK',
      data: course,
    })
  }

  async destroy({ params, response }: HttpContext) {
    const course = this.courseService.deleteCourse(Number(params.id))
    if (!course) {
      return response.status(404).json({
        message: 'NOT FOUND',
      })
```

```
    }
    return response.status(201).json({
      message: 'OK',
      data: course,
    })
  }
}
```

---

## 4. Definindo Rotas *"/start/routes.ts"*

TypeScript

```
// Rotas para CRUD
/*
router.get('/courses', '#controllers/courses_controller.index')
router.get('/courses/:id', '#controllers/courses_controller.show')
router.post('/courses', '#controllers/courses_controller.store')
router.put('/courses/:id', '#controllers/courses_controller.update')
router.delete('/courses/:id', '#controllers/courses_controller.destroy')
*/
// Ou apenas
router.resource('courses', '#controllers/courses_controller')
```

---

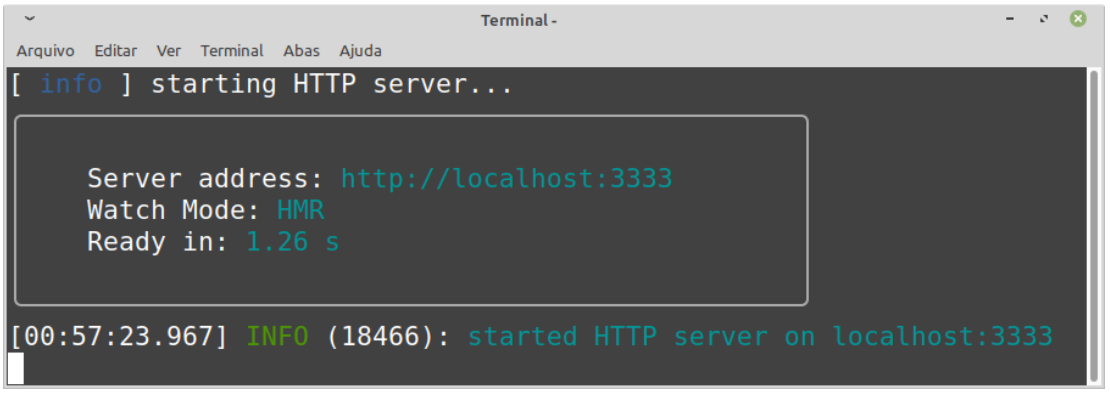
## 5. Executando e Testando a Aplicação

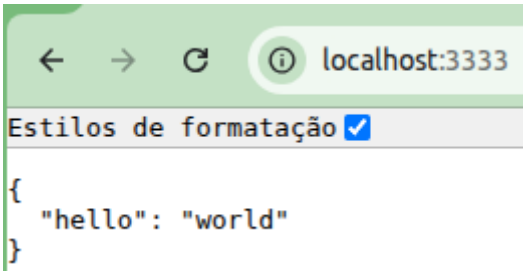
### 5.1 Iniciando o Servidor

*(no terminal de comando)*

Shell

```
node ace serve --hmr
#ou
npm run dev
```



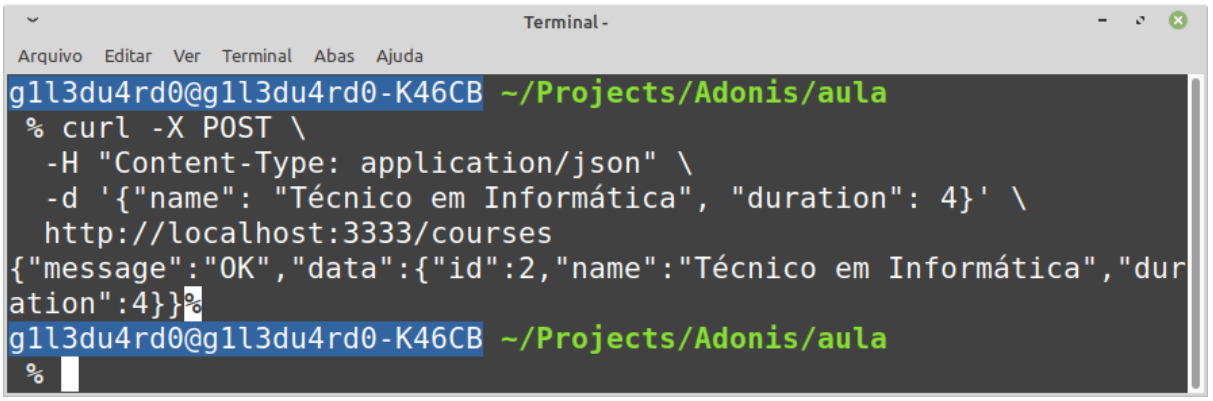




**5.2 Testando os Endpoints**

***Criar Curso***

*(no terminal de comando)*

```shell
curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"name": "Técnico em Informática", "duration": 4}' \
  http://localhost:3333/courses
```

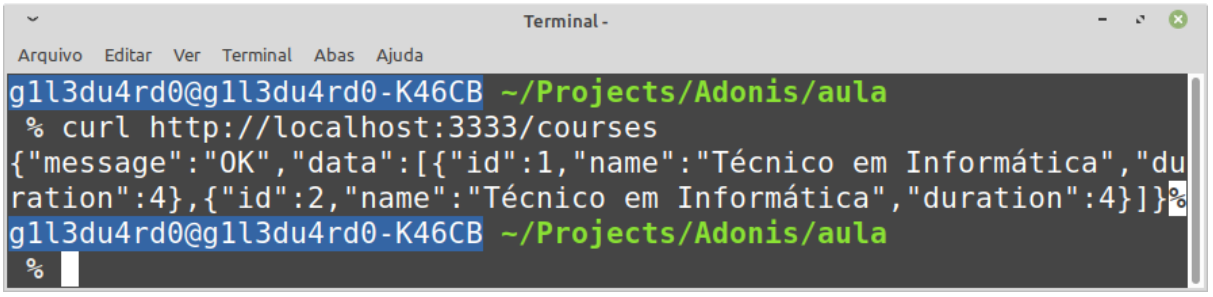


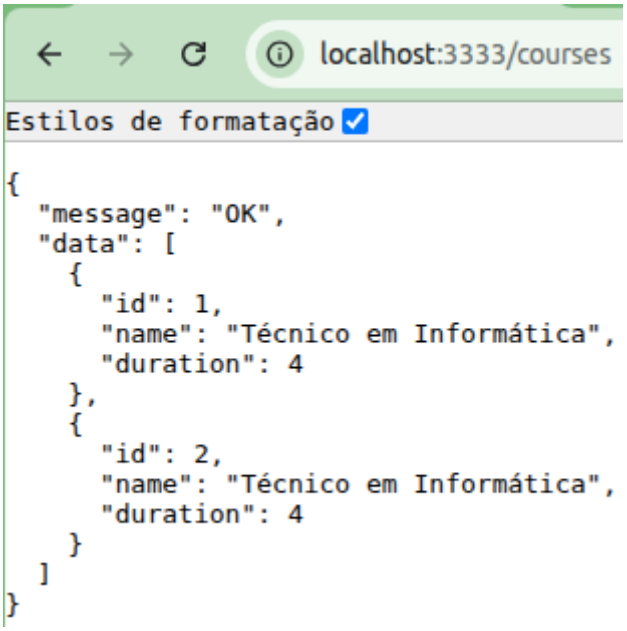

---

***Listar Cursos***

*(no terminal de comando)*

```
Shell
curl http://localhost:3333/courses
```

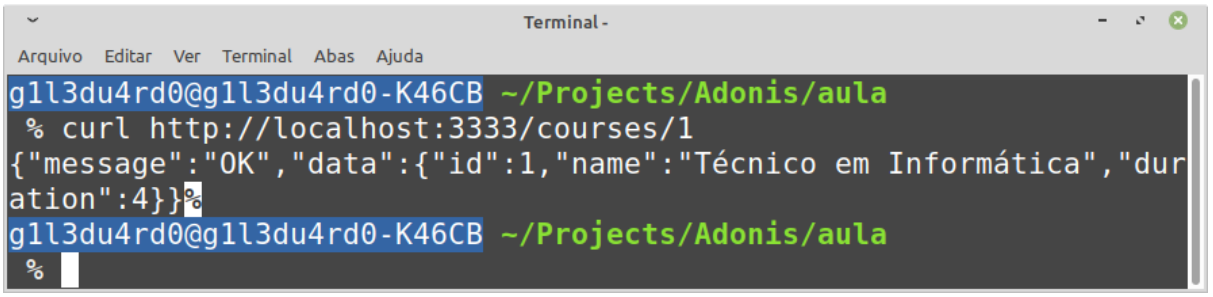


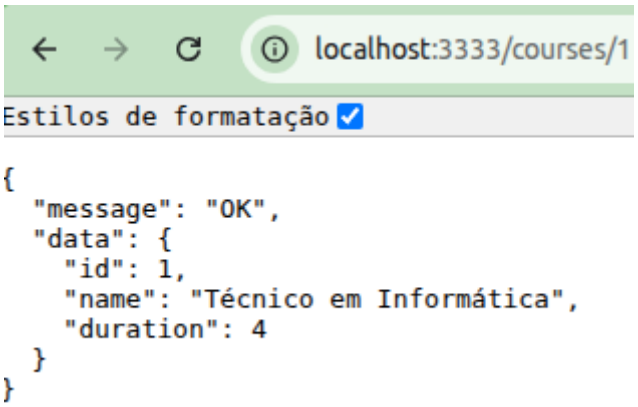




---

***Buscar Curso por ID***
*(no terminal de comando)*

```
Shell
curl http://localhost:3333/courses/1
```

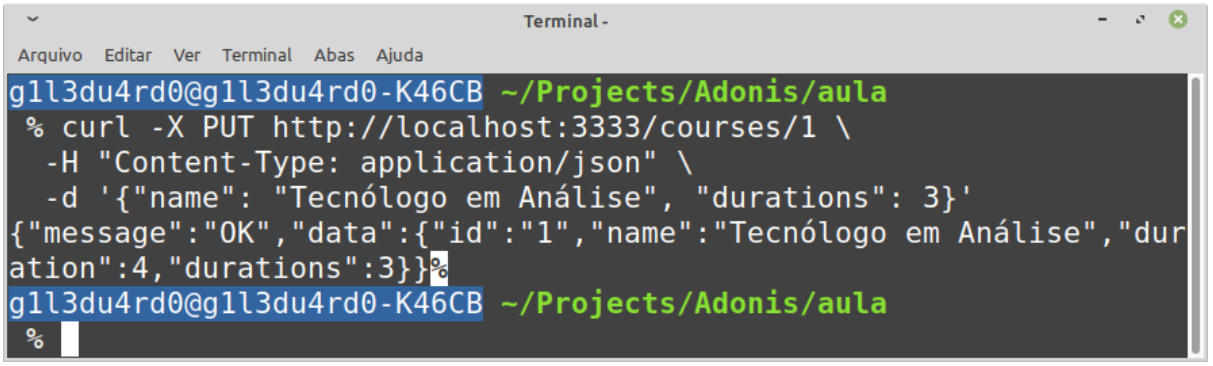



---

***Atualizar Curso por ID***
*(no terminal de comando)*

```shell
curl -X PUT http://localhost:3333/courses/1 \
  -H "Content-Type: application/json" \
  -d '{"name": "Tecnólogo em Análise", "durations": 3}'
```




---

***Remover Curso***
*(no terminal de comando)*

Shell

```
curl -X DELETE http://localhost:3333/courses/1
```