INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

ENSINO MÉDIO INTEGRADO: INFORMÁTICA
Disciplina de Desenvolvimento Web
Aula 17: Adonis JS - Migration / Seeder / Lucid ORM

Gil Eduardo de Andrade

Conceitos Preliminares

(https://adonisjs.com/)
(https://lucid.adonisjs.com/docs/migrations)
(https://lucid.adonisjs.com/docs/seeders)
(https://lucid.adonisjs.com/docs/crud-operations)
(https://lucid.adonisjs.com/docs/relationships)
(https://lucid.adonisjs.com/docs/model-query-builder)

Migration

As *"migrations"* permitem controlar a versão da base de dados que está sendo utilizada pela aplicação. Elas podem ser vistas como scripts independentes escritos em TypeScript para criar e alterar o esquema da base de dados ao longo do tempo.

Com funcionam as *Migrations*

As *migrations* funcionam da seguinte forma:

1. Um novo arquivo de *migration* é criado para cada alteração no esquema do banco de dados (criar ou alterar tabela);
2. Dentro do arquivo de *migration* são escritas as instruções que permitem realizar as ações necessárias;
3. As *migrations* são executadas usando a ferramenta de linha de comando do AdonisJS (ace);
4. O AdonisJS mantém o controle das migrations executadas, garantindo que cada migration seja executada apenas uma vez;
5. Durante o desenvolvimento, você também pode reverter migrations para editá-las.

Criando uma *Migration*

Uma nova *migration* pode ser criada através da execução do seguinte comando Ace.

```shell
node ace make:migration users

# CREATE: database/migrations/1630981615472_create_users_table.ts
```

Também é possível criar um modelo Lucid, juntamente com a *migration*, executando o comando com a flag *"-m"*

```shell
node ace make:model User -m
```

Estrutura da classe *Migration*
Uma classe de *migration* sempre estende a classe *"BaseSchema"* e deve implementar os métodos ***up*** e ***down***.

- O método ***up*** é usado para evoluir o esquema do banco de dados. Geralmente, você criará novas tabelas ou alterará tabelas existentes dentro deste método;

- O método ***down*** é usado para reverter as ações executadas pelo método ***up***. Por exemplo, se o método ***up*** cria uma tabela, o método ***down*** deve excluir a mesma tabela.

```typescript
import { BaseSchema } from '@adonisjs/lucid/schema'

export default class extends BaseSchema {
    protected tableName = 'users'

    async up() {
        this.schema.createTable(this.tableName, (table) => {
            table.increments('id')
            table.string('username').unique()
            table.string('email').unique()
            table.string('password')
            table.timestamp('created_at', { useTz: true })
```

```
            table.timestamp('updated_at', { useTz: true })
        })
    }

    async down() {
        this.schema.dropTable(this.tableName)
    }
}
```

Executando e Revertendo Migrations
Para executar as migrations, use o comando:

Shell
```
node ace migration:run
```

Para reverter migrations:

Shell
```
# Reverter o último lote
node ace migration:rollback

# Reverter até o início das migrations
node ace migration:rollback --batch=0

# Reverter até o lote 1
node ace migration:rollback --batch=1

# Reverter as últimas 3 migrations
node ace migration:rollback --step=3
```

Comandos úteis de Migration

```shell
# Reverter todas as migrations e executar novamente
node ace migration:refresh

# Reverter, migrar e executar seeders
node ace migration:refresh --seed

# Excluir todas as tabelas e migrar novamente
node ace migration:fresh

# Excluir todas as tabelas, migrar e executar seeders
node ace migration:fresh --seed

# Resetar todas as migrations
node ace migration:reset
```

Criando Tabelas

```typescript
import { BaseSchema } from '@adonisjs/lucid/schema'

export default class extends BaseSchema {
  protected tableName = 'posts'

  async up() {
    this.schema.createTable(this.tableName, (table) => {
      table.increments('id')
      table.string('title')
      table.text('content')
      table.string('status').defaultTo('draft')

      // Relacionamento (chave estrangeira)
      table.integer('user_id').unsigned().references
      ('id').inTable('users')

      table.timestamp('published_at', { useTz: true
      }).nullable()
```

```
            table.timestamp('created_at', { useTz: true })
            table.timestamp('updated_at', { useTz: true })
        })
    }

    async down() {
        this.schema.dropTable(this.tableName)
    }
}
```

Alterando Tabelas

TypeScript

```
export default class extends BaseSchema {
    protected tableName = 'users'

    async up() {
        this.schema.alterTable(this.tableName, (table) => {
            table.string('phone').nullable()
            table.date('birth_date').nullable()
            table.dropColumn('old_column')
        })
    }

    async down() {
        this.schema.alterTable(this.tableName, (table) => {
            table.dropColumn('phone')
            table.dropColumn('birth_date')
            table.string('old_column')
        })
    }
}
```

Renomeando Tabelas

```TypeScript
export default class extends BaseSchema {

      async up() {
            this.schema.renameTable('old_table_name',
      'new_table_name')
      }

      async down() {
            this.schema.renameTable('new_table_name',
      'old_table_name')
      }
}
```

Seeders (Povoamento de Dados)

Database seeding é uma forma de configurar sua aplicação com alguns dados iniciais necessários para executar e usar a aplicação. Os seeders são úteis para:

- Criar dados iniciais como países, estados e cidades antes de implantar e executar sua aplicação;
- Inserir usuários no banco de dados para desenvolvimento local;
- Criar dados de teste para desenvolvimento.

Criando um Seeder

Os seeders são armazenados no diretório database/seeders, e podem ser criados através da execução do seguinte comando Ace:

```Shell
node ace make:seeder User

# CREATE: database/seeders/user_seeder.ts
```

Estrutura do Seeder

Todo arquivo seeder deve estender a classe *"BaseSeeder"* e implementar o método run.

```typescript
import { BaseSeeder } from '@adonisjs/lucid/seeders'
import User from '#models/user'

export default class UserSeeder extends BaseSeeder {
     async run() {
          await User.createMany([
               {
                    username: 'admin',
                    email: 'admin@example.com',
                    password: 'secret',
               },
               {
                    username: 'user1',
                    email: 'user1@example.com',
                    password: 'secret',
               },
          ])
     }
}
```

Executando Seeders

Para executar todos ou seeders selecionados utiliza-se o seguinte comando Ace:

```shell
# Executa todos os seeders
node ace db:seed

# Executa um seeder específico
node ace db:seed --files "./database/seeders/user_seeder.ts"

# Executa seeders de forma interativa
node ace db:seed -i
```

Seeder com Relacionamentos

```typescript
import { BaseSeeder } from '@adonisjs/lucid/seeders'
import User from '#models/user'
import Post from '#models/post'

export default class PostSeeder extends BaseSeeder {
      async run() {
            // Buscar usuários existentes
            const users = await User.all()

            for (const user of users) {
                  // Criar posts para cada usuário
                  await user.related('posts').createMany([
                        {
                              title: `Post 1 do ${user.username}`,
                              content: 'Conteúdo do primeiro post',
                              status: 'published',
                        },
                        {
                              title: `Post 2 do ${user.username}`,
                              content: 'Conteúdo do segundo post',
                              status: 'draft',
                        },
                  ])
            }
      }
}
```

Seeder com Factory (se estiver usando Model Factories)

```typescript
import { BaseSeeder } from '@adonisjs/lucid/seeders'
import User from '#models/user'

export default class UserSeeder extends BaseSeeder {
      async run() {
            // Criar 10 usuários usando factory
            await User.factory().count(10).create()
      }
```

```
}
```

Models (Modelos)

Os modelos de dados do Lucid são construídos sobre o padrão ***Active Record (*)*** e facilitam muito a realização de operações CRUD e gerenciamento de relacionamentos entre modelos.

(*) Active Record é um padrão de projeto de ORM (Object-Relational Mapping) em que uma classe de objeto representa uma única linha (ou registo) de uma tabela de banco de dados, permitindo que os desenvolvedores manipulem a base de dados de forma orientada a objetos.

Criando um *Model*

É possível criar um modelo Lucid usando o comando Ace *"make:model"*:

Shell

```
node ace make:model User

# CREATE: app/Models/User.ts
```

Também é possível gerar uma *migration* vinculada com o modelo, utilizando a flag *"-m"*:

Shell

```
node ace make:model User -m

# CREATE: database/migrations/1618903673925_users.ts
# CREATE: app/Models/User.ts
```

Também é possível criar uma *factory* vinculada com o modelo, utilizando a flag *"-f"*:

Shell

```
node ace make:model User -f

# CREATE: app/Models/User.ts
# CREATE: database/factories/User.ts
```

Estrutura Básica de um *Model*

Todo modelo deve estender a classe *"BaseModel"* para herdar funcionalidades adicionais:

TypeScript

```
import { DateTime } from 'luxon'
import { BaseModel, column } from '@adonisjs/lucid/orm'

export default class User extends BaseModel {
      @column({ isPrimary: true })
      declare id: number

      @column()
      declare username: string

      @column()
      declare email: string

      @column({ serializeAs: null })
      declare password: string

      @column()
      declare avatarUrl: string | null

      @column.dateTime({ autoCreate: true })
      declare createdAt: DateTime

      @column.dateTime({ autoCreate: true, autoUpdate: true })
      declare updatedAt: DateTime
      }
}
```

Definindo Colunas

As colunas do banco de dados são definidas como propriedades na classe, e são decoradas pelo uso do *decorator* ***@column***.

Opções do Decorator @column

```typescript
export default class User extends BaseModel {
	// Chave primária
	@column({ isPrimary: true })
	declare id: number

	// Coluna normal
	@column()
	declare username: string

	// Coluna que não aparece na serialização
	@column({ serializeAs: null })
	declare password: string

	// Coluna com nome personalizado no banco
	@column({ columnName: 'user_email' })
	declare email: string

	// Coluna com preparação e consumo de dados
	@column({
		prepare: (value: string) => value.toLowerCase(),
		consume: (value: string) => value.toUpperCase(),
	})

	declare status: string
}
```

Colunas de Data

O Lucid permite aprimorar as propriedades de data e data-hora, convertendo os valores do driver do banco de dados para uma instância de "*luxon.DateTime*":

```typescript
import { DateTime } from 'luxon'
import { BaseModel, column } from '@adonisjs/lucid/orm'

export default class User extends BaseModel {
      @column.date()
      declare birthDate: DateTime

      @column.dateTime({ autoCreate: true })
      declare createdAt: DateTime

      @column.dateTime({ autoCreate: true, autoUpdate: true })
      declare updatedAt: DateTime

      // Para usar Date nativo ao invés de Luxon
      @column({
            consume: (value: string) => new Date(value),
            prepare: (value: Date) => value.toISOString(),
      })

      declare lastLogin: Date
}
```

Configurações do Model

Nome da Tabela Personalizado

```typescript
export default class User extends BaseModel {
      static table = 'app_users'
}
```

Chave Primária Personalizada

```typescript
export default class User extends BaseModel {
```

```
      static primaryKey = 'uuid'

      @column({ isPrimary: true })
      declare uuid: string
}
```

Auto-atribuição de Chave Primária

TypeScript
```
import { randomUUID } from 'node:crypto'
import { BaseModel, beforeCreate, column } from '@adonisjs/lucid/orm'
export default class User extends BaseModel {

      static selfAssignPrimaryKey = true

      @column({ isPrimary: true })
      declare id: string

      @beforeCreate()
      static assignUuid(user: User) {
            user.id = randomUUID()
      }
}
```

Hooks de Ciclo de Vida

Os hooks permitem executar código em pontos específicos do ciclo de vida do modelo:

TypeScript
```
import {
      BaseModel,
      beforeSave,
      afterCreate,
      beforeCreate,
      afterUpdate
```

```
}
from '@adonisjs/lucid/orm'
import Hash from '@adonisjs/core/services/hash'

export default class User extends BaseModel {

      @column({ serializeAs: null })
      declare password: string

      // Executado antes de salvar (criar ou atualizar)
      @beforeSave()
      static async hashPassword(user: User) {
            if (user.$dirty.password) {
                  user.password = await Hash.make(user.password)
            }
      }

      // Executado antes de criar
      @beforeCreate()
            static async setDefaults(user: User) {
            user.status = user.status || 'active'
      }

      // Executado após criar
      @afterCreate()
      static async sendWelcomeEmail(user: User) {
            // Enviar email de boas-vindas
            console.log(`Enviando email de boas-vindas para ${user.email}`)
      }

      // Executado após atualizar
      @afterUpdate()
            static async logUpdate(user: User) {
            console.log(`Usuário ${user.id} foi atualizado`)
      }
}
```

Hooks Disponíveis

TypeScript

```typescript
import {
	BaseModel,
	beforeSave,
	afterSave,
	beforeCreate,
	afterCreate,
	beforeUpdate,
	afterUpdate,
	beforeDelete,
	afterDelete,
	beforeFind,
	afterFind,
	beforeFetch,
	afterFetch
} from '@adonisjs/lucid/orm'

export default class User extends BaseModel {

	@beforeSave()
	static async beforeSaveHook(user: User) {
		// Executado antes de salvar (criar ou atualizar)
	}

	@afterSave()
	static async afterSaveHook(user: User) {
		// Executado após salvar (criar ou atualizar)
	}

	@beforeCreate()
	static async beforeCreateHook(user: User) {
		// Executado antes de criar
	}

	@afterCreate()
	static async afterCreateHook(user: User) {
		// Executado após criar
```

```
    }

    @beforeUpdate()
    static async beforeUpdateHook(user: User) {
        // Executado antes de atualizar
    }

    @afterUpdate()
    static async afterUpdateHook(user: User) {
        // Executado após atualizar
    }

    @beforeDelete()
    static async beforeDeleteHook(user: User) {
        // Executado antes de deletar
    }

    @afterDelete()
    static async afterDeleteHook(user: User) {
        // Executado após deletar
    }
}
```

Model Factories (Fábrica de Modelos)

As *Model Factories* são uma ferramenta poderosa para gerar dados falsos em massa, especialmente úteis durante testes, possibilitando povoar o banco de dados com dados aleatórios. Com as *factories*, é possível extrair toda a configuração de dados de teste para um arquivo dedicado e escrever o mínimo de código necessário para configurar o estado do banco de dados.

Criando Factories

As *Model Factories* são armazenadas no diretório database/factories. Você pode definir todas as *factories* em um único arquivo ou criar arquivos dedicados para cada modelo.

```shell
node ace make:factory User

# CREATE: database/factories/user.ts
```

Também é possível criar uma *factory* vinculada ao modelo, usando a flag *"-f"*:

```shell
node ace make:model User -f

# CREATE: app/Models/User.tsc
# CREATE: database/factories/user.ts
```

Estrutura Básica de uma Factory

```typescript
import User from '#models/user'
import Factory from '@adonisjs/lucid/factories'

export const UserFactory = Factory.define(User, ({ faker }) => {
      return {
            username: faker.internet.userName(),
            email: faker.internet.email(),
            password: faker.internet.password(),
            firstName: faker.person.firstName(),
            lastName: faker.person.lastName(),
            birthDate: faker.date.birthdate(),
      }
}).build()
```

Usando Factories

Criar um único registro:

```typescript
import { UserFactory } from '#database/factories/user'

const user = await UserFactory.create()
console.log(user.email) // email gerado pelo faker
```

Criar múltiplos registros

```typescript
const users = await UserFactory.createMany(10)
console.log(users.length) // 10
```

Factory States (Estados)

Os states permitem definir variações das suas factories. Por exemplo, em uma *factory* de *"Post"*, é possível ter diferentes states para representar posts publicados e rascunhos.

```typescript
import Post from '#models/post'
import Factory from '@adonisjs/lucid/factories'

export const PostFactory = Factory.define(Post, ({ faker }) => {
      return {
            title: faker.lorem.sentence(),
            content: faker.lorem.paragraphs(4),
            status: 'draft',
            viewCount: 0,
            }
})
.state('published', (post) => {
      post.status = 'published'
      post.publishedAt = new Date()
})
.state('popular', (post) => {
      post.viewCount = faker.number.int({ min: 1000, max: 10000})
})
```

```
.build()
```

Usando States

```typescript
// Criar posts com state padrão (draft)
const draftPosts = await PostFactory.createMany(3)

// Criar posts publicados (published)
const  publishedPosts = await
      PostFactory.apply('published').createMany(3)

// Criar posts populares e publicados
const popularPosts = await PostFactory
      .apply('published')
      .apply('popular')
      .createMany(2)
```

Relacionamentos com Factories

As factories facilitam o trabalho com relacionamentos entre modelos.

```typescript
import Post from '#models/post'
import Comment from '#models/comment'
import Factory from '@adonisjs/lucid/factories'

export const CommentFactory = Factory.define(Comment, ({ faker }) => {
      return {
            content: faker.lorem.paragraph(),
            authorName: faker.person.fullName(),
      }
}).build()

export const PostFactory = Factory.define(Post, ({ faker }) => {
      return {
```

```
            title: faker.lorem.sentence(),
            content: faker.lorem.paragraphs(4),
            status: 'draft',
      }
})
.relation('comments', () => CommentFactory)
.build()

export const UserFactory = Factory.define(User, ({ faker }) => {
      return {
            username: faker.internet.userName(),
            email: faker.internet.email(),
            password: faker.internet.password(),
      }
})
.relation('posts', () => PostFactory)
.relation('profile', () => ProfileFactory)
.build()
```

Criando Modelos com Relacionamentos

TypeScript

```
// Criar usuário com 3 posts
const user = await UserFactory.with('posts', 3).create()
console.log(user.posts.length) // 3

// Criar usuário com posts e comentários
const user = await UserFactory.with(
      'posts', 2, (post) => post.with('comments', 5)
)
.create()

// Criar usuário com posts publicados
const user = await UserFactory.with(
      'posts', 3, (post) => post.apply('published')
)
```

```
.create()

// Criar usuário com posts mistos
const user = await UserFactory
.with('posts', 2, (post) => post.apply('published'))
.with('posts', 3) // posts em draft
.create()
```

Lucid ORM

O Lucid é um ORM (*Object-Relational Mapper*) para Node.js, construído sobre o Knex.js. Ele faz parte do framework AdonisJS, mas pode ser usado de forma independente. Ele implementa o padrão *Active Record*, que facilita a interação com o banco de dados de forma orientada a objetos.
Esta aula tem como objetivo apresentar os métodos básicos de CRUD (Create, Read, Update, Delete) e os métodos de relacionamento.

Métodos Básicos (CRUD)

Os métodos de CRUD (Create, Read, Update, Delete) são a base de qualquer ORM. O Lucid oferece uma API intuitiva para realizar essas operações.

Create (Criar)

Para criar novos registros no banco de dados, podem ser usados os métodos *"create"*, *"createMany"*, ou instanciar um novo modelo e usar o método *"save"*.

create(data)
Cria e persiste um novo registro no banco de dados.

JavaScript
```
import User from '#models/user'

const user = await User.create({
      username: 'virk',
      email: 'virk@adonisjs.com',
})
```

createMany(data)
Cria e persiste múltiplos registros no banco de dados.

```JavaScript
const users = await User.createMany([
      {
            email: 'virk@adonisjs.com',
            password: 'secret',
      },
      {
            email: 'romain@adonisjs.com',
            password: 'secret',
      },
])
```

new Model() + save()
Cria uma nova instância do modelo, atribui os valores e então persiste no banco de dados.

```JavaScript
import User from '#models/user'

const user = new User()
user.username = 'virk'
user.email = 'virk@adonisjs.com'
await user.save()
```

Read (Ler)
O Lucid oferece vários métodos para buscar dados do banco de dados.

all()
Busca todos os registros da tabela.

```JavaScript
const users = await User.all()
```

find(id)
Busca um registro pela sua chave primária.

```javascript
const user = await User.find(1)
```

findBy(column, value)
Busca um registro por uma coluna específica.

```javascript
const user = await User.findBy('email', 'virk@adonisjs.com')
```

findMany(ids)
Busca múltiplos registros por um array de chaves primárias.

```javascript
const users = await User.findMany([1, 2])
```

first()
Busca o primeiro registro da tabela.

```javascript
const user = await User.first()
```

findOrFail(id) , firstOrFail() , findByOrFail(column, value)
As variações apresentadas acima, para os métodos de busca, lançam uma exceção E_ROW_NOT_FOUND se nenhum registro for encontrado.

```javascript
const user = await User.findOrFail(1)
```

Update (Atualizar)
Para atualizar um registro, é necessário, primeiro, buscá-lo no banco de dados. Após essa etapa, os valores obtidos podem ser alterados e o método *"save"* utilizado para consolidar a mudança na base de dados.

```javascript
const user = await User.findOrFail(1)
user.username = 'lucid'
await user.save()
```

Também é possível usar o método *"merge"* para atribuir múltiplos valores de uma vez.

```javascript
const user = await User.findOrFail(1)
await user.merge({ username: 'lucid' }).save()
```

Delete (Deletar)
Para deletar um registro, é necessário, primeiro, buscálo na base de dados, para então usar o método *"delete"*.

```javascript
const user = await User.findOrFail(1)
await user.delete()
```

Para deletar múltiplos registros, é possível usar o query builder.

```javascript
await User.query().where('isVerified', false).delete()
```

Métodos de Relacionamento

O Lucid ORM oferece um poderoso sistema de relacionamentos para definir e consultar dados relacionados.

HasOne (Um para Um)

Define um relacionamento ***um-para-um***. Por exemplo, um *"User"* tem um *"Profile"*.

```typescript
// app/models/User.ts

import Profile from '#models/profile'
import type { HasOne } from '@adonisjs/lucid/types/relations'
import { BaseModel, column, hasOne } from '@adonisjs/lucid/orm'

export default class User extends BaseModel {
      @column({ isPrimary: true })
      declare id: number

      @hasOne(() => Profile)
      declare profile: HasOne<typeof Profile>
}
```

HasMany (Um para Muitos)

Define um relacionamento ***um-para-muitos***. Por exemplo, um *"User"* tem muitos *"Posts"*.

```typescript
// app/models/User.ts

import Post from '#models/post'
import type { HasMany } from '@adonisjs/lucid/types/relations'
import { BaseModel, column, hasMany } from '@adonisjs/lucid/orm'

export default class User extends BaseModel {
      @column({ isPrimary: true })
      declare id: number

      @hasMany(() => Post)
      declare posts: HasMany<typeof Post>
}
```

BelongsTo (Pertence a)

Define o inverso de um relacionamento hasOne ou hasMany. Por exemplo, um *"Post"* pertence a um *"User"*.

```typescript
// app/models/Post.ts

import User from '#models/user'
import type { BelongsTo } from '@adonisjs/lucid/types/relations'
import { BaseModel, column, belongsTo } from '@adonisjs/lucid/orm'

export default class Post extends BaseModel {
      @column({ isPrimary: true })
      declare id: number

      @column()
      declare userId: number

      @belongsTo(() => User)
      declare user: BelongsTo<typeof User>
}
```

ManyToMany (Muitos para Muitos)

Define um relacionamento ***muitos-para-muitos***. Por exemplo, um *"User"* pode ter muitas *"Skills"*, e uma *"Skill"* pode pertencer a muitos *"Users"*. Este relacionamento requer uma tabela *"pivot"*.

```typescript
// app/models/User.ts

import Skill from '#models/skill'
import type { ManyToMany } from '@adonisjs/lucid/types/relations'
import { BaseModel, column, manyToMany } from '@adonisjs/lucid/orm'

export default class User extends BaseModel {
      @column({ isPrimary: true })
```

```
    declare id: number

    @manyToMany(() => Skill)
    declare skills: ManyToMany<typeof Skill>
}
```

Consultando Relacionamentos

O Lucid oferece métodos para carregar e filtrar relacionamentos de forma eficiente.

preload(relationship)

Carrega os dados de um relacionamento junto com a consulta principal.

```typescript
const users = await User.query().preload('posts')
users.forEach(user => {
    console.log(user.posts) // Array de posts do usuário
})
```

Também é possível adicionar *"constraints"* (restrições) à consulta do relacionamento.

```typescript
const users = await User.query().preload('posts', (postsQuery) => {
    postsQuery.where('status', 'published')
})
```

whereHas(relationship, callback)

Filtra os resultados da consulta principal com base na existência de um relacionamento que satisfaça uma determinada condição.

```typescript
// Busca usuários que têm posts publicados
```

```
const users = await User.query().whereHas('posts', (postsQuery) => {
      postsQuery.where('status', 'published')
})
```

has(relationship)
Filtra os resultados da consulta principal com base na existência de um relacionamento.

```
TypeScript
// Busca usuários que têm pelo menos um post
const users = await User.query().has('posts')

// Busca usuários que têm pelo menos 2 posts
const users = await User.query().has('posts', '>=', 2)
```

Métodos Idempotentes

O Lucid oferece métodos que simplificam a criação de registros verificando primeiro se eles já existem.

firstOrCreate(searchPayload, savePayload)
Busca um registro no banco de dados ou cria um novo se não encontrar.

```
TypeScript
const user = await User.firstOrCreate(
      { email: 'virk@adonisjs.com' },
      { password: 'secret' }
)
```

updateOrCreate(searchPayload, updatePayload)
Busca um registro e o atualiza, ou cria um novo se não encontrar.

```typescript
const user = await User.updateOrCreate(
      { email: 'virk@adonisjs.com' },
      { lastLoginAt: DateTime.local() }
)
```

Query Builder Avançado

O Lucid oferece um query builder poderoso que permite construir consultas SQL complexas.

```typescript
const users = await User
      .query()
      .where('countryCode', 'BR')
      .orWhereNull('countryCode')
      .orderBy('createdAt', 'desc')
      .limit(10)
```

Paginação

O Lucid oferece suporte nativo à paginação.

```typescript
const users = await User.query().paginate(1, 20)

console.log(users.total) // Total de registros
console.log(users.perPage) // Registros por página
console.log(users.currentPage) // Página atual
```

Transações

Para operações que envolvem múltiplas consultas, é possível utilizar transações.

```typescript
import Database from '@adonisjs/lucid/services/db'

const trx = await Database.transaction()
try {
      const user = await User.create({
            email: 'virk@adonisjs.com'
      }, { client: trx })

      await user.related('profile').create({
            fullName: 'Harminder Virk'
      }, { client: trx })

      await trx.commit()
} catch(error) {
      await trx.rollback()
      throw error
}
```

CRIANDO UMA APLICAÇÃO SIMPLES

Passo a Passo

Criando Projeto Aula

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

---

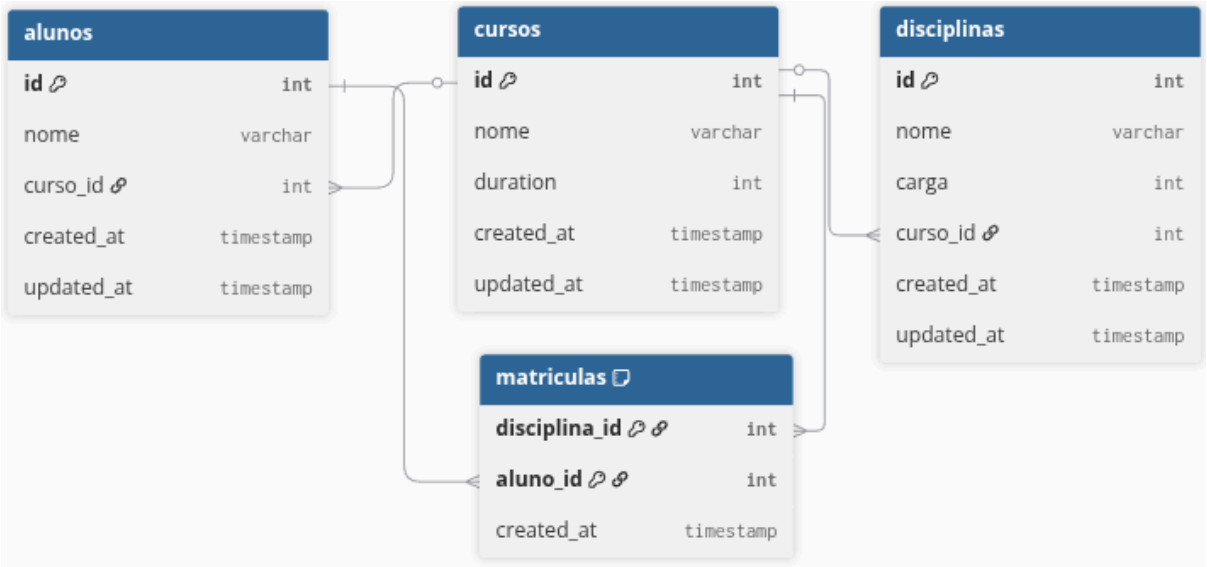
1. Configurando Variáveis de Ambiente (BD) *“.env”*

```typescript
TZ=UTC
```

```
PORT=3333
HOST=localhost
LOG_LEVEL=info
APP_KEY=fap2wDv8xLQoUyeYuPF89a7WFV2s2jZY
NODE_ENV=development
DB_HOST=127.0.0.1
DB_PORT=3306
DB_USER=seu_usuario
DB_PASSWORD=sua_senha
DB_DATABASE=sua_base
```

---

2. Definição do Diagrama do Banco de Dados



---

3. Criando as Models e Migrações *"/app/models/*.ts"* e *"/database/migrations/*.ts"*

Comandos

(no terminal de comando)

Shell

```
node ace make:model Curso -m
node ace make:model Disciplina -m
node ace make:model Aluno -m
node ace make:model Matricula -m
```

Arquivo: "/database/migrations/…_create_cursos_table.ts"

TypeScript

```
async up() {
    this.schema.createTable(this.tableName, (table) => {
      table.increments('id')
      table.string('nome')
      table.integer('duracao')
      table.timestamp('created_at')
      table.timestamp('updated_at')
    })
}
```

Arquivo: "/app/models/curso.ts"

TypeScript

```
import { DateTime } from 'luxon'
import { BaseModel, column, hasMany } from '@adonisjs/lucid/orm'
import Aluno from './aluno.js'
import Disciplina from './disciplina.js'
import type { HasMany } from '@adonisjs/lucid/types/relations'

export default class Curso extends BaseModel {
  @column({ isPrimary: true })
  declare id: number

  @column()
  declare nome: string

  @column()
  declare duracao: number

  @column.dateTime({ autoCreate: true })
  declare createdAt: DateTime
```

```
  @column.dateTime({ autoCreate: true, autoUpdate: true })
  declare updatedAt: DateTime

  // Relacionamentos
  @hasMany(() => Aluno, { foreignKey: 'curso_id' })
  declare alunos: HasMany<typeof Aluno>

  @hasMany(() => Disciplina, { foreignKey: 'curso_id' })
  declare disciplinas: HasMany<typeof Disciplina>
}
```

Arquivo: "/database/migrations/…_create_disciplinas_table.ts"

TypeScript

```
async up() {
    this.schema.createTable(this.tableName, (table) => {
      table.increments('id')
      table.string('nome')
      table.integer('carga')

      table.integer('curso_id').unsigned().references('id')
            .inTable('cursos')

      table.timestamp('created_at')
      table.timestamp('updated_at')
    })
}
```

Arquivo: "/app/models/disciplina.ts"

TypeScript

```
import { DateTime } from 'luxon'
import { BaseModel, column, belongsTo, manyToMany } from '@adonisjs/lucid/orm'
import Curso from './curso.js'
import Aluno from './aluno.js'
import type { BelongsTo, ManyToMany } from '@adonisjs/lucid/types/relations'

export default class Disciplina extends BaseModel {
  @column({ isPrimary: true })
```

```
  declare id: number

  @column()
  declare nome: string

  @column()
  declare carga: number

  @column()
  declare curso_id: number

  @column.dateTime({ autoCreate: true })
  declare createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  declare updatedAt: DateTime

  // Relacionamentos
  @belongsTo(() => Curso, { foreignKey: 'curso_id' })
  declare curso: BelongsTo<typeof Curso>

  @manyToMany(() => Aluno, { pivotTable: 'matriculas' })
  declare alunos: ManyToMany<typeof Aluno>
}
```

Arquivo: "/database/migrations/…_create_alunos_table.ts"

TypeScript

```
async up() {
    this.schema.createTable(this.tableName, (table) => {
      table.increments('id')
      table.string('nome')

      table.integer('curso_id').unsigned().references('id')
            .inTable('cursos')

      table.timestamp('created_at')
      table.timestamp('updated_at')
    })
}
```

Arquivo: "/app/models/aluno.ts"

TypeScript

```typescript
import { DateTime } from 'luxon'
import { BaseModel, column, belongsTo, manyToMany } from '@adonisjs/lucid/orm'
import Curso from './curso.js'
import Disciplina from './disciplina.js'
import type { BelongsTo, ManyToMany } from '@adonisjs/lucid/types/relations'

export default class Aluno extends BaseModel {
  @column({ isPrimary: true })
  declare id: number

  @column()
  declare nome: string

  @column()
  declare curso_id: number

  @column.dateTime({ autoCreate: true })
  declare createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  declare updatedAt: DateTime

  // Relacionamentos
  @belongsTo(() => Curso, { foreignKey: 'curso_id' })
  declare curso: BelongsTo<typeof Curso>

  @manyToMany(() => Disciplina, { pivotTable: 'matriculas' })
  declare disciplinas: ManyToMany<typeof Disciplina>
}
```

Arquivo: "/database/migrations/…_create_matriculas_table.ts"

TypeScript

```typescript
async up() {
    this.schema.createTable(this.tableName, (table) => {
      table.integer('aluno_id').unsigned().references('id')
            .inTable('alunos').primary()

      table.integer('disciplina_id').unsigned().references('id')
```

```
            .inTable('disciplinas').primary()

      table.timestamp('created_at')
    })
}
```

Arquivo: "/app/models/matricula.ts"

TypeScript

```typescript
import { DateTime } from 'luxon'
import { BaseModel, column, belongsTo } from '@adonisjs/lucid/orm'
import Aluno from './aluno.js'
import Disciplina from './disciplina.js'
import type { BelongsTo } from '@adonisjs/lucid/types/relations'

export default class Matricula extends BaseModel {
  @column({ isPrimary: true })
  declare aluno_id: number

  @column({ isPrimary: true })
  declare disciplina_id: number

  @column.dateTime({ autoCreate: true })
  declare createdAt: DateTime

  // Relacionamentos
  @belongsTo(() => Aluno, { foreignKey: 'aluno_id' })
  declare aluno: BelongsTo<typeof Aluno>

  @belongsTo(() => Disciplina, { foreignKey: 'disciplina_id' })
  declare disciplina: BelongsTo<typeof Disciplina>
}
```

---

4. Criando Seeders (Curso / Disciplina / Aluno)

Comando

(no terminal de comando)

Shell

```
node ace make:seeder 01_Curso
node ace make:seeder 02_Disciplina
node ace make:seeder 03_Aluno
```

Arquivo: "/database/seeders/01_curso_seeder.ts"

TypeScript

```typescript
import { BaseSeeder } from '@adonisjs/lucid/seeders'
import Curso from '#models/curso'

export default class extends BaseSeeder {
  async run() {
    await Curso.createMany([
      {
        nome: 'Técnico em Informática',
        duracao: 4,
      },
      {
        nome: 'Técnólogo em Análise e Desenvolvimento',
        duracao: 3,
      },
    ])
  }
}
```

Arquivo: "/database/seeders/02_disciplina_seeder.ts"

TypeScript

```typescript
import { BaseSeeder } from '@adonisjs/lucid/seeders'
import Curso from '#models/curso'
import Disciplina from '#models/disciplina'

export default class extends BaseSeeder {
  async run() {
    const cursos = await Curso.all()
    for (const curso of cursos) {
      // Criar disciplinas para cada curso
      // EMI
      if (curso.id === 1) {
```

```
        await Disciplina.createMany([
          {
            nome: 'Linguagem de Programação',
            carga: 4,
            curso_id: curso.id,
          },
          {
            nome: 'Matemática I',
            carga: 3,
            curso_id: curso.id,
          },
        ])
      }
      //TADS
      else {
        await Disciplina.createMany([
          {
            nome: 'Programação Paralela e Distribuída',
            carga: 2,
            curso_id: curso.id,
          },
          {
            nome: 'Desenvolvimento Mobile',
            carga: 4,
            curso_id: curso.id,
          },
        ])
      }
    }
  }
}
```

Arquivo: "/database/seeders/aluno_seeder.ts"

TypeScript

```
import { BaseSeeder } from '@adonisjs/lucid/seeders'
import Curso from '#models/curso'
import Aluno from '#models/aluno'

export default class extends BaseSeeder {
```

```
  async run() {
    const cursos = await Curso.all()

    for (const curso of cursos) {
      // Criar alunos para cada curso
      await Aluno.createMany([
        {
          nome: `Ana Luisa (${curso.nome})`,
          curso_id: curso.id,
        },
        {
          nome: `Luiz Eduardo (${curso.nome})`,
          curso_id: curso.id,
        },
      ])
    }
  }
}
```

---

5. Efetuando a Migração
Comando
(no terminal de comando)

Shell

```
node ace migration:run
#ou
node ace migration:fresh --seed
```

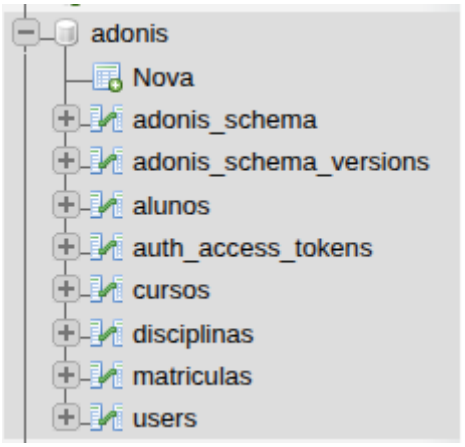


Tabela de Cursos

+ Opções

	id	nome	duracao	created_at	updated_at
Edita Copiar Apagar	1	Técnico em Informática	4	2025-09-07 19:57:00	2025-09-07 19:57:00
Edita Copiar Apagar	2	Técnólogo em Análise e Desenvolvimento	3	2025-09-07 19:57:00	2025-09-07 19:57:00

Tabela de Disciplinas

+ Opções

	id	nome	carga	curso_id	created_at	updated_at
Edita Copiar Apagar	1	Programação Paralela e Distribuída	2	2	2025-09-07 19:57:00	2025-09-07 19:57:00
Edita Copiar Apagar	2	Desenvolvimento Mobile	4	2	2025-09-07 19:57:00	2025-09-07 19:57:00
Edita Copiar Apagar	3	Linguagem de Programação	4	1	2025-09-07 19:57:00	2025-09-07 19:57:00
Edita Copiar Apagar	4	Matemática I	3	1	2025-09-07 19:57:00	2025-09-07 19:57:00

Tabela de Alunos

+ Opções

	id	nome	curso_id ▲ 1	created_at	updated_at
Edita Copiar Apagar	3	Ana Luisa (Técnico em Informática)	1	2025-09-07 19:57:00	2025-09-07 19:57:00
Edita Copiar Apagar	4	Luiz Eduardo (Técnico em Informática)	1	2025-09-07 19:57:00	2025-09-07 19:57:00
Edita Copiar Apagar	1	Ana Luisa (Técnólogo em Análise e Desenvolvimento)	2	2025-09-07 19:57:00	2025-09-07 19:57:00
Edita Copiar Apagar	2	Luiz Eduardo (Técnólogo em Análise e Desenvolvimen...	2	2025-09-07 19:57:00	2025-09-07 19:57:00

---

6. Criando os *Validators*

Comando

(no terminal de comando)

Shell

```
node ace make:validator Curso
node ace make:validator Disciplina
```

```
node ace make:validator Aluno
node ace make:validator Matricula
```

Arquivo: "/app/validators/curso.ts"

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

// Valida a criação dos cursos (create)
export const createCurso = vine.compile(
  vine.object({
    nome: vine.string().trim().minLength(4),
    duracao: vine.number().positive().withoutDecimals(),
  })
)
// Valida a atualização dos cursos (update)
export const updateCurso = vine.compile(
  vine.object({
    nome: vine.string().trim().minLength(4).optional(),
    duracao: vine.number().positive().withoutDecimals().optional(),
  })
)
```

Arquivo: "/app/validators/disciplina.ts"

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

// Valida a criação das disciplinas (create)
export const createDisciplina = vine.compile(
  vine.object({
    nome: vine.string().trim().minLength(4),
    carga: vine.number().positive().withoutDecimals(),
    curso_id: vine.number().positive().withoutDecimals(),
  })
)
// Valida a atualização das disciplinas (update)
export const updateDisciplina = vine.compile(
  vine.object({
```

```
    nome: vine.string().trim().minLength(4).optional(),
    carga: vine.number().positive().withoutDecimals().optional(),
    curso_id: vine.number().positive().withoutDecimals().optional(),
  })
)
```

Arquivo: "/app/validators/aluno.ts"

TypeScript

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

// Valida a criação dos alunos (create)
export const createAluno = vine.compile(
  vine.object({
    nome: vine.string().trim().minLength(4),
    curso_id: vine.number().positive().withoutDecimals(),
  })
)
// Valida a atualização dos alunos (update)
export const updateAluno = vine.compile(
  vine.object({
    nome: vine.string().trim().minLength(4).optional(),
    curso_id: vine.number().positive().withoutDecimals().optional(),
  })
)
```

Arquivo: "/app/controllers/matricula.ts"

TypeScript

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

// Valida a criação das matriculas (create)
export const createMatricula = vine.compile(
  vine.object({
    aluno_id: vine.number().positive().withoutDecimals(),
    disciplina_id: vine.number().positive().withoutDecimals(),
  })
)
```

7. Criando as *Controllers*

Comando

(no terminal de comando)

Shell

```
node ace make:controller CursoController --resource
node ace make:controller DisciplinaController --resource
node ace make:controller AlunoController --resource
node ace make:controller MatriculaController --resource
```

Arquivo: "/app/controllers/cursos_controller.ts"

TypeScript

```typescript
import type { HttpContext } from '@adonisjs/core/http'
import Curso from '#models/curso'
import { createCurso, updateCurso } from '#validators/curso'

export default class CursosController {
  /**
   * Display a list of resource
   */
  async index({ response }: HttpContext) {
    try {
      // const cursos = await Curso.all()
      const cursos = await Curso.query().preload('disciplinas')
            .preload('alunos')

      return response.status(200).json({
        message: 'OK',
        data: cursos,
      })
    } catch (error) {
      return response.status(500).json({
        message: 'ERROR',
      })
    }
  }

  /**
   * Display form to create a new record
```

```
 */
async create({}: HttpContext) {}

/**
 * Handle form submission for the create action
 */
async store({ request, response }: HttpContext) {
  const payload = await request.validateUsing(createCurso)
  try {
    const curso = await Curso.create({
      ...payload,
    })
    return response.status(201).json({
      message: 'OK',
      data: curso,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}

/**
 * Show individual record
 */
async show({ params, response }: HttpContext) {
  try {
    // const curso = await Curso.findOrFail(params.id)
    const curso = await Curso.query()
          .where('id', params.id).preload('disciplinas')
          .preload('alunos').firstOrFail();

    return response.status(200).json({
      message: 'OK',
      data: curso,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
```

```
  }
}

/**
 * Edit individual record
 */
async edit({ params }: HttpContext) {}

/**
 * Handle form submission for the edit action
 */
async update({ params, request, response }: HttpContext) {
  const payload = await request.validateUsing(updateCurso)
  try {
    const curso = await Curso.findOrFail(params.id)
    await curso.merge({ ...payload }).save()

    return response.status(200).json({
      message: 'OK',
      data: curso,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}

/**
 * Delete record
 */
async destroy({ params, response }: HttpContext) {
  try {
    const curso = await Curso.findOrFail(params.id)
    await curso.delete()

    return response.status(200).json({
      message: 'OK',
    })
  } catch (error) {
    return response.status(500).json({
```

```
        message: 'ERROR',
      })
    }
  }
}
```

Arquivo: "/app/controllers/disciplinas_controller.ts"

TypeScript

```
import type { HttpContext } from '@adonisjs/core/http'
import Disciplina from '#models/disciplina'
import { createDisciplina, updateDisciplina } from '#validators/disciplina'

export default class DisciplinasController {
  /**
   * Display a list of resource
   */
  async index({ response }: HttpContext) {
    try {
      // const disciplinas = await Disciplina.all()
      const disciplinas = await Disciplina.query()
            .preload('curso').preload('alunos')

      return response.status(200).json({
        message: 'OK',
        data: disciplinas,
      })
    } catch (error) {
      return response.status(500).json({
        message: 'ERROR',
      })
    }
  }

  /**
   * Display form to create a new record
   */
  async create({}: HttpContext) {}

  /**
```

```
 * Handle form submission for the create action
 */
async store({ request, response }: HttpContext) {
  const payload = await request.validateUsing(createDisciplina)
  try {
    const disciplina = await Disciplina.create({
      ...payload,
    })
    return response.status(201).json({
      message: 'OK',
      data: disciplina,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}

/**
 * Show individual record
 */
async show({ params, response }: HttpContext) {
  try {
    // const disciplina = await Disciplina.findOrFail(params.id)
    const disciplina = await Disciplina.query()
          .where('id', params.id).preload('curso')
          .preload('alunos').firstOrFail();

    return response.status(200).json({
      message: 'OK',
      data: disciplina,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}

/**
```

```
 * Edit individual record
 */
async edit({ params }: HttpContext) {}

/**
 * Handle form submission for the edit action
 */
async update({ params, request, response }: HttpContext) {
  const payload = await request.validateUsing(updateDisciplina)
  try {
    const disciplina = await Disciplina.findOrFail(params.id)
    await disciplina.merge({ ...payload }).save()

    return response.status(200).json({
      message: 'OK',
      data: disciplina,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}

/**
 * Delete record
 */
async destroy({ params, response }: HttpContext) {
  try {
    const disciplina = await Disciplina.findOrFail(params.id)
    await disciplina.delete()

    return response.status(200).json({
      message: 'OK',
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}
```

```
}
```

Arquivo: "/app/controllers/alunos_controller.ts"

TypeScript

```typescript
import type { HttpContext } from '@adonisjs/core/http'
import { errors } from '@adonisjs/core'
import Aluno from '#models/aluno'
import { createAluno, updateAluno } from '#validators/aluno'

export default class AlunosController {
  /**
   * Display a list of resource
   */
  async index({ response }: HttpContext) {
    try {
      // const alunos = await Aluno.all()
      const alunos = await Aluno.query().preload('curso')
            .preload('disciplinas')

      return response.status(200).json({
        message: 'OK',
        data: alunos,
      })
    } catch (error) {
      return response.status(500).json({
        message: 'ERROR',
      })
    }
  }

  /**
   * Display form to create a new record
   */
  async create({}: HttpContext) {}

  /**
   * Handle form submission for the create action
   */
  async store({ request, response }: HttpContext) {
```

```
  const payload = await request.validateUsing(createAluno)
  try {
    const aluno = await Aluno.create({
      ...payload,
    })
    return response.status(201).json({
      message: 'OK',
      data: aluno,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}

/**
 * Show individual record
 */
async show({ params, response }: HttpContext) {
  try {
    // const aluno = await Aluno.findOrFail(params.id)
    const aluno = await Aluno.query()
          .where('id', params.id).preload('curso')
          .preload('disciplinas').firstOrFail();

    return response.status(200).json({
      message: 'OK',
      data: aluno,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}

/**
 * Edit individual record
 */
async edit({ params }: HttpContext) {}
```

```
  /**
   * Handle form submission for the edit action
   */
  async update({ params, request, response }: HttpContext) {
    const payload = await request.validateUsing(updateAluno)
    try {
      const aluno = await Aluno.findOrFail(params.id)
      await aluno.merge({ ...payload }).save()

      return response.status(200).json({
        message: 'OK',
        data: aluno,
      })
    } catch (error) {
      return response.status(500).json({
        message: 'ERROR',
      })
    }
  }

  /**
   * Delete record
   */
  async destroy({ params, response }: HttpContext) {
    try {
      const aluno = await Aluno.findOrFail(params.id)
      await aluno.delete()

      return response.status(200).json({
        message: 'OK',
      })
    } catch (error) {
      return response.status(500).json({
        message: 'ERROR',
      })
    }
  }
}
```

Arquivo: “/app/controllers/matriculas_controller.ts”

TypeScript

```typescript
import { Disciplina } from '#models/disciplina';
import type { HttpContext } from '@adonisjs/core/http'
import Matricula from '#models/matricula'
import { createMatricula } from '#validators/matricula'

export default class MatriculasController {
  /**
   * Display a list of resource
   */
  async index({ response }: HttpContext) {
    try {
      // const matriculas = await Matricula.all()
      const matriculas = await Matricula.query()
            .preload('aluno').preload('disciplina')

      return response.status(200).json({
        message: 'OK',
        data: matriculas,
      })
    } catch (error) {
      return response.status(500).json({
        message: 'ERROR',
      })
    }
  }

  /**
   * Display form to create a new record
   */
  async create({}: HttpContext) {}

  /**
   * Handle form submission for the create action
   */
  async store({ request, response }: HttpContext) {
    const payload = await request.validateUsing(createMatricula)
    try {
      const matricula = await Matricula.create({
        ...payload,
      })
      return response.status(201).json({
```

```
      message: 'OK',
      data: matricula,
    })
  } catch (error) {
    return response.status(500).json({
      message: 'ERROR',
    })
  }
}

/**
 * Show individual record
 */
async show({ params }: HttpContext) {}

/**
 * Edit individual record
 */
async edit({ params }: HttpContext) {}

/**
 * Handle form submission for the edit action
 */
async update({ params, request }: HttpContext) {}

/**
 * Delete record
 */
async destroy({ params, response }: HttpContext) {
  try {
    const matricula = await Matricula.query()
          .where('aluno_id', params.alunoId)
          .where('disciplina_id', params.disciplinaId).firstOrFail()

    await matricula.delete()

    return response.status(200).json({
      message: 'OK',
    })
  } catch (error) {
    return response.status(500).json({
```

```
        message: 'ERROR',
      })
    }
  }
}
```

---

8. Definindo as Rotas

Arquivo: "/start/routes.ts"

TypeScript

```
router.resource('cursos', '#controllers/cursos_controller')
router.resource('disciplinas', '#controllers/disciplinas_controller')
router.resource('alunos', '#controllers/alunos_controller')
// Matrículas
router.get('matriculas', '#controllers/matriculas_controller.index')
router.post('matriculas', '#controllers/matriculas_controller.store')
router.delete('matriculas/:alunoId/:disciplinaId',
'#controllers/matriculas_controller.destroy')
```

---

9. Executando e Testando a Aplicação

Comando
(no terminal de comando)

Shell

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

CURSOS
(Create - *no terminal de comando)*

```shell
curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"nome": "Técnico em Mecânica", "duracao": 4}' \
  http://localhost:3333/cursos
```

(Read - *no terminal de comando)*

```shell
curl http://localhost:3333/cursos
curl http://localhost:3333/cursos/1
```

(Update - *no terminal de comando)*

```shell
curl -X PUT http://localhost:3333/cursos/1 \
  -H "Content-Type: application/json" \
  -d '{"nome": "Técnico em Meio Ambiente", "duracao": 4}'
```

(Delete - *no terminal de comando)*

```shell
curl -X DELETE http://localhost:3333/cursos/3
```

DISCIPLINAS

(Create - *no terminal de comando)*

```shell
curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"nome": "Química I", "carga": 2, "curso_id": 1}' \
  http://localhost:3333/disciplinas
```

(Read - *no terminal de comando)*

```shell
curl http://localhost:3333/disciplinas
```

```
curl http://localhost:3333/disciplinas/1
```

(Update - *no terminal de comando)*

```
Shell
curl -X PUT http://localhost:3333/disciplinas/6 \
  -H "Content-Type: application/json" \
  -d '{"nome": "Artes II", "carga": 3, "curso_id": 1}'

curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"nome": "Matemática Computacional", "carga": 3, "curso_id": 2}' \
  http://localhost:3333/disciplinas
```

(Delete - *no terminal de comando)*

```
Shell
curl -X DELETE http://localhost:3333/disciplinas/6
```

ALUNOS
(Create - *no terminal de comando)*

```
Shell
curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"nome": "Amarildo dos Santos", "curso_id": 1}' \
  http://localhost:3333/alunos

curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"nome": "Bianca Oiiveira", "curso_id": 2}' \
  http://localhost:3333/alunos
```

(Read - *no terminal de comando)*

```shell
curl http://localhost:3333/alunos
curl http://localhost:3333/alunos/1
```

(Update - *no terminal de comando)*

```shell
curl -X PUT http://localhost:3333/alunos/6 \
  -H "Content-Type: application/json" \
  -d '{"nome": "Mauro Rodrigues", "curso_id": 1}'
```

(Delete - *no terminal de comando)*

```shell
curl -X DELETE http://localhost:3333/alunos/6
```

MATRÍCULAS

(Create - *no terminal de comando)*

```shell
curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"disciplina_id": 1, "aluno_id": 1}' \
  http://localhost:3333/matriculas

curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"disciplina_id": 3, "aluno_id": 2}' \
  http://localhost:3333/matriculas
```

(Read - *no terminal de comando)*

```shell
curl http://localhost:3333/matriculas
```

(Delete - *no terminal de comando)*

Shell

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