CURSO DE ADONISJS 5 [2h]

Youtube > Conejos programadores

1. Introducción a AdonisJS

Adonis es un framework que funciona sobre Node.js, basado en el modelo vista controlador

Permite construir APIs y aplicaciones web

REQUISTOS

- NodeJS
- Git (opcional)

CREANDO NUEVO PROYECTO

```
npm init adonis-ts-app@latest hello-world
```

AdonisJS dara 3 opciones:

Web: para crear app web que renderizan vistas en el servidor, viene con EDGE (motor de plantillas) e incluye una estructura de creación de rutas y controladores que se conectan con las vistas

API: Si necesitas una API REST que será consumida por otro cliente. Enfocado en solicitudes/respuestas JSON

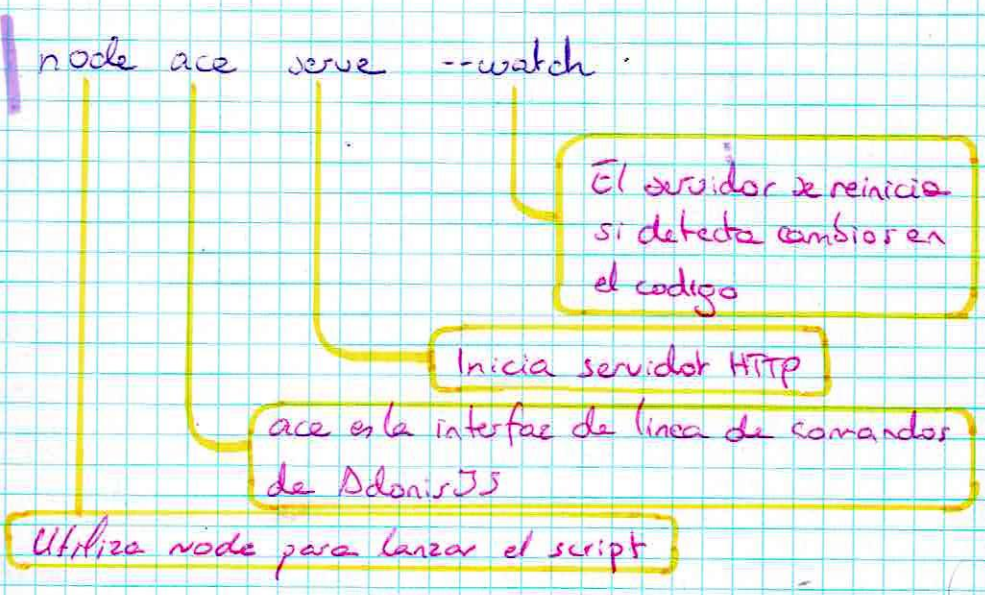
Slim: Lo mínimo para una API o logica back end.

A continuacion pedirá el nombre del proyecto

Tambien preguntará si usaremos ESLint, una herramienta de correccion de codigo para mantenerlo limpio, consistente y con menos errores comunes

Por ultimo preguntaré si usamos prettier, una herramienta que se enfoca en la consistencia del estilo del codigo (indentacion, tipo de comillas, uso de ;, alineado de parentesis y llaves

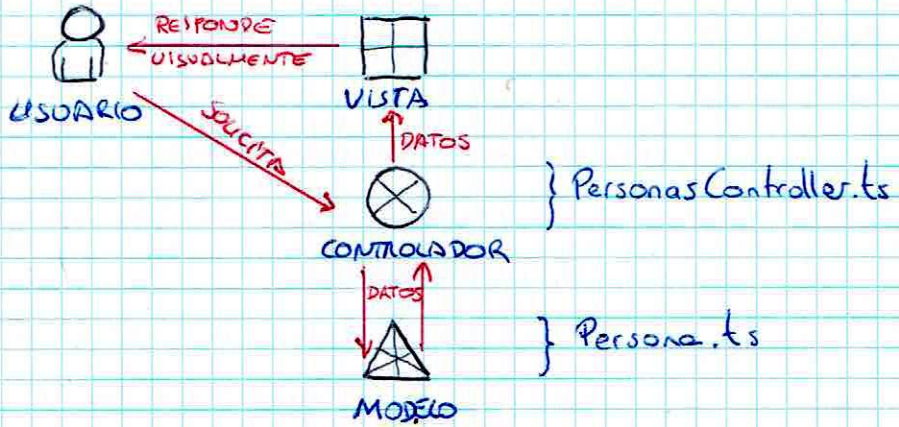
Adonis nos informa de que el proyecto se ha creado y de que podemos crear un servidor local para pruebas con este comando:



TIPOS PETICIONES HTTP

↓ GET Recibimos respuesta
↑ POST Enviamos info a nuestro servidor
↑ PUT Modificamos info del servidor
↑ DELETE Borramos info del servidor

MODELO VISTA-MODELO-CONTROLADOR



INICIAR PROYECTO

```
#instala adonis
npm i @adonisjs/lucid
#configura
node ace configure @adonis/lucid
    #tipo de BD
    #terminal o navegador
```

A continuación crearemos la BD, y en .env ajustamos las variables de conexión

Creamos un nuevo archivo de migración personas

```
node ace make:migration personas
```

Este archivo creará en el futuro una tabla llamada personas, agregamos las ~~campos~~ columnas que queramos

```
this.schema.createTable(this.tableName, ...
    table.increments('id').primary()
    ~~table.increments~~
    table.string('nombre', 40).notNullable()
    table.bigInteger('edad').nullable()
    ...
```

(S) Migration script que describe el esquema de la BD

(S) Model Permite trabajar con los datos de la BD como si fueran objetos

Creamos nuevo archivo de modelo persona

```
node ace make:model persona
```

Agregamos al archivo de modelo generado el contenido que hemos agregado al esquema (migration)

```
@column({ isPrimary: true })
public id: number

@column()
public nombre: string

@column()
public edad: number

@column()
...

@column.dateTime({ autoCreate: true})
public ~~description~~ createdAt: DateTime
```

Controlador: maneja la logica de solicitudes y respuestas HTTP y es intermediario entre las vistas, modelos y solicitudes. Contiene funciones.

Creamos nuestro controlador:

Prepara las peticiones HTTP

```
node ace make:controller Persona -r ....
```

Editamos PersonasController con las funciones que necesitemos:

```
public async index ({ response }: HttpContextContract) {
  try {
    const persona = await Persona.all()
    response.ok(persona)
  } catch (e) {
    response.badRequest()
  }
}
...
```

/App/controllers/http/PersonasController.ts
/App/models/Persona.ts
/App/migrations/xxxx_personas.ts

RUTAS (App/start/routes.ts)

Las rutas HTTP consigue que una URL especifica desencadene una función del servidor. Es una combinación de METODO HTTP y URL

Pueden contener parametros dinamicos con el prefijo :

```
Route.get('/posts/:id', 'PostController.show')
```

Tambien pueden tener Middleware; funciones que se ejecutan antes de la acción del controlador

```
Route.get('/profile', 'UserController.profile').middleware('auth')
```

Route.resource evita tener que indicar cada una de las rutas generadas automaticamente con -r

```
Route.resource('persona', 'personasController').apiOnly()
```

```
http://localhost:3333/persona
```

Tras esto se requiere:

```
node ace migration:run
```

PODEMOS USAR POSTMAN PARA GENERAR METODOS HTTP

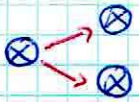
VALIDAR LA INFO. RECIBIDA

desde la migración o el modelo podremos ver que tipo de dato recibiremos para cada campo

```
public async store({request, response}: HttpContextContract){
    const personaSchema = schema.create({
        nombre: schema.string([
            rules.minLength(3),
            rules.maxLength(90)
        ]),
        edad: schema.~~bigint~~ number.nullableAndOptional(),
        descripcion: schema.string()
    })

    //Validamos la info recibida con el esquema
    const payload = request.validate({schema: personaSchema})

    //guardamos
    ~~const~~ const data = await Persona.create(await payload)

    response.ok({
        msg: "se inserto en la BD",
        data: ~~payload~~ data
    })
}
```

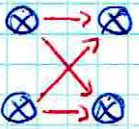
RELACIONES

UNO a UNO
persona - pasaporte



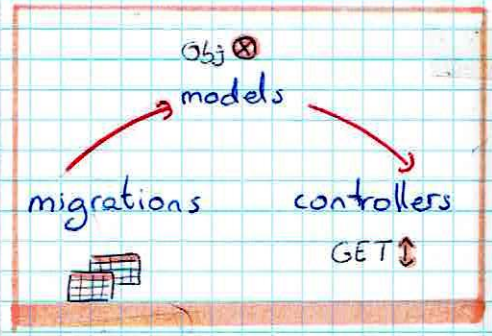
UNO a muchos
persona - dirección

muchos a UNO
uno a muchos



muchos a muchos
persona - grupo_persona

En el modelo Persona.ts indicamos el tipo de relación entre campos




```sql
CREATE TABLE personas (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nombre VARCHAR(100),
  email VARCHAR(100)
);

CREATE TABLE pasaportes (
  id INT PRIMARY KEY AUTO_INCREMENT,
  numero_pasaporte VARCHAR(50),
  persona_id INT,
  FOREIGN KEY (persona_id) REFERENCES personas(id)
);

CREATE TABLE direcciones (
  id INT PRIMARY KEY AUTO_INCREMENT,
  direccion VARCHAR(255),
  persona_id INT,
  FOREIGN KEY (persona_id) REFERENCES personas(id)
);

CREATE TABLE grupos (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nombre VARCHAR(100)
);

CREATE TABLE grupo_persona (
  persona_id INT,
  grupo_id INT,
  FOREIGN KEY (persona_id) REFERENCES personas(id),
  FOREIGN KEY (grupo_id) REFERENCES grupos(id),
  PRIMARY KEY (persona_id, grupo_id)
);
```

```php
// UNO A UNO
// Persona.js
class Persona extends Model {
  pasaporte() {
    return this.hasOne('App/Models/Pasaporte');
  }
}

// Pasaporte.js
class Pasaporte extends Model {
  persona() {
    return this.belongsTo('App/Models/Persona');
  }
}

// UNO A MUCHOS O MUCHOS A UNO
// Persona.js
class Persona extends Model {
  direcciones() {
    return this.hasMany('App/Models/Direccion');
  }
}

// Direccion.js
class Direccion extends Model {
  persona() {
    return this.belongsTo('App/Models/Persona');
  }
}

// MUCHOS A MUCHOS
// Persona.js
class Persona extends Model {
  grupos() {
    return this.belongsToMany('App/Models/Grupo');
  }
}

// Grupo.js
class Grupo extends Model {
  personas() {
    return this.belongsToMany('App/Models/Persona');
  }
}
```