# **joi**

## Npm

<https://www.npmjs.com/package/joi>

## Api y documentacion

https://joi.dev/

## Instalación

**npm i express-joi-validation joi –save**

### Introducción

**joi** le permite describir sus datos usando un lenguaje simple, intuitivo y legible.

|  |
| --- |
| const Joi = require('joi');  const schema = Joi.object({  username: Joi.string()  .alphanum()  .min(3)  .max(30)  .required(),  password: Joi.string()  .pattern(new RegExp('^[a-zA-Z0-9]{3,30}$')),  repeat\_password: Joi.ref('password'),  access\_token: [  Joi.string(),  Joi.number()  ],  birth\_year: Joi.number()  .integer()  .min(1900)  .max(2013),  email: Joi.string()  .email({ minDomainSegments: 2, tlds: { allow: ['com', 'net'] } })  })  .with('username', 'birth\_year')  .xor('password', 'access\_token')  .with('password', 'repeat\_password');  schema.validate({ username: 'abc', birth\_year: 1994 });  // -> { value: { username: 'abc', birth\_year: 1994 } }  schema.validate({});  // -> { value: {}, error: '"username" is required' }  // Also -  try {  const value = await schema.validateAsync({ username: 'abc', birth\_year: 1994 });  }  catch (err) { } |

## Uso general

El uso es un proceso de dos pasos:

Primero, se construye un esquema utilizando los tipos y restricciones proporcionados:

const schema = Joi.object({

a: Joi.string()

});

Tenga en cuenta que los objetos de esquema **joi** son inmutables, lo que significa que cada regla adicional agregada (por ejemplo .min(5), ) devolverá un nuevo objeto de esquema.

En segundo lugar, el valor se valida con el esquema definido:

const { error, value } = schema.validate({ a: 'a string' });

Si la entrada es válida, entonces errorserá undefined. Si la entrada no es válida, errorse le asigna un [ValidationError](https://github.com/sideway/joi/blob/master/API.md" \l "validationerror)objeto que proporciona más información.

El esquema puede ser un objeto simple de JavaScript donde a cada tecla se le asigna un tipo **joi** , o puede ser un tipo **joi** directamente:

const schema = Joi.string().min(10);

Si el esquema es un tipo **joi** , schema.validate(value)se puede llamar directamente en el tipo. Al pasar un objeto de esquema que no es de tipo, el módulo lo convierte internamente en un tipo de objeto () equivalente a:

const schema = Joi.object().keys({

a: Joi.string()

});

Al validar un esquema:

* Los valores (o claves en el caso de los objetos) son opcionales por defecto.

Joi.string().validate(undefined); // validates fine

Para rechazar este comportamiento, puede configurar el esquema como required()o configurarlo presenceal "required"pasar options:

Joi.string().required().validate(undefined);

// or

Joi.string().validate(undefined, /\* options \*/ { presence: "required" });

* Las cadenas están codificadas en utf-8 de forma predeterminada.
* Las reglas se definen de forma aditiva y se evalúan en orden, primero las reglas inclusivas, luego las reglas exclusivas.

**Implementación de esquemas de validación**

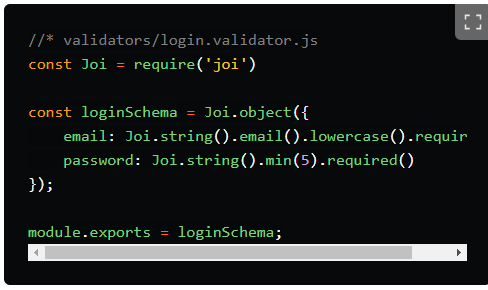
Comencemos con el esquema de inicio de sesión.

Antes del proceso de inicio de sesión, validaremos el correo electrónico y la contraseña del usuario. El cuerpo de la solicitud será así;



* emailel campo debe ser una cadena y un correo electrónico válido
* passwordel campo tiene que ser una cadena y una longitud mínima de 4

También estos dos campos son obligatorios.



Continuemos con el esquema de registro.

Para registrar un usuario, necesitamos , email, usernamee info . Todos estos campos son obligatorios. Y aquí está el esquema. password name surname



Un usuario puede compartir una publicación. Una publicación tiene title, contenty tagscampos. Aquí hay un cuerpo de solicitud de ejemplo:



* Como ve, tagsel campo es una matriz de cadenas.
* tagsla longitud mínima es 2 y la longitud máxima es 4

**Exportar todos los esquemas como módulo único**

Requerir todos los validadores y exportarlos como un objeto



**Validatorsoftware intermedio**

Ahora vamos a crear un middleware llamado Validatory actúa como método de fábrica.

* Toma el nombre del validador como parámetro.
* Si el validador dado no existe, arroja un error.
* Si ocurre un error de validación, el controlador de errores regresaHTTP 422 Unprocessable Entity



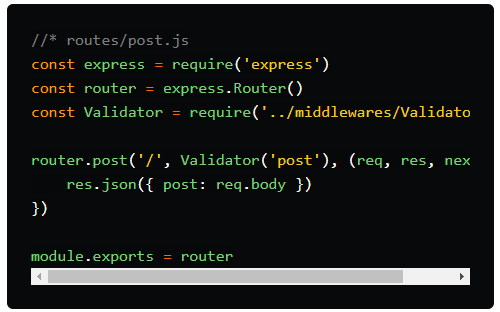
## ¿Cómo utilizar?

Aquí están los authpuntos finales falsos para usar

1. [POST] /auth/login: llamar Validator('login')antes de la devolución de llamada de respuesta
2. [POST] /auth/register: llamar Validator('register')antes de la devolución de llamada de respuesta



Y aquí está el punto final falso postpara usar Validator middleware.



Finalmente, creamos el servidor HTTP.

