Schema Validator Library Comparisons

Here, we compare the main features of 2 server side schema validator libraries - **Mongoose-Validator** and **Joi**.

Mongoose Validator:

- It validates the mongoose schema and returns mongoose-style validation objects. [1]
- It uses validators from **Validator.js**. [1]
- It validates only string-based data since the validators in **Validator.js** support only strings. [1]
- It supports sending error messages and HTTP Error Codes in its validators. [1]
- It supports regex through the **validator.matches** method. [1]

Joi:

It is a validation library that allows us to build schemas to validate JavaScript objects.

- First a Joi schema, **schema** is created using **Joi.object**. Then, some data object that follows the schema structure is validated using **schema.validate**(data). An error is returned if the validation fails. [2]
- A schema can also be a JavaScript object with each of the keys being Joi types. When such an object is passed to **schema.validate**, then it converts the object directly to a **Joi** type with keys. [2]
- **Joi** schema objects are immutable. This means that any new rule or key added to a **Joi** object will return a new schema. [2]
- It supports Strings, Numbers, Boolean, etc. and also validators for email, regex patterns, ranges for numbers. [3]
- It supports **Validation against Data References**. Eg - If we have 2 fields, **wealth** and **savings** in a schema, then **savings** can never be > **wealth**. This condition can be implemented using the **ref()** method. [3]
- It supports **Conditional Validation**. For example, fields related to credit cards will be shown only when the **creditCardIsTrue** field is **true**.
- **Joi** also supports Conditional Relationships between 2 or more fields. Eg - **with**(A, B) [Both A and B fields must be present] / **xor**(A, B) [Either A or B can be present].
- One more thing that **Joi** can do is that it can be used to validate the request body of an HTTP POST request. [6]

Mongoose schemas can be validated using the **joiValidate** method. This is done as follows:

- First a mongoose schema (Eg - **userSchema**) is created using **mongoose.schema**. [4]
- Then the **userSchema.methods.joiValidate** property is used to create a function in which the schema is recreated as a **Joi** object. It is then validated using the **Joi.validate** method. [4]

One disadvantage of using **Joi** to validate a mongoose schema is that the schema needs to be written twice, once as a **mongoose schema** and another time as a **Joi object**. [4]

This drawback can be overcome using the **Joigoose** npm package. [5]

Joigoose:

- The **Joigoose** npm package allows a joi schema to be converted to a mongoose style schema. [5]

REFERENCES

[1] https://www.npmjs.com/package/mongoose-validator
[2] https://joi.dev/api/?v=17.13.3
[3]https://amandeepkochhar.medium.com/what-ive-learned-validating-with-joi-object-schema-validation-7a90847f9ed4
[4] https://gist.github.com/stongo/6359042
[5] https://www.npmjs.com/package/joigoose
[6]https://medium.com/@techsuneel99/validate-incoming-requests-in-node-js-like-a-pro-95fbdff4bc07