# Schema Validator Library Comparisons - Updated

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]

**Note:** Conditional validation and conditional relations between fields aren't possible in **joigoose**.

**AJV - Another JSON Validator**

AJV is a validation library that allows data validation and sanitization logic to be written as a JSON Schema. It can compile code into JSON Schema very quickly. It is currently the fastest validator. It uses Just In Time compilation. Its latest version, version 7, supports secure code generation. [7]

**Basic Data Validation**:

AJV compiles schemas to functions and caches them in all cases (using the schema itself as a key in a Map), so that the next time the same schema object is used it won't be compiled again. [7]

**Using Standalone Validation**

Pre-Compiling schemas has some advantages. They are:

- faster startup times
- lower memory footprint/bundle size
- compatible with strict content security policies
- almost no risk to compile schema more than once
- better for short-lived environments

The simplest and best way to compile schemas is to do so during the application start. It should be done outside the code that handles requests. It is done using the **ajv.compile** method. [8]

**Using AJV Instance Cache:**

This approach allows us to have all the compiled validators available everywhere in the application from a single import. [8] We create a single module for validation, load all the schemas there and then add it to an AJV instance (done using the **addSchema** method). Then this AJV module can be imported and all the schemas can be accessed from there. The schemas can be accessed using the **ajv.getSchema** method. [8]

**Asynchronous Schema Loading:**
An async function can be passed to the **compileAsync** function that allows schemas to be downloaded as they are compiled. Additionally, it is possible to maintain a list of cached schemas independent from AJV in our application. [8]

**Format Validators:**
A variety of format validators such as those for date, email, regex, ipv4-address, ipv6-address and URL are there. These are stored in the **add-formats** package. They can be added to the schema using the **addFormat()** method after importing the said package. [9]

**Interfacing With Mongoose:**
It is possible to have an AJV schema validated using a mongoose schema by using the **mongoose-ajv-plugin**. This is done by creating an equivalent AJV schema for every corresponding mongoose schema and validating the mongoose schema using the AJV schema. [10]

**Advantages**:
- High Speed
- High Efficiency
- Secure Code Generation
- Adherence to JSON Type Definition Specifications.
- Many different types of JSON Schemas are supported.

**Disadvantages:**
- A lot less intuitive than Joi.
- Cannot define conditional relations between different fields.
- Cannot perform conditional validation.
- The **mongoose-ajv-plugin** on npm is much less popular and doesn't appear to be well maintained [Only 7 stars on Github and last update 6 years ago]. [10]

**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
[7] https://ajv.js.org/guide/why-ajv.html
[8] https://ajv.js.org/guide/managing-schemas.html
[9] https://ajv.js.org/guide/formats.html
[10] https://www.npmjs.com/package/mongoose-ajv-plugin