


# Node.js API Validation Using Joi - Adding Certainty to a Loosely-Typed World

xo group/ *the knot* the nest the **BUMP**  gigmasters

Wes Tyler

# The Plan

- Brief examples of why data validation is a good idea
- Brief introduction to Joi
- Using Joi in Express to mitigate first 2 examples
- Using Joi for data consistency in document stores
- Managing Joi schema and testing with Felicity

# Why Data Validation?

# Why Data Validation?

```
$ curl example.com/users -H "Content-Type: application/json" -X POST -d '{"id": "" OR 1=1--"}'
```

# Why Data Validation?

```
$ curl example.com/users -H "Content-Type: application/json" -X POST -d '{"id": "" OR 1=1--"}'
```

```
1 | SELECT * FROM users WHERE id = '' OR 1=1 -- AND NOT type = 'admin'
```

# Why Data Validation?

```
return dbAccessLayer.get(/**/) →  
    .then((userResult) => { →  
        delete userResult.password; →  
    } →  
    return reply(userResult); →  
}); →
```

# Why Data Validation?

```
return dbAccessLayer.get(/**/)-  
    .then((userResult) => {  
        delete userResult.password;  
    })  
    .return reply(userResult);  
});
```

# Why Data Validation?

```
const saveRecord = function (unMappedData) {  
  ... const mappedData = mapper(unMappedData);  
  ...  
  ... return dbAccessLayer.save(mappedData);  
}
```



# Enter Joi



# Enter Joi

Object schema description language and validator for JavaScript objects.



# Enter Joi

Object schema description language and validator for JavaScript objects.

**3,189,895** downloads in the last month



# Enter Joi

Object schema description language and validator for JavaScript objects.

**3,189,895** downloads in the last month

**Dependents (2596)**



# Enter Joi

Object schema description language and validator for JavaScript objects.

**3,189,895** downloads in the last month

**Dependents (2596)**



**marsup** published a week ago

---

**10.4.1** is the latest of 156 releases





Object schema description language and validator for JavaScript objects.



```
Joi.string() →  
Joi.number() →  
Joi.boolean() →  
Joi.array() →  
Joi.object() →  
Joi.func() →
```



```
Joi.date() →
```

```
Joi.binary() →
```

```
Joi.any() →
```

```
Joi.alternatives().try() →
```

```
Joi.alternatives().when() →
```

```
Joi.lazy() →
```





```
Joi.string().guid()→  
Joi.number().integer()→  
Joi.boolean().truthy('yes')→  
Joi.array().items(Joi.string())→  
Joi.object().keys({})→  
Joi.func().arity()→
```



```
Joi.any().required() →  
Joi.any().optional() →  
Joi.any().allow(null) →  
Joi.any().valid(/**/) →
```

# Data Validation as a Security Step

```
$ curl example.com/users -H "Content-Type: application/json" -X POST -d '{"id": "" OR 1=1--"}'
```

# Data Validation as a Security Step

```
'use strict';  
  
const Express = require('express');  
const bodyParser = require('body-parser');  
const app = Express();  
  
app.use(bodyParser.json());  
  
app.post('/search', function (req, res) {  
  res.send(`Got a record back for ${req.body.id}`);  
});  
  
app.listen(3000, function () {  
  console.log('Example app listening on port 3000!');  
});
```

# Data Validation as a Security Step

```
'use strict';  
  
const Express = require('express');  
const bodyParser = require('body-parser');  
const app = Express();  
  
app.use(bodyParser.json());  
  
app.post('/search', function (req, res) {  
  res.send(`Got a record back for ${req.body}`);  
});  
  
app.listen(3000, function () {  
  console.log('Example app listening on port');  
});
```

The screenshot shows a web browser interface for a REST client. The top bar indicates a POST request to `localhost:3000/search`. The 'Body' tab is selected, showing a JSON payload: `{ "id": "' OR 1=1 --" }`. The 'Send' button is highlighted. Below the request, the response is shown in the 'Body' tab: `Got a record back for ' OR 1=1 --`. The status is `200 OK` and the time is `42 ms`.

```
'use strict';  
  
const Express = require('express');  
const bodyParser = require('body-parser');  
const Joi = require('joi');  
const app = Express();  
  
app.use(bodyParser.json());  
  
const searchSchema = Joi.object().keys({  
  ...id: Joi.string().guid().required()  
});  
  
app.post('/search', function (req, res) {  
  ...res.send(`Got a record back for ${req.body.id}\n`);  
});  
  
app.listen(3000, function () {  
  ...console.log('Example app listening on port 3000!')  
});
```

```
'use strict';  
  
const Express = require('express');  
const bodyParser = require('body-parser');  
const Joi = require('joi');  
const app = Express();  
  
app.use(bodyParser.json());  
  
const searchSchema = Joi.object().keys({  
  ...id: Joi.string().guid().required()  
});  
  
app.post('/search', function (req, res) {  
  ...res.send(`Got a record back for ${req.body.id}\n`);  
});  
  
app.listen(3000, function () {  
  ...console.log('Example app listening on port 3000!')  
});
```

```
const searchSchema = Joi.object().keys({  
  ...id: Joi.string().guid().required()  
});  
  
const validateSearch = function (req, res, next) {  
  ...searchSchema.validate(req.body, (err, value) => {  
    ...if (err) {  
      ...return res.status(400).send('Bad Request');  
    }  
    ...return next();  
  });  
};  
  
app.post('/search', function (req, res) {  
  ...res.send(`Got a record back for ${req.body.id}\n`);  
});
```



```
const searchSchema = Joi.object().keys({  
  ...id: Joi.string().guid().required()  
});  
  
const validateSearch = function (req, res, next) {  
  ...searchSchema.validate(req.body, (err, value) => {  
    ...if (err) {  
      ...return res.status(400).send('Bad Request');  
    }  
    ...return next();  
  });  
};  
  
app.post('/search', function (req, res) {  
  ...res.send(`Got a record back for ${req.body.id}\n`);  
});
```

```

const searchSchema = Joi.object().keys({
  ...id: Joi.string().guid().required()
});

const validateSearch = function (req, res, next) {
  ...searchSchema.validate(req.body, (err, value) => {
    ...if (err) {
      ...return res.status(400).send('Bad Request');
    }
    ...return next();
  });
};

app.post('/search', validateSearch, function (req, res) {
  ...res.send(`Got a record back for ${req.body.id}\n`);
});

```

```

const searchSchema = Joi.object({
  id: Joi.string().guid().required(),
});

const validateSearch = function (req, res, next) {
  const err = searchSchema.validate(req.body);
  if (err) {
    return res.status(400).json({
      error: err.message,
    });
  }
  return next();
};

app.post('/search', validateSearch, function (req, res) {
  res.send(`Got a record back for ${req.body.id}\n`);
});

```

POST localhost:3000/search Params Send Save

Authorization Headers (1) **Body** Pre-request Script Tests Code

form-data x-www-form-urlencoded **raw** binary JSON (application/json)

```

1 {
2   "id": "' OR 1=1 --"
3 }

```

Body Cookies Headers (6) Tests Status: 400 Bad Request Time: 43 ms

Pretty Raw Preview HTML

```

1 Bad Request

```

```
app.post('/search', validateSearch, function (req, res) {  
  ... const exampleDbRecord = {  
    ... id          : '32daa3aa-7cd6-48af-9ec8-1db4a1ca9887',  
    ... publicInfo: {  
      ... name: 'Clara Oswald'  
    ... },  
    ... password: 'imp0ssibl3!1!'  
  ... };  
  
  ... delete exampleDbRecord.pasword;  
  
  ... res.send(exampleDbRecord);  
});
```

```

app.post('/search', validateSearch,
  ... const exampleDbRecord = {
  ...   id      : '32daa3aa-7cd6-48af-9ec8-1db4a1ca9887'
  ...   publicInfo: {
  ...     name: 'Clara Oswald'
  ...   },
  ...   password: 'imp0ssibl3!1!'
  ... };
  ... delete exampleDbRecord.password;
  ...
  ... res.send(exampleDbRecord);
});

```

POST localhost:3000/search Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```

1 {
2   "id": "32daa3aa-7cd6-48af-9ec8-1db4a1ca9887"
3 }

```

Body Cookies Headers (6) Tests Status: 200 OK Time: 17 ms

Pretty Raw Preview JSON

```

1 {
2   "id": "32daa3aa-7cd6-48af-9ec8-1db4a1ca9887",
3   "publicInfo": {
4     "name": "Clara Oswald"
5   },
6   "password": "imp0ssibl3!1!"
7 }

```

```
const userOutputSchema = Joi.object({  
  ...id: Joi.string().guid(),  
  ...publicInfo: Joi.object({  
    ...name: Joi.string()  
  })  
}).options({stripUnknown: true});
```

```
app.post('/search', validateSearch, function (req, res) {  
  ...const exampleDbRecord = {  
    ...id : '32daa3aa-7cd6-48af-9ec8-1db4a1ca9887',  
    ...publicInfo: {  
      ...name: 'Clara Oswald'  
    },  
    ...password: 'imp0ssibl3!1!'  
  };  
  ...delete exampleDbRecord.pasword;  
  ...res.send(exampleDbRecord);  
});
```

```

const userOutputSchema = Joi.object({
  ...id: Joi.string().guid(),
  ...publicInfo: Joi.object({
  ...}).options({stripUnknown: true});
});

app.post('/search', validateSearch, function (req, res) {
  ...const exampleDbRecord = {
  ...      id      : '32daa3aa-7cd6-48af-9ec8-1db4a1ca9887',
  ...      publicInfo: {
  ...      password: 'imp0ssibl3!1!'
  ...    };
  ...
  ...return userOutputSchema.validate(exampleDbRecord, (err, value) => {
  ...  ...if (err) {
  ...    ...return res.status(500).send('Internal Server Error');
  ...  }
  ...  ...return res.send(value);
  ...});
});

```

```

const userOutputSchema = Joi.object({
  ...id: Joi.string().guid(),
  ...publicInfo: Joi.object({
  ...}).options({stripUnknown: true});
});

app.post('/search', validateSearch, function(req, res) {
  ...const exampleDbRecord = {
  ...id: '32daa3aa-7cd6-48af-9ec8-1db4a1ca9887',
  ...publicInfo: {
  ...password: 'imp0ssibl3!1!'
  ...};

  ...return userOutputSchema.validate(exampleDbRecord, {
  ...if (err) {
  ...return res.status(500).send(err.message);
  ...}
  ...return res.send(value);
  ...});
});

```

POST localhost:3000/search Params Send Save

Authorization Headers (1) **Body** Pre-request Script Tests [Code](#)

form-data x-www-form-urlencoded **raw** binary **JSON (application/json)**

```

1 {
2   "id": "32daa3aa-7cd6-48af-9ec8-1db4a1ca9887"
3 }

```

Body Cookies Headers (6) Tests Status: 200 OK Time: 44 ms

Pretty Raw Preview **JSON**

```

1 {
2   "id": "32daa3aa-7cd6-48af-9ec8-1db4a1ca9887",
3   "publicInfo": {
4     "name": "Clara Oswald"
5   }
6 }

```





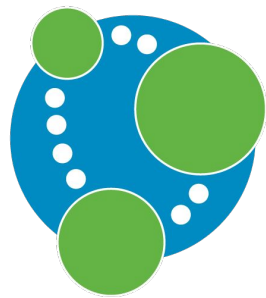
elastic



joi



PostgreSQL



neo4j



mongoDB®

# Why Data Validation?

```
const saveRecord = function (unMappedData) {  
  ... const mappedData = mapper(unMappedData);  
  ...  
  ... return dbAccessLayer.save(mappedData);  
}
```

```

const userSchema = Joi.object({
  ...id: Joi.string().guid(),
  ...name: Joi.string(),
  ...timestamp: Joi.object({
    ...create: Joi.date(),
    ...update: Joi.date()
  })
});

const saveRecord = function (unMappedData) {
  ...const mappedData = mapper(unMappedData);
  ...
  ...return userSchema.validate(mappedData, (err, value) => {
    ...if (err) {
      ...throw err;
    }
    ...return dbAccessLayer.save(value);
  });
}

```

# So What's Even Left?

- Schema management
  - Schema for request validation
  - Schema for response validation
  - Schema for database document insertion validation
- You **are** unit testing your application
  - Mock data for requests
  - Assertions against expected response payloads
  - Mock data for database documents

# Felicity

Two main functions:

# Felicity

Two main functions:

1. Object constructors

# Felicity

Two main functions:

1. Object constructors

```
const userSchema = Joi.object({  
  ... id: Joi.string().guid(),  
  ... name: Joi.string(),  
  ... timestamp: Joi.object({  
    ... create: Joi.date(),  
    ... update: Joi.date()  
  ... })  
});  
  
const User = Felicity.entityFor(userSchema);  
const user = new User(userData);  
  
user.validate((err, value) => {/**/});
```

# Felicity

Two main functions:

1. Object constructors
2. Mock data

```
const userSchema = Joi.object({  
  id: Joi.string().guid(),  
  name: Joi.string(),  
  timestamp: Joi.object({  
    create: Joi.date(),  
    update: Joi.date()  
  })  
});  
  
const User = Felicity.entityFor(userSchema);  
  
User.example();  
  
/*  
{ id: 'edc8a67f-590d-4512-b58e-f7e5b0a7f1f5',  
  name: 'cg96llq0qcwl0mg10bfhia4i',  
  timestamp:  
    { create: 2022-05-31T08:01:40.235Z,  
      update: 2026-06-15T23:41:16.443Z }  
}  
*/
```



```
// schema_hub.js
const userSchema = Joi.object({
  id: Joi.string().guid(),
  name: Joi.string(),
  isAdmin: Joi.boolean().default(false)
});

const itemSchema = Joi.object({
  id: Joi.number().positive().integer(),
  categories: Joi.array().items(Joi.string()).min(1)
});

const searchSchema = Joi.object().keys({
const userOutputSchema = Joi.object({

const user = Felicity.entityFor(userSchema);
const item = Felicity.entityFor(itemSchema);
const search = Felicity.entityFor(searchSchema);
const userOutput = Felicity.entityFor(userOutputSchema);

module.exports = {
  user,
  item,
  search,
  userOutput
};
```

```
// test.mocha.js
const schemaHub = require('./schema_hub');

const mockUsers = [];
const mockItems = [];

for (let i = 0; i < 10; ++i) {
  mockUsers.push(schemaHub.user.example());
  mockItems.push(schemaHub.item.example());
}

server.inject('/search', schemaHub.search.example())
  .then((response) => {
    schemaHub.searchResponse.validate(response.payload, (err, result) => {
      result.should.have.property('success').and.equal(true);
      done();
    });
  });
```

```

app.use(bodyParser.json());

const validateSearch = function (req, res, next) {
  SchemaHub.search.validate(req.body, (err, value) => {
    if (err) {
      return res.status(400).send('Bad Request');
    }
    return next();
  });
};

app.post('/search', validateSearch, function (req, res) {
  const exampleDbRecord = {
    id: '32daa3aa-7cd6-48af-9ec8-1db4a1ca9887',
    publicInfo: {
      name: 'Clara Oswald'
    },
    password: 'imp0ssibl3!1!'
  };

  return SchemaHub.userOutput.validate(exampleDbRecord, (err, result) => {
    if (err) {
      return res.status(500).send('Internal Server Error');
    }
    return res.send(result.value);
  });
});

```

```
const User = schemaHub.user;

const saveRecord = function (data) {
  const user = new User(data);

  return user.validate((err, result) => {
    if (err) {
      throw err;
    }
    return dbAccessLayer.save(result.value);
  })
};
```

# tl;dr

- Request and Response validation is easy but important
  - Joi makes it easier
- Schema management yields consistent and versioned data contracts
  - Felicity makes it better

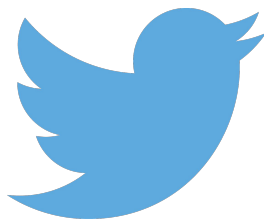
# Thank you!



 [hapijs](#) / [joi](#)

 [xogroup](#) / [felicity](#)

 [WesTyler](#) / [talks](#)



[@westyler1](#)

[@XOgroupTech](#)