Validation with JSONSchema

Download Demo Code

Goals

- Understand how and why to validate APIs
- Validate our JSON APIs using jsonschema

Data Validation with Schemas

Server-side Data Validation

A server lacking adequate validation can result in:

- corrupt or incomplete data *(can cause errors later)*
- crashing or locking up the server
- displaying unhelpful errors to the frontend/API users
- extra server or database operations due to unsuccessfully trying to process bad data

Why JSON Schema?

There are three main reasons for using a schema validation system:

- So user data can fail fast, before bad data gets to your db.
- To reduce amount of code for processing and validating data.
- To get a validation system that is easy to setup and maintain.

Rolling Your Own Validation Doesn't Always Scale

Let's assume you have a /books endpoint, and a JSON payload to add a new book looks like this:

```
{
  "book": {
    "isbn": "0691161518",
    "amazon-url": "http://a.co/eobPtX2",
    "author": "Matthew Lane",
    "language": "english",
    "pages": 264,
    "publisher": "Princeton University Press",
    "title": "Power-Up: Unlocking Hidden Math in Video Games",
    "year": 2017
  }
}
```

Your request

Your /books POST request handler might look like this:

demo/routes/books.js

```
router.post("/", function(req, res, next) {
  const bookData = req.body.book;

  if (!bookData) {
    // pass a 400 error to the error-handler
    let error = new ExpressError("Book data is required", 400);
    return next(error);
  }

  // (not implemented) insert book into database here

  return res.json(bookData);
});
```

Light validation here —checking if ***req.body.book*** is null/undefined

More validation

- What if you want title and author to be required fields?
- What if users send invalid Amazon URLs or ISBNs?

If we want to roll our own validation this way, every request handler is just going to have tons of conditional logic checking for edge cases.

JSON Schema Basics

- JSON Schema is a standard specification for describing JSON in a human and machine-readable format.
- We can auto-generate validation from sample data
- We can do this at JSONschema.net

JSONschema.net

- To get started, paste the JSON in left box and submit
- Resulting JSON schema from our input is in HTML tab
- You can edit directly on the website
- If we'd like, we can manually edit the schema.
- We can now copy and paste this into into any ***.json*** file
- We will be importing this ***.json*** file as our validator

Using the JSONSchema NPM Package in Express

We'll be using the jsonschema npm package (links: npm and github).

- Install the package using `npm install jsonschema`
- You import the validator.
- You supply the validator with a schema.
- You pass instances of user input to the validator.
- The validator checks if user input is valid against schema.
- If invalid, you respond with errors. Otherwise continue.

Note: Advanced validators

When you ask JSONSchema to infer a schema for you, it makes guesses as to the types of basic validation. But there are more advanced types of validation you can add directly from the interface.

For example, you can set a min or max length on strings, or verify that strings adhere to certain formats via the *format* key (e.g. emails or URLs). You can click on a property to see what sorts of validations are available.

The steps

demo/routes/books.js

```
const jsonschema = require("jsonschema");
const bookSchema = require("../schemas/bookSchema.json");

router.post("/with-validation", function(req, res, next) {
  const result = jsonschema.validate(req.body, bookSchema);

  if (!result.valid) {
    // pass validation errors to error handler
    //  (the "stack" key is generally the most useful)
    let listOfErrors = result.errors.map(error => error.stack);
    let error = new ExpressError(listOfErrors, 400);
    return next(error);
  }

  // at this point in code, we know we have a valid payload
  const { book } = req.body;
  return res.json(book);
});
```

Note: Error handling

In our current example, the error we send back to the client if the schema validation fails is just a single string with all of the individual failure messages concatenated together.

This may not be the most helpful way to send back errors to the client. If you're curious, you can spend some time in the next lab thinking about other ways to send multiple error messages back when the schema validation fails.

Things to be aware of!

- Validation can be quite strict, so take care when setting up schema
- As you add new fields/columns, make sure you update schema
- You may want different schemas for updating/creating

We're just scratching the surface

- You can create extremely robust validation systems with JSONSchema
- You can customize almost everything, from type to error message

Your Turn