JSON Information Architecture

Richard Warburton

@richardwarburto www.monotonic.co.uk

What do we mean by Information Architecture?

Implicit vs Explicit Schemas

Schema

A definition of the structure and content of the JSON documents you are producing and consuming

JSON Schemas

Implicit

Producer and Consumer assume the structure of the message

Explicit

Schema is documented and ideally enforced with class generation

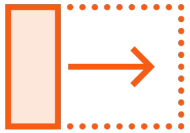
Implicit Schemas

Poor readability
Reliant on two sets of code – no overview

Unassigned responsibility
Which code is correct – consumer or producer?

Hard to maintain
Replicate any changes in two places

Explicit Schemas



Can generate API classes from common definition

JSON Schema – http://jsonschema.org

Swagger – https://swagger.io

Versioning

Evolution + Independent Deployment

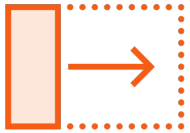
Service

Sends and receives JSON

Service

Sends and receives JSON

Versioning Advantages



Debuggability – “What is this weird JSON payload?”, “Oh it’s an old version”

Compatibility – can check what version an endpoint supports and act accordingly

Evolution – can check version when processing a request or response

How to Version

Independent Deployment

Only version when the users of the API can be deployed independently

Schema Driven

Bump the version when the schema changes

Minimizing the Impact of Change

Minimise Impact of Change

Only extract needed information from a JSON document

Less vulnerable to Schema changes

Binding API doesn't error when fields are missing

Demo

Small Changes demo

Evolve the Bank Loan Schema

Evaluate difficulty of coping with this from different APIs

Representing API Errors

JSON + HTTP

Web Browser

Sends and receives JSON

Web Service

Sends and receives JSON

Error Representation Choices

Use HTTP

Return a standard HTTP error status code. (Eg: 500, 404, etc.)

Use JSON

Return a JSON document representing the error condition

Use HTTP status codes for errors, otherwise you break other tools

Representing Data Types

```
{"year":2023, "month":"JANUARY", "chronology":
{"id":"ISO","calendarType":"iso8601"}, "dayOfMonth":15,
"dayOfWeek":"SUNDAY", "dayOfYear":15, "era":"CE",
"monthValue":1, "leapYear":false}
```

```
"2023-01-15"
```


Readability

JSON should be human readable, not just text

Make debugging easier by being able to read your encoded JSON documents

Consistency

Use the same representation of a datatype throughout your API

```
“numberField”: 123
```

```
“numberField”: “123”
```

Primitives

JSON has built-in representations for things – use them, don’t NIH

Portability

```
// Don’t expose platform specific representations

// JSON may be consumer or produced from other languages


{"year":2023, "month":"JANUARY", "chronology":
{"id":"ISO","calendarType":"iso8601"}, "dayOfMonth":15,
"dayOfWeek":"SUNDAY", "dayOfYear":15, "era":"CE",
"monthValue":1, "leapYear":false}


"2023-01-15"
```

Size

```
// Minimize JSON payload size (but not at the expense of
anything else)

// Likely to improve system performance
```

```
{"year":2023, "month":"JANUARY", "chronology":
{"id":"ISO","calendarType":"iso8601"}, "dayOfMonth":15,
"dayOfWeek":"SUNDAY", "dayOfYear":15, "era":"CE",
"monthValue":1, "leapYear":false}
```

```
"2023-01-15"
```

Conclusion

Summary

JSON isn’t just about learning an API

Learn from other people’s mistakes – use these best practices

Coming up next: Integrating JSON