OpenAPI

APIs of interest for web apps

- Purpose: information hiding - neither server nor client should know details of implementation on other side
- Unbreakable contract: should not change - standardized
    - Versions may update with breaking changes

Documentation?

- Highly subjective - some programmers better than others at documenting
- Incomplete - what one programmer finds important may not match others
- Outdated
- Human language specific

Description Files

- Machine readable - has very specific structure
- Enable automated processing:
  - boilerplate code
  - mock servers
- Example: assembly language is a version of the programming language of computers that is both machine and human readable
  - Structured so it can be compiled
- Versus: English language specification which needs someone to write code

OpenAPI Specification (OAS)

- **Vendor-neutral** format for **HTTP-based remote API** specification
- Does not aim to describe all possible APIs
- Efficiently describe the common use cases
- Originally developed as Swagger - evolved from Swagger 2.0

Current version: OAS3 - v3.1.0 as of Aug 2021

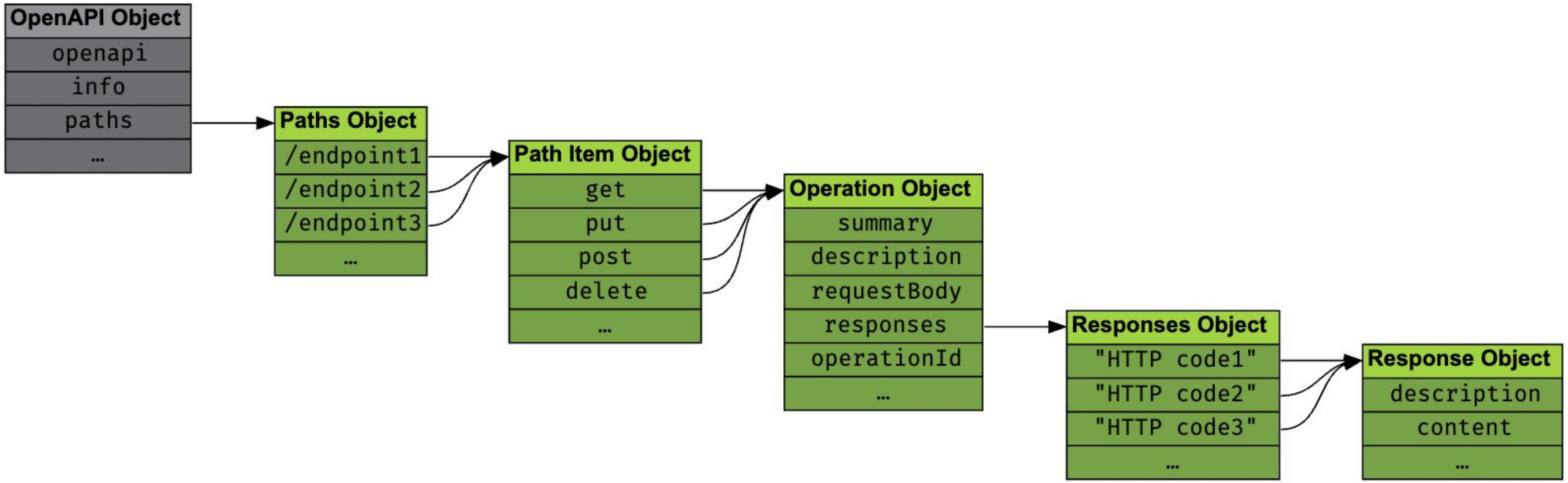
Concepts

- Describe in YAML (or possibly JSON)
- Specific structure to indicate overall information, paths, schemas etc.

eg:

```yaml
openapi: 3.1.0
info:
  title: A minimal OpenAPI document
  version: 0.0.1
paths: {} # No endpoints defined
```

Endpoints List



From: https://oai.github.io/Documentation/specification-paths.html

Paths

```yaml
openapi: 3.1.0
info:
  title: Tic Tac Toe
  description: |
    This API allows writing down marks on a Tic Tac Toe board
    and requesting the state of the board or of individual squares.
  version: 1.0.0
paths:
  /board:
    ...
```

Operations

```
paths:
  /board:
    get:
      ...
    put:
      ...
```

Operation object

```yaml
paths:
  /board:
    get:
      summary: Get the whole board
      description: Retrieves the current state of the board and the winner.
      parameters:
        ...
      responses:
        ...
```

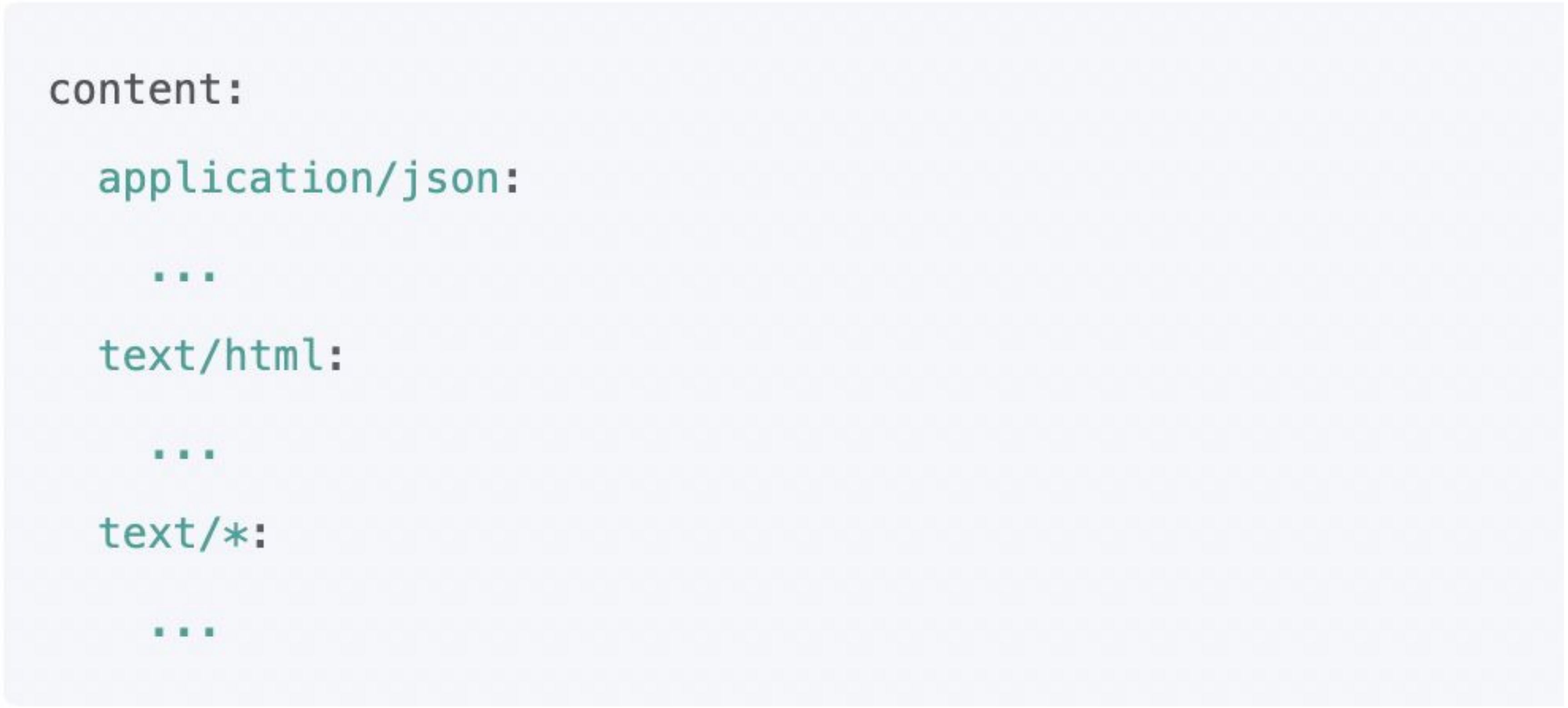
Responses

```
paths:
  /board:
    get:
      responses:
        "200":
          ...
        "404":
          ...
```

Response Objects

```yaml
paths:
  /board:
    get:
      responses:
        "200":
          description: Everything went fine.
          content:
            ...
```

Content Specification



```
content:
  application/json:
    ...
  text/html:
    ...
  text/*:
    ...
```

Schema

```yaml
content:
  application/json:
    schema:
      type: integer
      minimum: 1
      maximum: 100
```

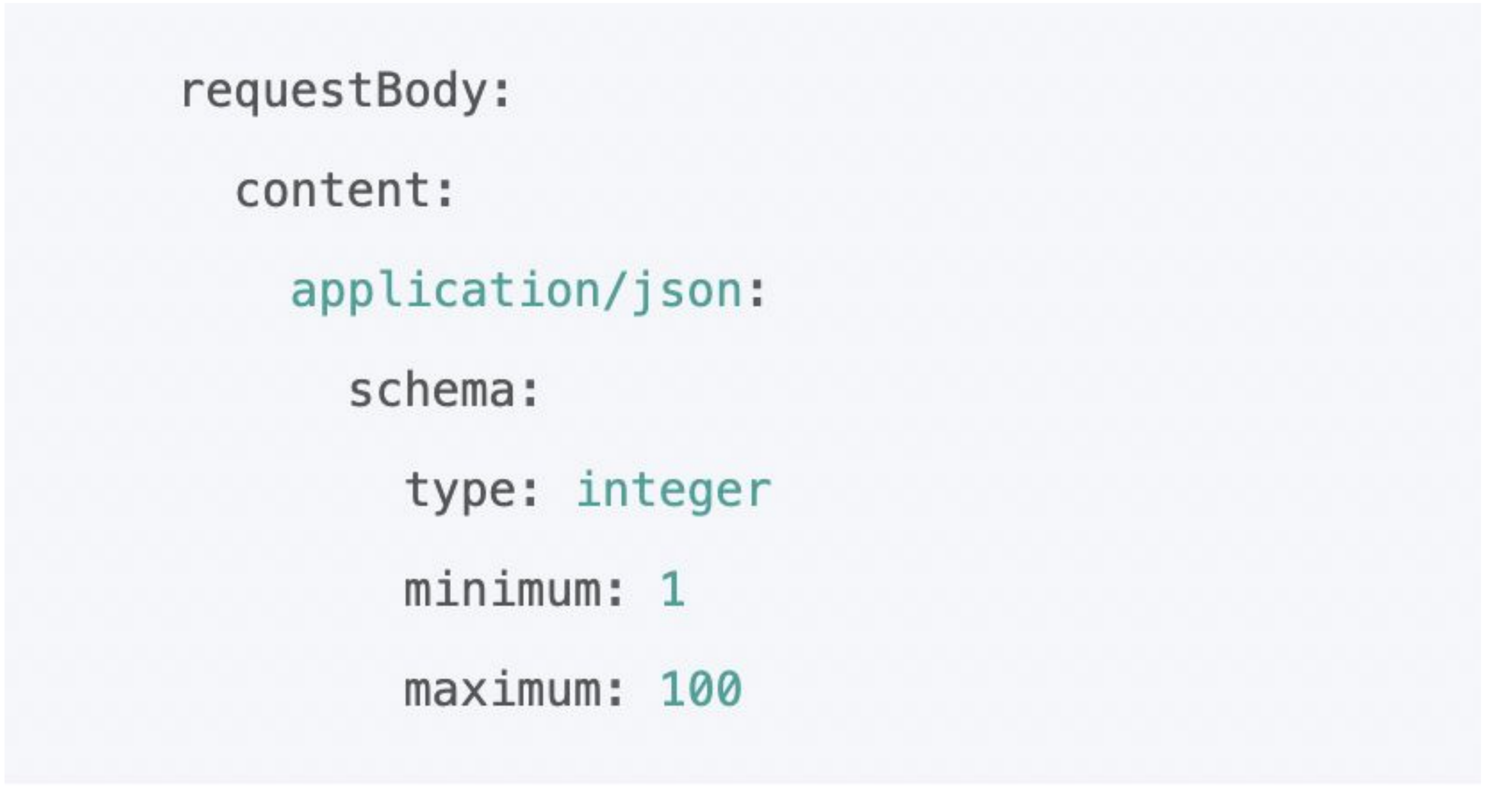
Complex Schema

```yaml
content:
  application/json:
    schema:
      type: object
      properties:
        productName:
          type: string
        productPrice:
          type: number
```

Parameters

```yaml
paths:
  /users/{id}:
    get:
      parameters:
      - name: id
        in: path
        required: true
```

Request Body



```
requestBody:
  content:
    application/json:
      schema:
        type: integer
        minimum: 1
        maximum: 100
```

Best Practices

- Design-first vs Code-first
  - Always prefer design-first!
- Single source of truth
  - The structure of the code should be *derived* from the OAS - *or* -
  - Spec should be derived from code
  - Minimize chances of code and documentation diverging
- Source code version control
- OpenAPI is … Open - public documentation better to identify problems
- Automated tools, editors - make use of them!