API Documentation

CMPU4023 - Enterprise Application Development

Documenting APIs

- Recall that one of the desirable characteristics of an API is that it is learnable by the consumer, typically a developer charged with implementing an API client in code
- Documenting APIs is challenging, in particular ensuring that the docs are accurate and up-to-date in the face of potential API changes
- There are various approaches used in the enterprise for doing this

What Should be Documented?

- Regardless of the approach there is some basic information we'd expect API documentation to provide (discussion here assumes REST)

1. A list of all of the publicly accessible endpoints
2. A list of the supported operations on each endpoint (i.e. HTTP commands)
3. For each operation, a description how each command is to be used, including the request format and response body, error codes, etc
4. A description of the behaviour of each endpoint including what default values are assumed, what constitutes valid input data, etc
5. Examples, in code or similar, of using each endpoint command
6. Any deprecation warnings related to any expected future API changes

- Ideally, all of this should be presented in a similar and predictable format

A Priori Knowledge vs Dynamic Learning

- Self-evidently, an API client must have some prior knowledge of how to use an API before it can use it - but how is this achieved?
- The big design tradeoff here is how much hard-coded knowledge a client should have baked in versus how much can learn dynamically
- A client that can learn how to use an API will be theoretically easier to maintain but only some kinds of clients can really make use of this
- A client can learn about an API in a number of ways

1. Separate accompanying documentation
2. Reference code with usage documentation
3. Queryable endpoints with capability descriptions or hints (self-documenting)

Separate Accompanying Documentation

- This is probably the most common approach to documentation and arguably the least effective
- The maintainer manually documents the API often separately from code itself
- The benefit is that the document maintainer gets to act as a wouldbe consumer which is a form of sanity testing
- The risk is that the document maintainer will miss (e.g. new version) or misunderstand some aspects of the API functionality leaving the docs incomplete or incorrect

Reference Code

- An alternative to or in addition to documenting the API itself, is to develop and document a reference client implementation in some language(s) of choice
- The advantage is that this offers a further level of abstraction for the consumer and saves time and cost for client maintainers
- The risk is that the reference implementation falls behind the API version or is incomplete in some other way
- The reference also constitutes another source of bugs and errors which can propagate to consumers

Self-documenting APIs

- In this approach, the API documentation lives closely with the API implementation allowing the consumer to query the API to learn what is supported and how it is used - i.e. it is discoverable
- There are a number of approaches to doing this in REST but each enterprise would likely develop its own conventions or modify these
- We'll consider two potential, contrasting approaches to building self-documenting APIs

1. Using a descriptive API Schema (traditional SOA philosophy)
2. Hypermedia As The Engine Of Application State (HATEOAS)

API Schema

- The idea here is that each each API endpoint could be queryable, say using an HTTP OPTIONS or GET command to discover its capabilities
- Coupled with a schema technology like JSON Schema, Open API Specification (Swagger) or WADL, the endpoint could respond with documentation details such as supported operations and attributes, attribute descriptions, example usage, etc
- The benefit is the potential for formal, consistent, rigorous and always-synchronised API documentation
- The downside is the close coupling and commitment to a specification format which may not serve all business needs

Open API Specification

```
swagger: "2.0"
info:
  version: "1.0"
  title: "Hello World API"
paths:
  /hello/{user}:
      get:
      description: Returns a greeting to the user!
      parameters:
      - name: user
            in: path
            type: string
            required: true
            description: The name of the user to greet.
      responses:
      200:
            description: Returns the greeting.
            schema:
            type: string
        400:
            description: Invalid characters in "user" were provided.
```

HATEOAS

- The a hypermedia-driven API allows the client to automatically discover and navigate to the service API endpoints by including hyperlinks within API responses

```
{
    "name": "Jane Doe",
    "links": [ {
        "rel": "self",
        "href": "https://api.example.com/customers/127678432"
    } ]
}
```

- Including hyperlinks relieves the client of having to know the logic as to how resource URIs are formed

Summary

- Documentation is essential to understanding the functionality and capabilities of an API
- Good documentation should list the available endpoints, the supported operations, the formats of messages, describe the resources and status and error codes
- It can be provided separate, in the form of reference code or be self-documenting
- The better the documentation the lower the risk of mistakes or misunderstandings occurring with API usage

References

- Web Application Description Language https://www.w3.org/Submission/wadl/
- JSON Schema (http://json-schema.org)
- Swagger (https://swagger.io)
- HATEOAS (http://restcookbook.com/Basics/hateoas/)

API Documentation

CMPU4023 - Enterprise Application Development