BITS Pilani
Pilani Campus

Full Stack Application Development- SE ZG503

Akshaya Ganesan

CSIS, WILP

BITS Pilani
Pilani Campus

Lecture No: 10 REST API Documentation

Module 4: Server Side: Implementing Web Services

- REST
- Principles of REST
- REST constraints
- Service Design with REST
  - Interaction Design with HTTP
  - Interface Design (URI)
  - Representation and Metadata design
- Implementing REST API
  - Using a Framework
  - URL Mapping
  - Routing Requests-Redirection
  - Implementing a web server
  - Processing request, response, data
- Storing data in databases
  - Models
  - Object Relational Mapper
  - Interaction with DBs
- API versioning and documentation
-

API Versioning

- Breaking changes primarily fit into the following categories:
  - Changing the request/response format (e.g. from XML to JSON)
  - Changing a property name (e.g. from name to productName) or data type on a property (e.g. from an integer to a float)
  - Adding a required field on the request (e.g. a new required header or property in a request body)
  - Removing a property on the response (e.g. removing description from a product)

API Change Management

- Effective change management in the context of an API is summarized by the following principles:
- Continue support for existing properties/endpoints
- Add new properties/endpoints rather than changing existing ones
- Thoughtfully sunset obsolete properties/endpoints

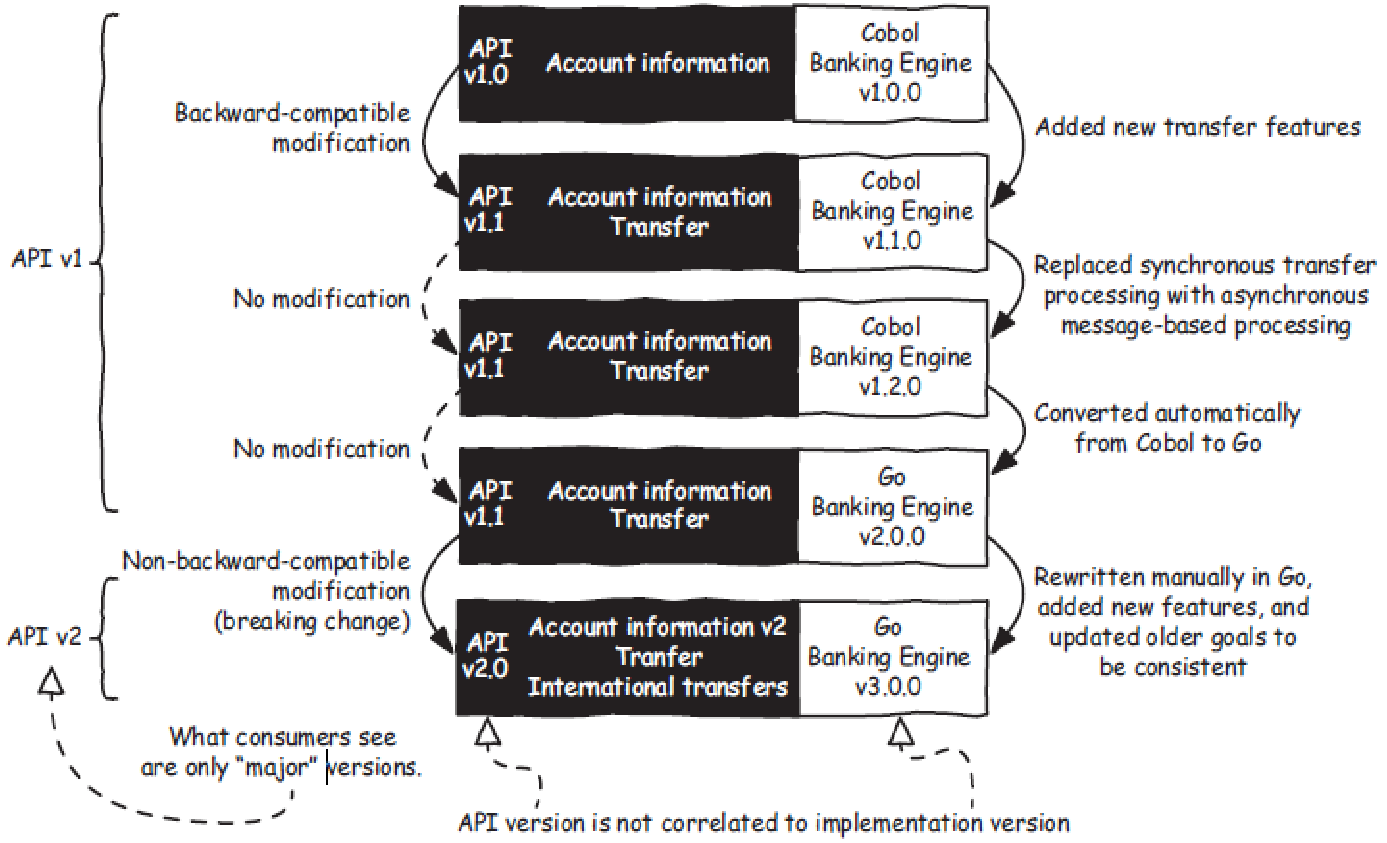
API Change Management

- We can think of levels of scope change within a tree analogy:
- Leaf - A change to an isolated endpoint with no relationship to other endpoints
- Branch - A change to a group of endpoints or a resource accessed through several endpoints
- Trunk - An application-level change, warranting a version change on most or all endpoints
- Root - A change affecting access to all API resources of all versions
- As you can see, moving from leaf to root, the changes become progressively more impactful and global in scope.

API versioning representation

- There are four different ways that we can implement the versioning
- **Versioning through the URI path**
  - http://www.example.org/v1/customer/1234
  - Ideal when major changes are introduced that completely break backward compatibility.
- **Versioning through query parameters**
  - http://www.example.org/customer/1234?version=v3
  - Ideal for simple APIs where backward compatibility is maintained across versions.
- **Versioning through custom headers**
  - Custom-Header: api-version=1
- **Versioning through content-negotiation**
  - Accept: application/vnd.adventure-works.v1+json

API Versioning


API v1.0
Account information
Cobol Banking Engine v1.0.0
Backward-compatible modification
Added new transfer features
API v1.1
Account information Transfer
Cobol Banking Engine v1.1.0
API v1
No modification
Replaced synchronous transfer processing with asynchronous message-based processing
API v1.1
Account information Transfer
Cobol Banking Engine v1.2.0
No modification
Converted automatically from Cobol to Go
API v1.1
Account information Transfer
Go Banking Engine v2.0.0
Non-backward-compatible modification (breaking change)
Rewritten manually in Go, added new features, and updated older goals to be consistent
API v2
API v2.0
Account information v2 Tranfer International transfers
Go Banking Engine v3.0.0
What consumers see are only "major" versions.
API version is not correlated to implementation version

Semantic Versioning

- format: *MAJOR.MINOR.PATCH*.
- The MAJOR digit is incremented only on breaking changes, such as adding a new mandatory parameter
- The MINOR digit is incremented when new features are added in a backward-compatible manner, like adding new HTTP methods or resource paths in a REST API.
- The PATCH digit is incremented when the modifications made involve backward-compatible bug
- This makes sense for an implementation, but not for an API.
- Semantic versioning applied to APIs consist of just two digits: *BREAKING.NONBREAKING*.
- This two-level versioning is interesting from the provider’s perspective; it helps to keep track of all the different backward-compatible and non-backward-compatible versions of an API.

API Documentation

- API documentation is a technical content deliverable containing instructions on using and integrating with an API effectively.
- API description formats like the OpenAPI/Swagger Specification have automated the documentation process, making it easier for teams to generate and maintain them.
- OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs.

OAS

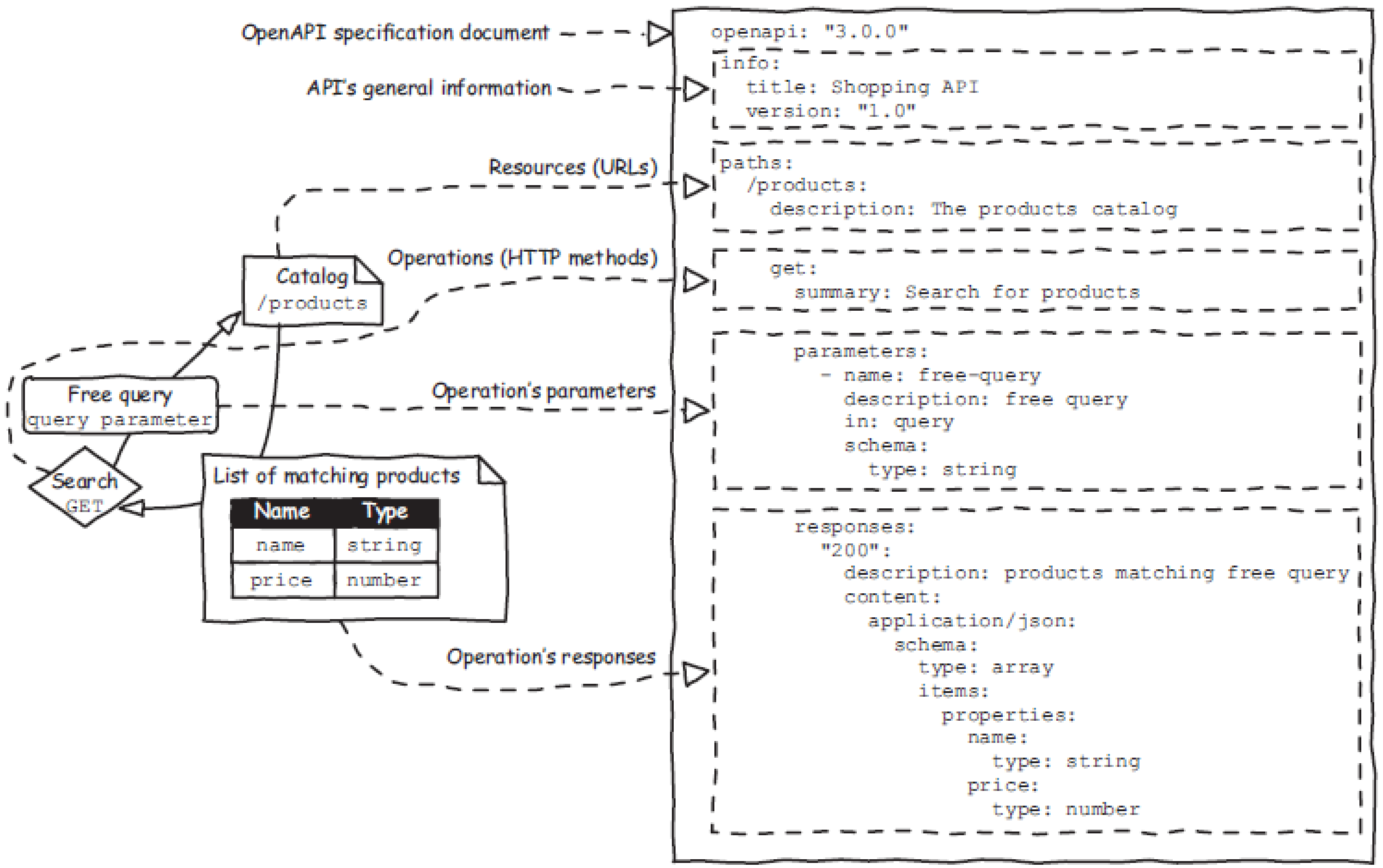
- The *OpenAPI Specication* (OAS) is a popular REST API description format
- An OpenAPI file allows you to describe your entire API, including:
    - Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
    - Operation parameters Input and output for each operation
    - Authentication methods
    - Contact information, license, terms of use and other information.
    - API specifications can be written in YAML or JSON.

Swagger

- **Swagger** is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs.
- http://swagger.io/docs/specification/basic-structure/

API Documentation

- An OAS document describing the search for products goal of the Shopping API



```yaml
openapi: "3.0.0"
info:
  title: Shopping API
  version: "1.0"
paths:
  /products:
    description: The products catalog
    get:
      summary: Search for products
      parameters:
        - name: free-query
          description: free query
          in: query
          schema:
            type: string
      responses:
        "200":
          description: products matching free query
          content:
            application/json:
              schema:
                type: array
                items:
                  properties:
                    name:
                      type: string
                    price:
                      type: number
```

Image: The Design of Web APIs by ARNAUD LAURET

API Documentation

- RAML and Blueprint are other alternatives
- API Blueprint is a markdown format to generate documentation. It was developed by Apiary in 2013 and then acquired by Oracle.
- RAML, which stands for RESTful API Modeling Language, is an API design format developed by MuleSoft in 2013 and then acquired by Salesforce.
- Swagger is an API description format developed by Wordnik in 2010 and then acquired by SmartBear. It was later renamed OpenAPI and donated to the Linux Foundation.

Thank You!