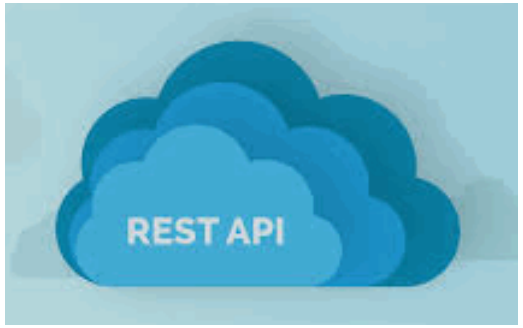
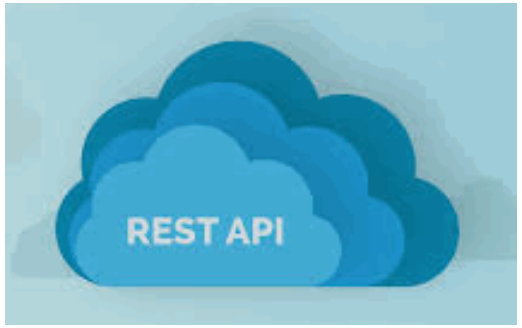


Understanding RESTful APIs and documenting them with Swagger

Presented by: Tanya Perelmuter

Date: 06/18/2018

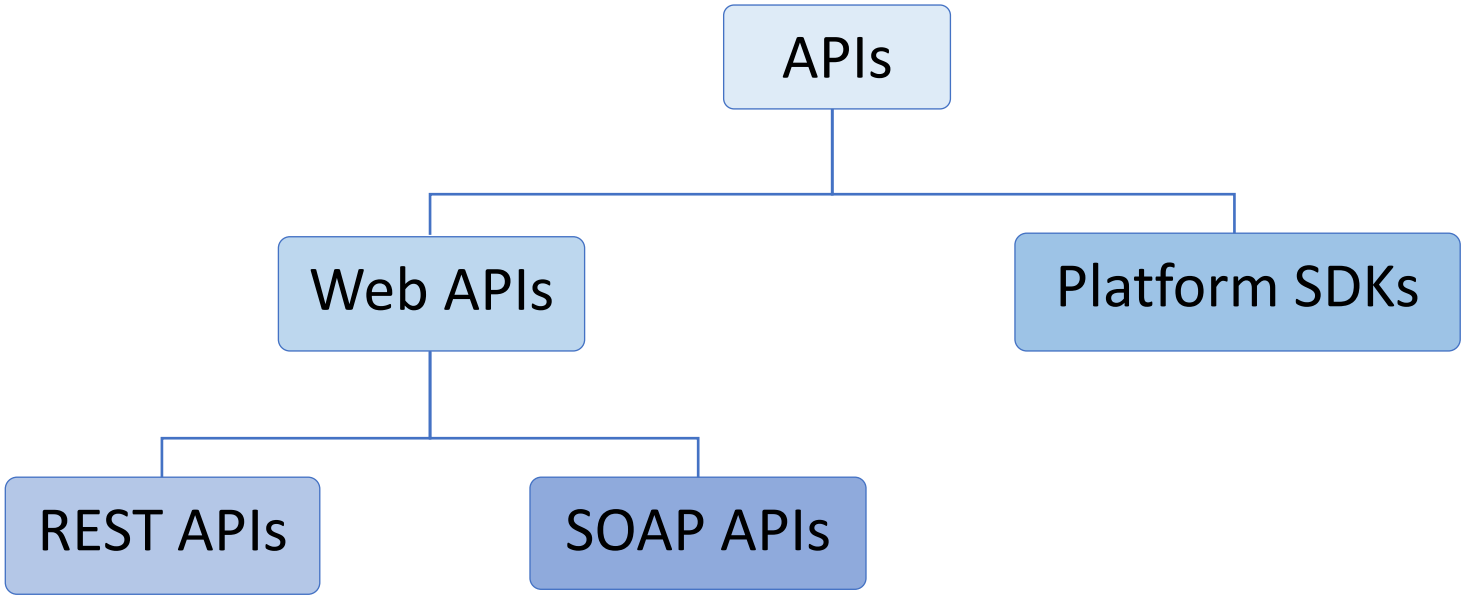
Part 1 – Understanding RESTful APIs



- API types and definitions
- REST architecture and RESTful API concepts
- REST versus SOAP
- Elements of REST API endpoint
- Minimum information set for REST API reference
- Resources for writing REST API documentation

API Types and Definitions

- API – Application Programming Interface – a collection of software functions that provides a coherent set of functionality.



- Platform SDKs – specific to programming language and platform (e.g. iOS SDK, Android SDK).
- Web APIs – agnostic to programming language and platform, for web-based applications, work over HTTP.
- SOAP – Simple Object Access Protocol – a protocol specification for exchanging structured information in the implementation of web services in computer networks; uses XML format for data interchange.
- REST – Representational State Transfer – an architectural style for distributed hypermedia systems that leverages the architecture of World Wide Web; usually uses JSON format for data interchange.

REST Architecture

- REST architecture is defined by the following six constraints:
  1. Client-server: separation of concerns between client and server, namely clients are not concerned with data storage, and servers are not concerned with the user interface or user state.
  2. Statelessness: no client context is stored on the server between requests, each request from any client contains all the information necessary to service the request, and session state is held in the client.
  3. Cacheability: responses to read requests can be defined as cacheable by servers and can be cached by clients; conditional GET requests and state expiration times support cache validation.
  4. Layered system: intermediary servers, such as proxies, gateways, and firewalls, can be introduced at various points of communication without changing the interfaces between components.
  5. Code on demand: servers can temporarily extend or customize the functionality of a client by transferring executable code, for example Java applets or client-side scripts such as JavaScript.
  6. Uniform interface: all components must interact through a uniform interface; its principles include identification of resources by URIs and hypermedia as the engine of application state (HATEOAS).
- The constraints of the REST architectural style affect the following architectural properties:
  - scalability (thanks to constraints 1, 2, 4), performance (thanks to constraints 3, 4), simplicity (thanks to constraint 6), portability of components across multiple platforms (thanks to constraint 1), modifiability (thanks to constraints 1, 5, 6), visibility (thanks to constraint 6).
- For more details on architecture, see Wikipedia article Representational state transfer.

RESTful API Concepts

- RESTful Web Services – web services that conform to the REST architectural style.
- RESTful APIs – APIs provided by RESTful web services.
- One of the main concepts of RESTful API is a resource.
  - Resources are fundamental building blocks of web-based systems. A resource is anything exposed to the Web, e.g. a document, a video clip, a device, etc. The characteristics of a resource are:
  - A resource might be a collection of objects or an individual object.
  - A resource is identified by URI (Uniform Resource Identifier). The relationship between resources and URIs is one-to-many: a URI identifies only one resource, but a resource may have more than one URI.
  - A resource is manipulated through CRUD – Create, Read, Update, Delete – operations, which are usually mapped to HTTP methods/verbs POST, GET, PUT, DELETE correspondingly.
  - A REST API endpoint is defined by a combination of a resource URI and an HTTP verb that manipulates it.

Method	URI	Description
GET	/customers	List all customers in the collection.
POST	/customers	Create a new customer.
GET	/customers/{customerId}	Retrieve a customer specified by ID.
PUT	/customers/{customerId}	Update a customer specified by ID.
DELETE	/customers/{customerId}	Delete a customer specified by ID.

REST versus SOAP

SOAP web services characteristics

- SOAP interfaces are based on operations, which are verbs that describe business logic.
- SOAP interfaces are strongly typed and must be defined in WSDL (Web Services Description Language), which is XML-based specification for defining SOAP web services' contracts.
- SOAP web services are auto-discoverable if get registered in a UDDI (Universal Description, Discovery, and Integration) -compliant registry.
- SOAP web services have high reliability due to SOAP messages' encoding (called "SOAP envelopes") and built-in retry logic.
- SOAP web services have enterprise-level security thanks to WS-Security support.

RESTful web services characteristics

- RESTful interfaces are based on resources, which are nouns that describe object model.
- RESTful interfaces might be described either in Swagger/OpenAPI specification or in RAML (RESTful API Modeling Language), which are both YAML-based specifications for defining RESTful APIs.
- RESTful web services are not auto-discoverable; therefore, REST API reference documentation is important for published RESTful APIs.
- RESTful web services have high scalability because services are "stateless" (do not store client context between requests) and hence may scale indefinitely.
- RESTful web services have high performance due to "cacheable" responses to GET requests.

Elements of REST API Endpoint

- Example: get all orders of a specified customer for year 2017

- Request:

GET http://api.ecommerce.com/v1/customers/{customerId}/orders?year=2017

HTTP Method | Base URI | Resource URI | Path Parameter | Query Parameter

Accept: application/json

Request Header | Media Type

- Response:

Status: 200 OK

HTTP Status Code

Response Body →

Content Type: application/json

Response Header | Media Type


```
[{
    "orderId" : "132",
    "customerId" : "86597",
    "productName" : "REST_API_Textbook_v1",
    "productDescription" : "REST API Textbook 1-st edition",
    "priceTotal" : 29.99,
    "placedAt" : "2017-01-10T23:43:33.741Z",
    "completed" : true,
    "completedAt" : "2017-01-11T05:11:00.562Z"
  }, {
    "orderId" : "134",
    "customerId" : "86597",
    "productName" : "Swagger_Textbook_v2",
    "productDescription" : "Swagger and OpenAPI Textbook 2-nd edition",
    "priceTotal" : 18.99,
    "placedAt" : "2017-03-05T23:43:33.741Z",
    "completed" : true,
    "completedAt" : "2017-03-06T05:11:00.562Z"
  }
]
```

Minimum Information Set for REST API Reference

- In order to write a client code that uses a published RESTful API, customers need to know at least the following information about each endpoint:
  - Request:
    1. HTTP Method (POST / GET / PUT / DELETE)
    2. Resource URI
    3. Parameters (Path, Query, etc.) including Data Type, Required/Optional, Allowable Values
    4. Request Headers
    5. Media Type (e.g. JSON, XML)
    6. Request Body Schema (for POST and PUT requests) and Request Example
    7. Authentication
  - Response:
    8. HTTP Status Codes
    9. Response Headers
    10. Media Type (e.g. JSON, XML)
    11. Response Body Schema and Response Example

Resources for Writing REST API Documentation

- Advanced REST Client – test RESTful API endpoints with this application:
  - https://install.advancedrestclient.com/#/install
- Postman – another popular application for testing RESTful API endpoints:
  - https://www.getpostman.com/apps
- HTTP request and response headers:
  - https://en.wikipedia.org/wiki/List_of_HTTP_header_fields
- HTTP response status codes:
  - https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- Online training course “Learn API Technical Writing 2: REST for Writers”:
  - https://www.udemy.com/learn-api-technical-writing-2-rest-for-writers
- O’Reilly book “RESTful Java with JAX-RS 2.0” – code examples of RESTful web services implemented in Java language according to JAX-RS specification:
  - https://www.amazon.com/RESTful-Java-JAX-RS-2-0-Distributed/dp/144936134X

Part 2 – Documenting RESTful APIs with Swagger

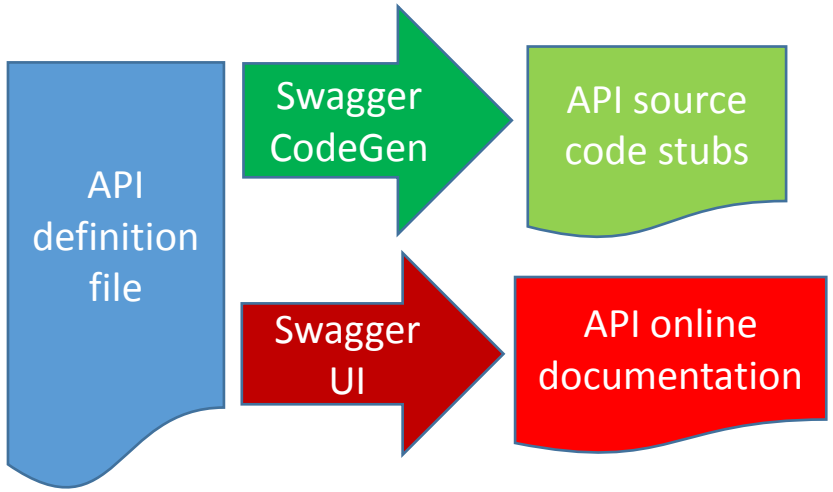
- Swagger/OAS definitions
- Swagger/OAS workflows
- Elements of API description in Swagger/OAS
- Swagger strengths and weaknesses
- Alternatives to Swagger and OAS
- Swagger/OAS resources

Swagger/OAS Definitions

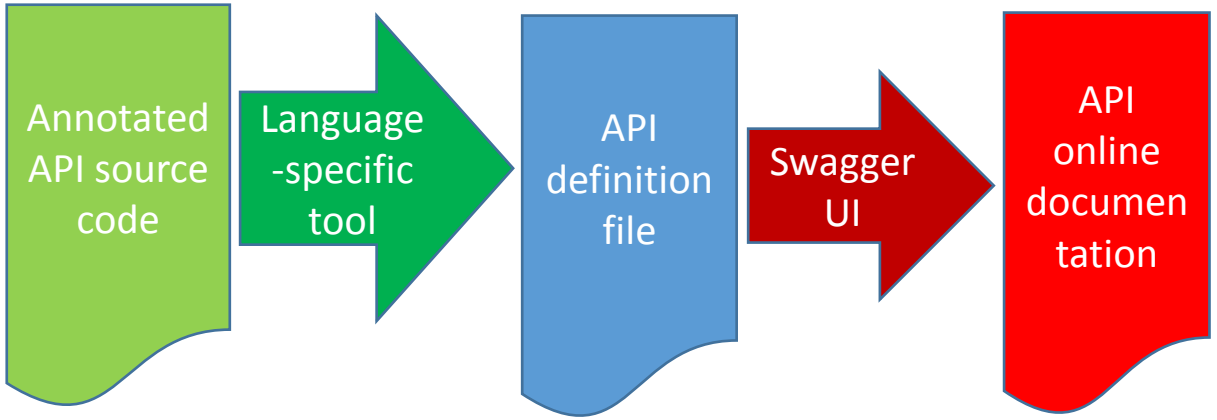
- Open API Specification (OAS) – a standard for defining RESTful APIs
  - API definition file – a YAML or JSON file describing an API according to the Open API Specification
  - YAML – a structured data format; minimizes characters compared to JSON
  - Swagger 1.0 was the specification; starting Swagger 2.0 it became the OAS
- Swagger – a set of tools compliant with the OAS
  - Swagger Editor – helps authoring and editing API definition files
  - Swagger CodeGen – generates source code stubs from API definition files
  - Swagger UI – generates online documentation from API definition files
- SwaggerHub – a platform that hosts Swagger toolkit
  - SwaggerHub is an alternative to installing Swagger toolkit on premises
  - SwaggerHub account is free for one user, offers subscription mode for a team, and enables collaboration

Swagger/OAS Workflows

- Workflow One
  1. Design an API and describe it in an API definition file
  2. Generate source code stubs from the API definition file
  3. Generate online documentation from the API definition file

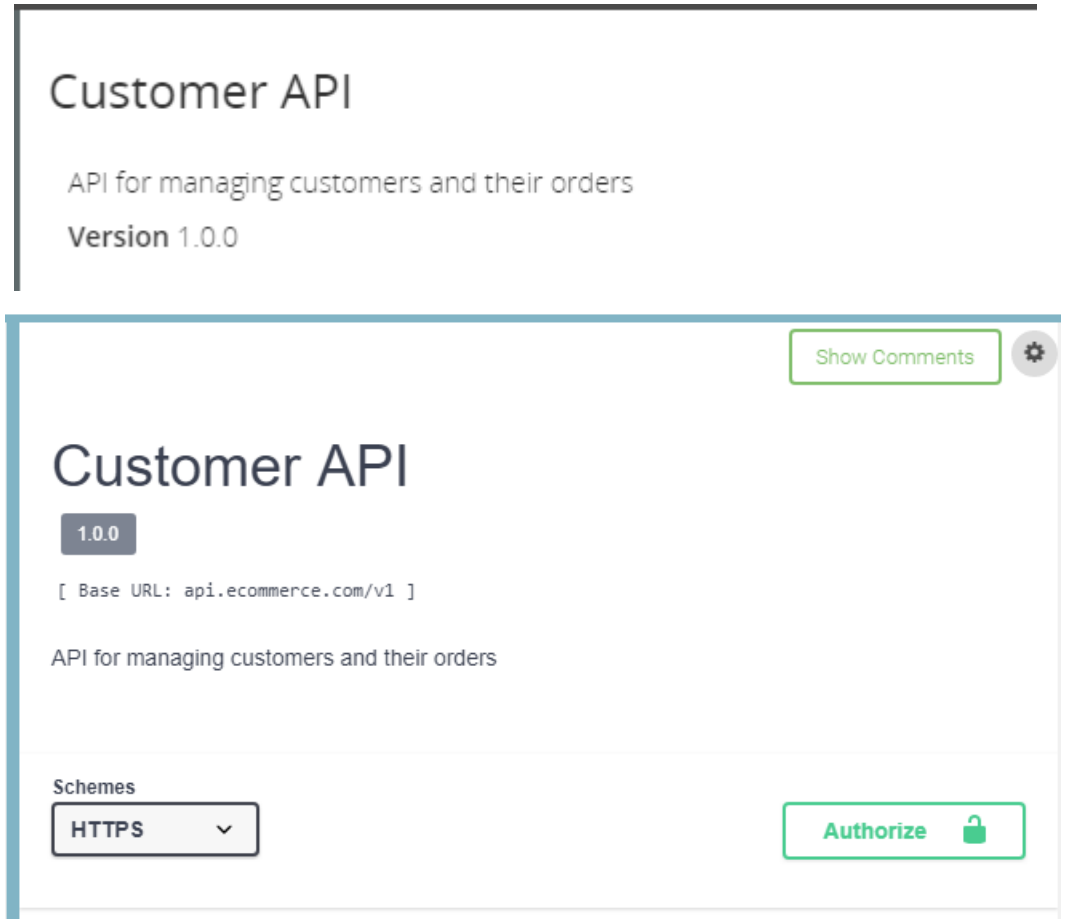


- Workflow Two
  1. Design an API and write source code without Swagger/OAS framework
  2. Annotate source code to accommodate API file generation (e.g. see swagger-core)
  3. Generate an API definition file from the annotated source code
  4. Generate online documentation from the API definition file



Defining API Metadata

Swagger Editor	Swagger UI (in editor.swagger.io and swaggerhub.com)

# Open API Specification version 	
swagger: '2.0'
 	
# Document metadata 	
info:
  version: "1.0.0"
  title: Customer API
  description: API for managing customers and their orders
 	
# URL data 	
host: api.ecommerce.com
basePath: /v1
schemes:
  - https
	

Defining Methods and URIs

Swagger Editor	Swagger UI

# Endpoints 	
paths:
  # Customers
  /customers:
  # Get list of customers filtered by criteria
  get:
  operationId: getCustomers
  description: Returns a list of customers by search criteria.
  …
  # Create a new customer
  post:
  operationId: createCustomer
  description: Creates a new customer.
  …
  # Customer
  /customers/{customerId}:
  # Update a customer
  put:
  operationId: putCustomer
  description: Updates a customer information.
  …
	Paths
	/customers
	GET /customers
	Description
	Returns a list of customers by search criteria.
	POST /customers
	Description
	Creates a new customer.
	/customers/{customerId}
	PUT /customers/{customerId}
	Description
	Updates a customer information.

Defining Query Parameters

Swagger Editor

```yaml
# Customers
/customers:
  # Get list of customers filtered by criteria
  get:
  ...
   # Query parameters
   parameters:
     # Customer status - active or not
     - name: active
       in: query
       required: false
       type: boolean
       description: Customer status

     # Year the customer account created
     - name: customerSince
       in: query
       required: false
       type: integer
       description: Year the customer joined
```

Swagger UI

GET /customers

Description

Returns a list of customers by search criteria.

Parameters

Name	Located in	Description	Required	Schema
active	query	Customer status	No	boolean
customerSince	query	Year the customer joined	No	integer

Defining Path Parameters

Swagger Editor

```yaml
# Customer
/customers/{customerId}:
  ...
  # Delete a customer
  delete:
    operationId: deleteCustomer
    description: Deletes a customer.

    # Path parameters
    parameters:
      # Customer ID
      - name: customerId
        in: path
        required: true
        type: string
        description: ID of the customer to delete
```

Swagger UI

DELETE /customers/{customerId}

Description

Deletes a customer.

Parameters

Name	Located in	Description	Required	Schema
customerId	path	ID of the customer to delete	Yes	⇄ string

Defining Request Body and Schema

Swagger Editor

```yaml
paths:
  /customers:
    post:
      parameters:
       # Request body for the new customer
       - name: newCustomer
         in: body
         required: true
         schema:
          $ref: '#/definitions/newCustomer'
          description: New customer object
...
# Schema definitions
definitions:
  # New customer
  newCustomer:
    properties:
      name:
        type: string
        description: Customer name
      address:
        type: string
        description: Customer address
    required:
      - name
```

Swagger UI

POST /customers

Description

Creates a new customer.

Parameters

Name	Located in	Description	Required	Schema
newCustomer	body	New customer object	Yes	⇄

```
▾ newCustomer {
  New customer
  name:    ▾ string *
             Customer name
  address: ▾ string
             Customer address
}
```

Defining Response Body and Schema

Swagger Editor

```yaml
paths:
  /customers:
    post:
      # List of possible responses
      responses:
        201:
          description: Created successfully
          schema:
            $ref: '#/definitions/customerSummary'
...
definitions:
  # Customer summary
  customerSummary:
    allOf:
      - $ref: '#/definitions/newCustomer'
    properties:
      customerId:
        type: integer
        description: Customer ID
      active:
        type: boolean
        description: Customer status
      customerSince:
        type: integer
        description: Year the customer joined
    required:
      - customerId
```

Swagger UI

POST /customers

Description

Creates a new customer.

Responses

Code	Description	Schema
201	Created successfully	⇄

```
customerSummary {
  Customer summary
  customerId:     integer *
                  Customer ID
  active:         boolean
                  Customer status
  customerSince:  integer
                  Year the customer joined
  all of:
    newCustomer {
      New customer
      name:     string *
                Customer name
      address:  string
                Customer address
    }
}
```

Defining Response Codes and Error Messages

Swagger Editor

```yaml
paths:
  /customers/{customerId}:
    put:
      # List of possible responses
      responses:
        200:
          description: Successful response
          schema:
            $ref: '#/definitions/customerSummary'
        404:
          description: Customer ID not found
          schema:
            $ref: '#/definitions/error'
...
definitions:
  # Customer summary
  customerSummary:
    ...
  # Error object
  error:
    properties:
      errorMessage:
        type: string
        description: Error message
      logData:
        $ref: '#/definitions/logEntry'
  # Log entry object
  logEntry:
  ...
```

Swagger UI

PUT /customers/{customerId}

Description

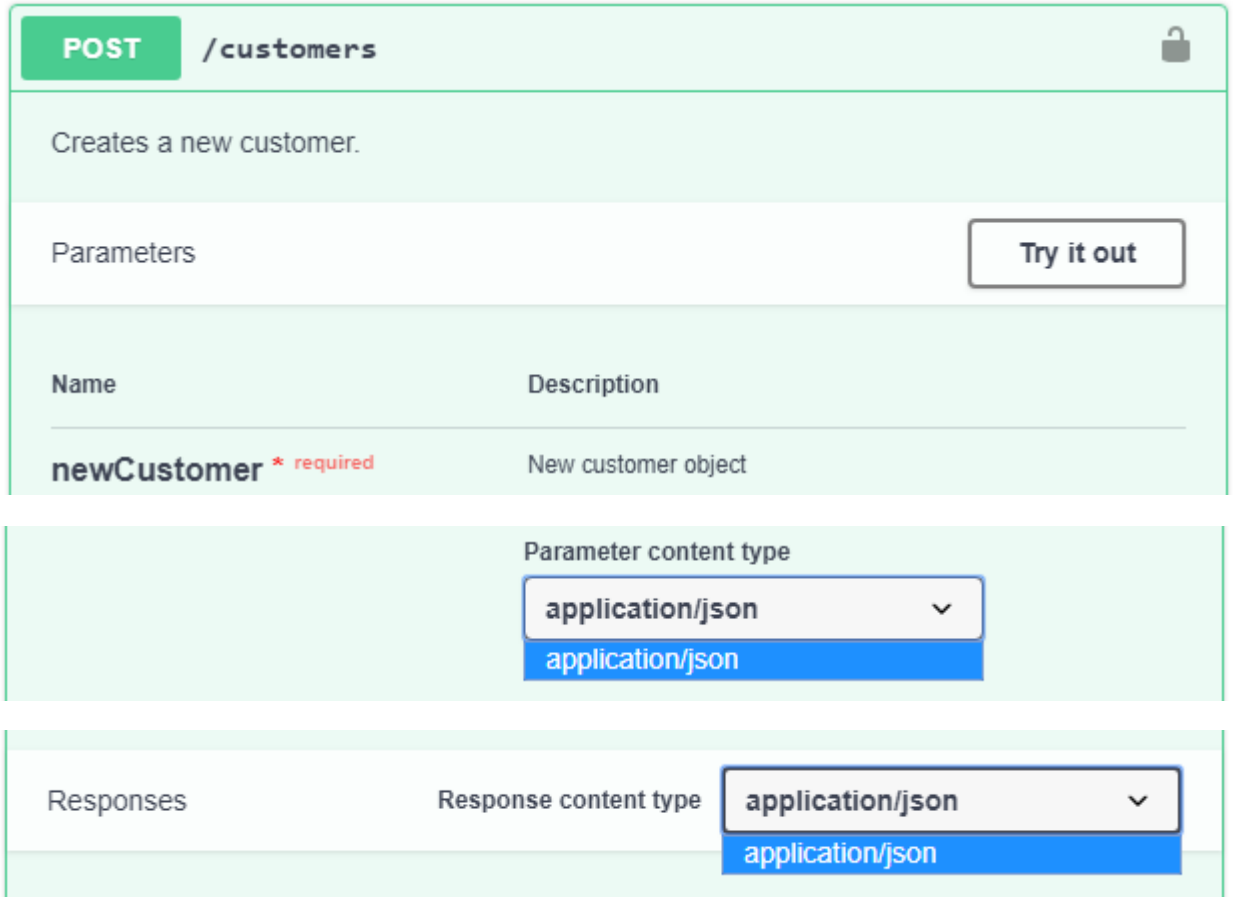
Updates a customer information.

Responses

Code	Description	Schema
200	Successful response	customerSummary { Customer summary customerId: integer * Customer ID active: boolean Customer status customerSince: integer Year the customer joined all of: newCustomer { } }
404	Customer ID not found	error { Error object errorMessage: string Error message logData: logEntry { Log entry object entry: integer Log entry number date: string Log entry date } }

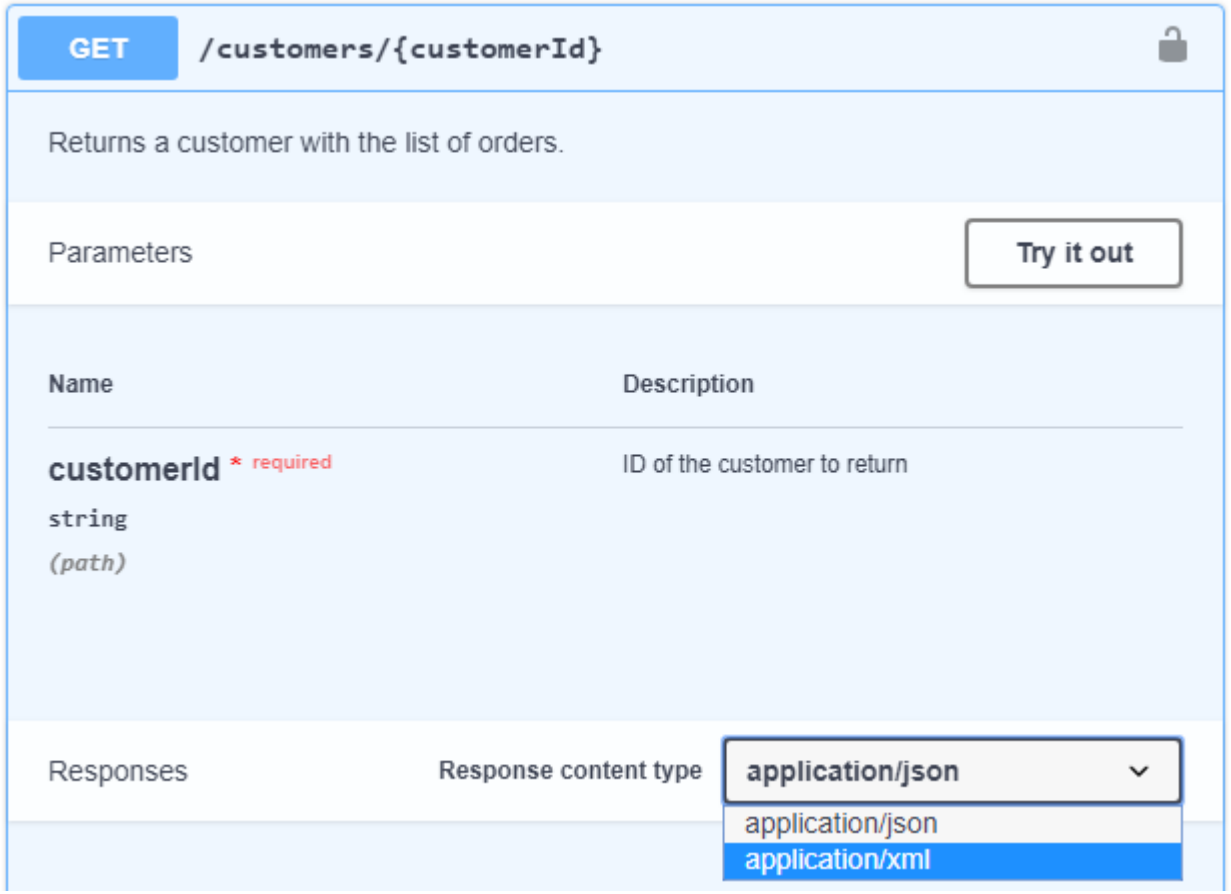
Defining API Request / Response Content Types

Swagger Editor	Swagger UI (in swaggerhub.com)

# Document metadata 	
info:
  version: "1.0.0"
  title: Customer API
  description: API for managing customers and their orders
 	
# URL data 	
host: api.ecommerce.com
basePath: /v1
schemes:
 - https
 	
# Content types of requests and responses 	
consumes:
  - application/json
produces:
  - application/json
 	
# Endpoints 	
paths:
 …
	

Defining Endpoint Request / Response Content Types

Swagger Editor	Swagger UI (in swaggerhub.com)

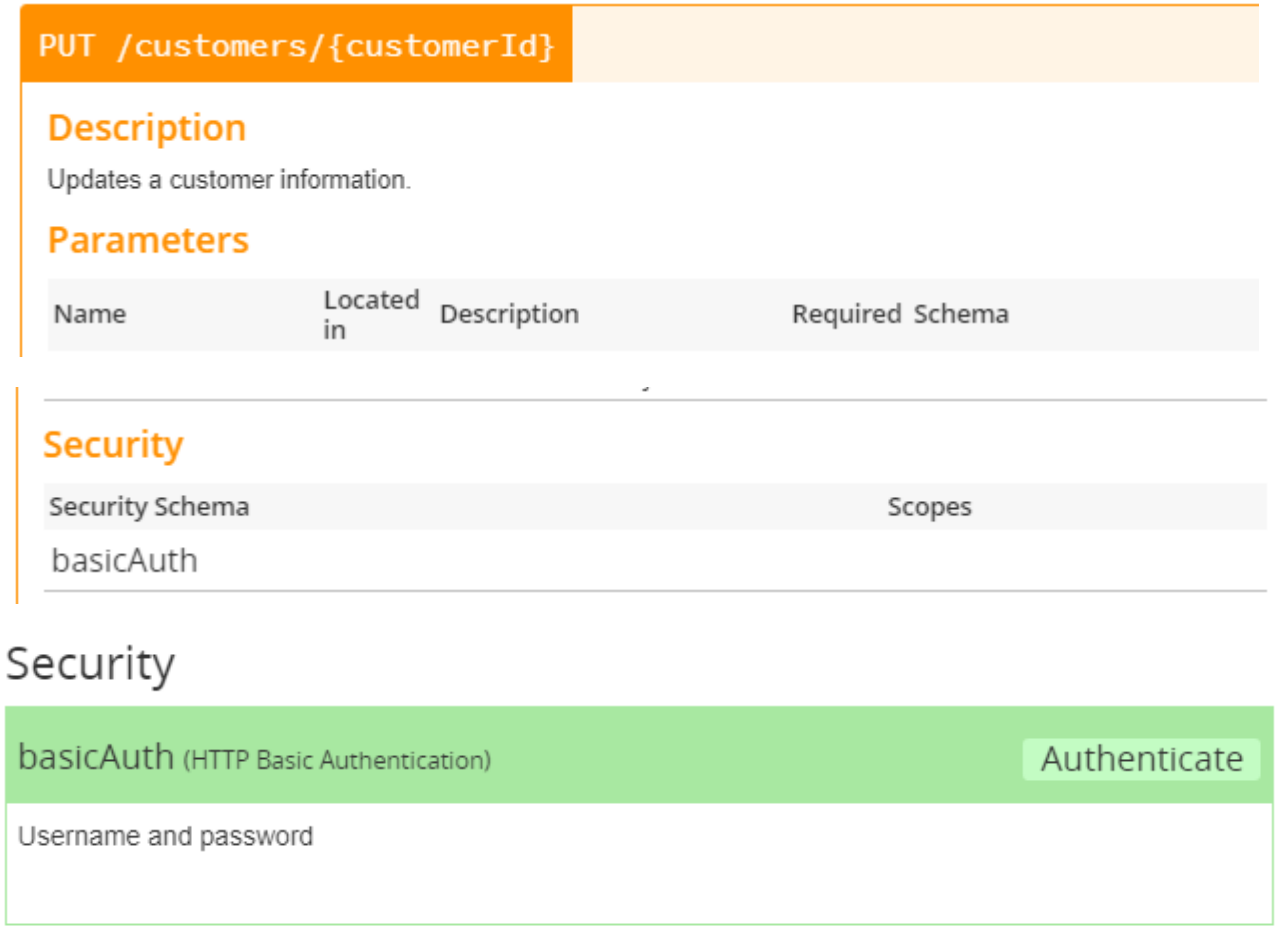
# Endpoints 	
paths:
…
  # Customer
  /customers/{customerId}:
 	
  # Get a customer with the list of orders
  get:
  operationId: getCustomer
  description: Returns a customer with the list of orders.
 	
  # Path parameters
  parameters:
  …
  # Content type of the response
  produces:
  - application/json
  - application/xml
 	
  # List of possible responses
  responses:
  …
	

Defining Security

Swagger Editor

```yaml
# Endpoints
paths:
  # Customer
  /customers/{customerId}:
    # Update a customer
    put:
      operationId: putCustomer
      description: Updates a customer information.
      # Path parameters
      parameters:
        ...
      # Basic auth security
      security:
        - basicAuth: [ ]
      # List of possible responses
      responses:
      ...
# Security definitions; can have types basic, apiKey, oauth2
securityDefinitions:
  basicAuth:
    type: basic
    description: Username and password
```

Swagger UI



Adding Descriptions for Docs – info, operations, parameters

Swagger Editor

```yaml
# Document metadata
info:
  version: "1.0.0"
  title: Customer API
  description: API for managing **customers** and their **orders**
...
paths:
  /customers:
    # Get list of customers filtered by criteria
    get:
      operationId: getCustomers
      description: Returns a list of customers by search criteria.

      parameters:
        # Customer status - active or not
        - name: active
          in: query
          required: false
          type: boolean
          description: Customer status
          ...
```

Swagger UI

Customer API

API for managing **customers** and their **orders**

Version 1.0.0

GET /customers

Description

Returns a list of customers by search criteria.

Parameters

Name	Located in	Description	Required	Schema
active	query	Customer status	No	⇄ boolean
customerSince	query	Year the customer joined	No	⇄ integer

Adding Descriptions for Docs – responses, schemas, security

Swagger Editor

```
# List of possible responses
  responses:
    200:
      description: Successful response
    ...
    404:
      description: Customer ID not found
      ...
# Schema definitions
definitions:
updateCustomer:
  description: Update customer object
  properties:
    name:
      type: string
      description: Customer name
    address:
      type: string
      description: Customer address
    ...
# Security definitions
securityDefinitions:
  basicAuth:
    type: basic
    description: Username and password
```

Swagger UI

Responses

Code	Description	Schema
200	Successful response	customerSummary { }
401	Not authorized	error { }
404	Customer ID not found	error { }

updateCustomer	body	Customer update object	Yes	updateCustomer { Update customer object name: string Customer name address: string Customer address active: boolean Customer status }

Security

basicAuth (HTTP Basic Authentication) — Authenticate

Username and password

Providing Examples of Query Parameters

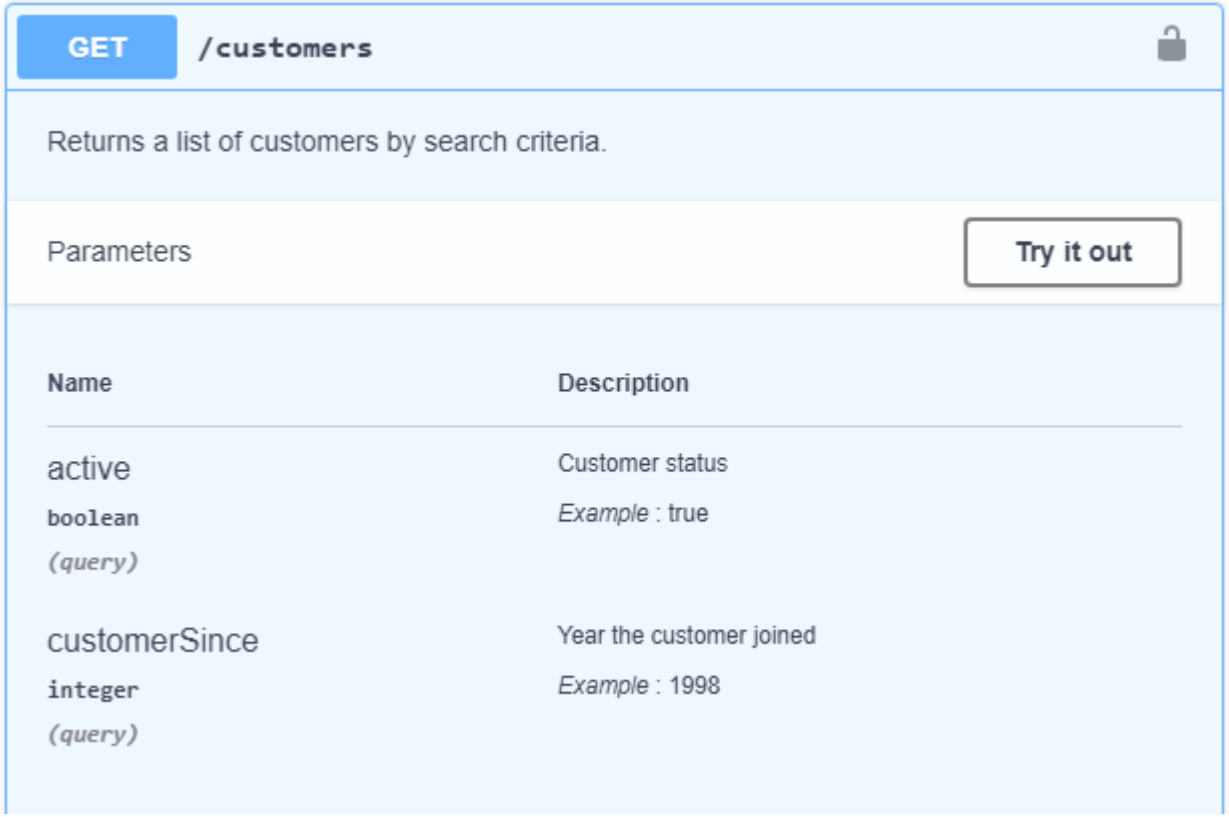
Swagger Editor

```yaml
paths:
  /customers:
    # Get list of customers filtered by criteria
    get:
      operationId: getCustomers
      description: Returns a list of customers by search criteria.

      # Query parameters
      parameters:
        # Customer status - active or not
        - name: active
          in: query
          required: false
          type: boolean
          example: true
          description: Customer status

        # Year the customer account created
        - name: customerSince
          in: query
          required: false
          type: integer
          example: 1998
          description: Year the customer joined
```

Swagger UI (in swaggerhub.com)

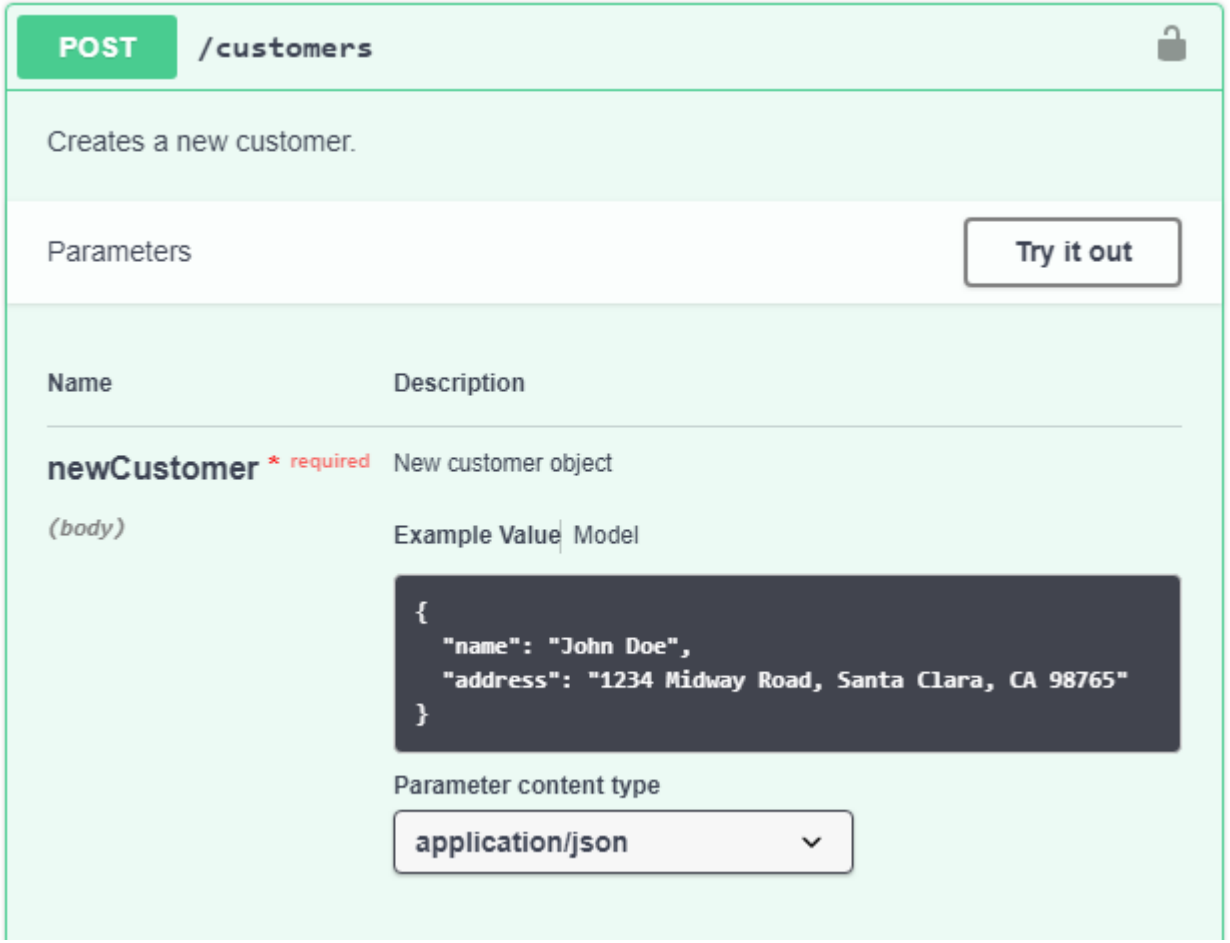


Providing Examples of Request Body

Swagger Editor

```yaml
paths:
  /customers:
    post:
      parameters:
       # Request body for the new customer
       - name: newCustomer
         in: body
         required: true
         schema:
          $ref: '#/definitions/newCustomer'
          description: New customer object
...
definitions:
 # New customer
  newCustomer:
    properties:
      name:
        type: string
        description: Customer name
        example: John Doe
      address:
        type: string
        description: Customer address
        example: 1234 Midway Road, Santa Clara, CA 98765
```

Swagger UI (in swaggerhub.com)



Swagger Strengths and Weaknesses

Swagger facilitates:

- Creating comprehensive reference documentation for RESTful APIs, covering all necessary elements for use of each API endpoint.
- Auto-generating updated documentation if and when an API definition has changed.
- Providing dynamic and interactive documentation experience, with ability to test API endpoints and collect responses.

Swagger DOES NOT facilitate:

- Creating conceptual documentation, such as architecture of the service, key concepts and objects, data model.
- Creating workflow documentation, such as common tasks, scenarios, use cases, dependencies between API calls, order of API calls.
- Providing "Getting Started" instructions for the service usage, such as system requirements, installation, and setup.

Alternatives to Swagger and OAS

Alternatives to Swagger

- DapperDox
  - Supports OAS, better UI look & feel
  - http://dapperdox.io
- Swagger UI Variants
  - Open-source modifications to Swagger UI
  - https://github.com/jensoleg/swagger-ui
- ReadMe.io
  - Supports OAS, integrates Overview documentation into API Reference
  - http://readme.io
- StopLight.io
  - Commercial platform for OAS files
  - http://stoplight.io

Alternatives to OAS

- RAML
  - RESTful API Modeling Language
  - Uses YAML format
  - https://raml.org
- API Blueprint
  - Run by the company Apiary
  - Uses markdown files with special formatting
  - https://apiary.io

What is new in OpenAPI 3.0

- OpenAPI 3.0 supports multiple hosts
  - Use case: facilitates API testing on a development server, a staging server, and a production server
  - See https://swagger.io/specification/#serverObject
- OpenAPI 3.0 supports callbacks
  - Use cases: asynchronous processing, subscription to events or notifications, etc.
  - See https://swagger.io/specification/#callbackObject
- OpenAPI 3.0 supports multipart document handling
  - Use case: divide a definition of a large API into multiple files, for better maintainability / readability
  - See https://swagger.io/specification/#documentStructure
- OpenAPI 3.0 treats request body as its own entity rather than type of parameters
  - See https://swagger.io/specification/#requestBodyObject
- OpenAPI 3.0 enhanced and simplified security definitions
  - See https://swagger.io/specification/#securitySchemeObject

Swagger/OAS Resources

- Swagger toolkit (Swagger Editor, Swagger CodeGen, Swagger UI):
  - https://swagger.io/tools/
- RESTful API documentation example with Swagger/OAS:
  - http://petstore.swagger.io/
- Swagger Editor – create and edit your API definition files:
  - https://editor2.swagger.io/
- Swagger Hub – create a free account to host your API definitions:
  - https://swaggerhub.com/
- Online training course “Learn Swagger and the Open API Specification”:
  - https://www.udemy.com/learn-swagger-and-the-open-api-specification/
- Open API Specification – Version 3.0.0:
  - https://swagger.io/specification/