OpenAPI

https://bell-sw.com/blog/documenting-rest-api-with-swagger-in-spring-boot-3/

OpenAPI Course Introduction-> Overview of OpenAPI -> Defining Services with OpenAI -> OpenAISchema

Swagger Hub -> used for editing openAPI specifications. This is an editor like vs code.

Overview OpenAPI:

OpenAPI is backed up by a schema (showing how the document can be laid out) from which we can generate object model.

Because it is a structured document it can be read programmatically.

OpenAPI tools - converters, validators, gui editors, mock services, sdk generators, documentation

OpenAPI codeGen: Generate Server Code for 20+ different languages; Client code for 40+ languages

Defines what does the API supposed to do.

Unit Tests use OpenAPI to validate the requests and responses (both locally and in CI/CD)

So in our built in development process if we do something not related to OpenAPI specification our test will fail.

Swagger is precursor to OpenAPI. It is the first edition of what became OpenAPI.

Company SmartBear bought them or acquired them and its 2.0 version named it OpenAPI.

Yml -> https://learnxinyminutes.com/docs/yaml/

OpenAPI is also known as OA
So OA3 is there (i.e. version 3.0)

OA3 had some redefined components from that of previous (i.e. standardised the components)

OpenAPI Specification mentions some structures like tags, components, servers, security, info, paths, externalDocs which are objects defined in OpenAPI schema.

.yml or .yaml both are accepted

In YAML, single quotes `' '` are used to explicitly denote a scalar string. They ensure that the string is treated as a plain string and not subject to any interpretation or special characters. Here are some use cases for single quotes in YAML:

1. **Preserving Special Characters**: Single quotes are useful when you want to preserve special characters or prevent them from being interpreted. For example:

```yaml
message: 'This is a string with special characters like !, *, etc.'
```

   In this case, the exclamation mark `!` and asterisk `*` are preserved as part of the string.

2. **Literal Strings**: Single quotes can be used for literal strings where you want to preserve the exact content without any interpretation. For example:

```yaml
multiline_string: '
This is a
multiline string
with preserved newlines.'
```

   The newlines and whitespace in the string are preserved exactly as written.

3. **Forcing Strings**: Sometimes, when a string could be interpreted as a different data type, using single quotes ensures it's treated as a string. For example:

```yaml
number_as_string: '123'
```

   Without the single quotes, `123` could be interpreted as a numeric value instead of a string.

Single quotes are particularly useful when you need to ensure that the string content is treated exactly as written, without any interpretation or processing by YAML parsers.

We can use 2 or 4 spaces for indentation.

Defining Micro-service with OpenAI 2.0

For examples we can see https://github.com/OAI/OpenAPI-Specification

Minimum fields to describe an OpenAPI object are:

```
openapi: 3.0.0
Info:
        version: ‘1.0’
        title : ‘Learning Open API’
        description: ’Specification for OpenAPI Course’
paths: {}
```

paths object is required can be empty by specifying empty array.

Control + space gives different fields to include in each field which itself is an object in swagger hub

https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.2.md#info-object

https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.2.md#server-object

e.g.

```
openapi: 3.0.2
info:
  title: OpenAPI Course
  description: Specification for OpenAPI Course
  termsOfService: http://example.com/terms/
  contact:
    name: John Thompson
    url: https://springframework.guru
    email: john@springframework.guru
  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0.html
  version: "1.0"
servers:
- url: /
paths: {}
components: {}
```

https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.2.md#pathsObject

e.g.

```
paths :
        /v1/customers:
                get:
                        responses:
                                '200' :
                                        description : List of Customer
```

Schema is metadata about the data. This is going to specify the API. What the API expects when we are giving the data and what we can expect to get back from the data e.i name types, full object description.

https://json-schema.org/understanding-json-schema/index.html

Json is loosely typed. OpenAPI part of the specification is tied to defining data types. So when we post a JSON object to the server, OpenAPI is gonna give you features of JSON schema to specify that object that we wanna be posting or if we' are getting data back as a json object or even an xml object from an api, OpenAPI is largely going to use JSON schema.

In JSON, there's no explicit need to include a `type` property when defining arrays of numbers, strings, or objects. JSON's syntax inherently allows you to define arrays using square brackets `[]` and objects using curly braces `{}`. The `type` property is more commonly used in JSON Schema, where it serves to specify the data type of a property.

However, in JSON Schema, when you're defining the structure and validation rules for JSON data, you might indeed use the `type` property to specify the type of the array or object. For example:

```json
{
  "type": "array",
  "items": {
    "type": "string"
  }
}
```

In this JSON Schema snippet, the `type` property is used to specify that the data should be an array, and the `items` property inside specifies that the items within the array should be of type string.

But in regular JSON data, when you're simply describing the structure of the data without enforcing validation rules, the `type` property is not mandatory.

Ah, I see the confusion. The `items` property in the JSON Schema you provided specifies the schema for each item in the array, rather than the actual items themselves. This property defines the expected type of each item within the array but does not hold the items themselves.

To add items to an array using this schema, you would create a JSON object with an array property and populate it with items. The `items` property in the schema just defines the expected type of those items. Here's an example:

```json
{
  "type": "array",
  "items": {
    "type": "string"
  },
  "values": ["apple", "banana", "orange"]
}
```

In this example, `"values"` is the key representing the array, and `["apple", "banana", "orange"]` is the array of strings. The `"items"` property specifies that each item in the array should be of type `"string"`. You can add more string items to the `"values"` array as needed, following the same structure.

We are going to describe the object using OpenAPI, YAML which is going to be extending the JSON schema for validation.

https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.2.md#data-types

However, to support documentation needs, the format property is an open string-valued property, and can have any value. Formats such as "email", "uuid", and so on, MAY be used even though undefined by this specification. Types that are not accompanied by a format property follow the type definition in the JSON Schema. Tools that do not recognise a specific format MAY default back to the type alone, as if the format is not specified.

Boolean type takes 0/1.

Sure, let's say you're defining a schema for an API parameter that represents an email address. You can use the `format` property to specify that the parameter should adhere to the email format. Here's an example in OpenAPI Specification (OAS) YAML format:

```yaml
parameters:
  - name: email
    in: query
    description: Email address of the user
    required: true
    schema:
      type: string
      format: email
```

In this example:

- The `type` property specifies that the parameter is of type `string`.
- The `format` property specifies that the string should adhere to the email format.

This tells developers and tools that the value of the `email` parameter should be in the format of an email address. While the OAS specification doesn't define every possible format (like "email"), it allows you to use such formats for better documentation and validation. If a tool encounters the "email" format and doesn't recognise it, it can still understand that the parameter is a string and interpret it accordingly, falling back to the `type` property.

e.g of defining the schema

```
openapi: 3.0.0
info:
  version: '1.0'
  title: openApi
  description: it contains intro to OpenAPI
paths:
    /v1/customer:
      'get':
```

```
    responses:
      '200':
        description: return a list of object
        content:
          application/json:
            schema:
              type: array
              minItems: 1
              maxItems: 10
              items:
                type: string
                minLength: 1
                maxLength: 100

servers:
  # Added by API Auto Mocking Plugin
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/prakumar/intro_open_api/1.0
```

We are telling that from the api we are expecting a return type of array containing at least 1 item and not more than 10 and for each item in array we want it as string and whose length should be between 1 and 100 inclusive.

More complex example of using object inside object

```
openapi: 3.0.2
info:
  title: OpenAPI Course
  description: Specification for OpenAPI Course
  termsOfService: http://example.com/terms/
  contact:
    name: John Thompson
    url: https://springframework.guru
    email: john@springframework.guru
  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0.html
  version: "1.0"
servers:
- url: https://dev.example.com
  description: Development Server
- url: https://qa.example.com
  description: QA Server
- url: https://prod.example.com
  description: Production Server
paths:
```

```
/v1/customers:
  get:
    responses:
      200:
        description: List of Customers
        content:
          application/json:
            schema:
              type: array
               minItems: 1
               maxItems: 10
               description: list of customer objects
               items:
                       description: customer object
                       type: object
                       properties:
                               id:
                                       type: string
                                       format: uuid
                               firstName:
                                       type: string
                                       minLength: 2
                                       maxLength: 100
                                       example: Pratik
                               lastName:
                                       type: string
                                       minLength: 2
                                       maxLength: 100
                                       example: Kumar
                               address:
                                       type: object
                                       properties:
                                               line1:
                                                       type: string
                                                       example: 123 Main
                                               city:
                                                       type: string
                                                       example: St Pete
                                               stateCode:
                                                        maxLength: 2
                                                        minLength: 2
                                                        type: string
                                                       description: 2 Letter State Code
                                                       # other way of specifying
                                                       # enum : [AL, AK, AZ, AR, CA]
                                                       enum:
                                                               - AL
```

```
                                                - AK
                                                - AZ
                                                - AR
                                                - CA
                                    zipCode
                                          type: string
                                          example: 33071
```

Properties define a map of schema objects. So each of the properties is going to be an actual schema object. We can also add enumerations.

A lot of times we are going to use schema objects, so the OpenAPI components is a way we can standardise and come up with a component and then reuse it everywhere. It saves a lot of time in coding and improves the quality of specifications.

We can refer using the ref object

```
{
      “$ref”: “#/components/schemas/Pet”
}
// this is going to search within the document
// # -> means this document
```

We can also refer to a file in the same directory of the file system (file system reference). It is a relative path.

```
{
      “$ref”: “Pet.json”
}
```

A JSON Reference is a JSON object, which contains a member named
"$ref", which has a JSON string value. Example:

  { "$ref": "http://example.com/example.json#/foo/bar" }

https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.2.md#components-object
It is a top level component.

  If a JSON value does not have these characteristics, then it SHOULD
  NOT be interpreted as a JSON Reference.

  The "$ref" string value contains a URI [RFC3986], which identifies
  the location of the JSON value being referenced. It is an error
  condition if the string value does not conform to URI syntax rules.
  Any members other than "$ref" in a JSON Reference object SHALL be

  ignored.

So we can say that we can also mention some resource on internet. (Like having resources on GitHub)

e.g.

```
paths:
  /v1/customers:
    get:
      responses:
        200:
          description: List of Customers
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/CustomerList'
  /v1/beers:
    get:
      responses:
        200:
          description: List of Beers
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/inline_response_200'
        404:
          description: No Beers Found
components:
  schemas:
    Address:
      type: object
      properties:
        line1:
          type: string
          example: 123 main
        city:
          type: string
          example: St Pete
        stateCode:
          maxLength: 2
          minLength: 2
          type: string
          description: 2 Letter State Code
          enum:
```

```yaml
        - AL
        - AK
        - AZ
        - AR
        - CA
      zipCode:
        type: string
        example: "33701"
  Customer:
    type: object
    properties:
      id:
        type: string
        format: uuid
      firstName:
        maxLength: 100
        minLength: 2
        type: string
        example: John
      lastName:
        maxLength: 100
        minLength: 2
        type: string
        example: Thompson
      address:
        $ref: '#/components/schemas/Address'
    description: customer object
  CustomerList:
    maxItems: 100
    minItems: 1
    type: array
    description: List of Customers
    items:
      $ref: '#/components/schemas/Customer'
  v1beers_brewery:
    type: object
    properties:
      name:
        type: string
      location:
        type: string
  inline_response_200:
    type: object
    properties:
      beerName:
        type: string
      style:
        type: string
        enum:
        - ALE
        - PALE_ALE
        - IPA
        - WHEAT
        - LAGER
      price:
        type: number
        format: float
      quantityOnHand:
        type: integer
        format: int32
      brewery:
        $ref: '#/components/schemas/v1beers_brewery'
    description: Beer Object
```

OpenAPI Object Inheritance

E.g of use of allOf property

```
BeerPagedList:
      type: object
      properties:
        content:
          $ref: '#/components/schemas/BeerList'
      allOf:
      - $ref: '#/components/schemas/PagedResponse'
    PagedResponse:
      type: object
      properties:
        pageable:
          $ref: '#/components/schemas/PagedResponse_pageable'
        totalPages:
          type: integer
          format: int32
        last:
          type: boolean
        totalElements:
          type: integer
          format: int32
        size:
          type: integer
          format: int32
        number:
          type: integer
          format: int32
        numberOfElements:
          type: integer
          format: int32
        sort:
          $ref: '#/components/schemas/PagedResponse_pageable_sort'
        first:
          type: boolean
    PagedResponse_pageable_sort:
      type: object
      properties:
        sorted:
          type: boolean
        unsorted:
          type: boolean
    PagedResponse_pageable:
      type: object
      properties:
        sort:
          $ref: '#/components/schemas/PagedResponse_pageable_sort'
        offset:
          type: integer
          format: int32
        pageNumber:
          type: integer
          format: int32
        pageSize:
          type: integer
          format: int32
        paged:
          type: boolean
        unpaged:
          type: boolean
```

OpenAPI Parameters:

We gonna look out at request or query parameter (appended using ?) and path parameter.
It can also be used to describe the header and cookie values.

https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.2.md#parameter-object

There are four possible parameter locations specified by the in field:

- path - Used together with Path Templating, where the parameter value is actually part of the operation's URL. This does not include the host or base path of the API. For example, in /items/{itemId}, the path parameter is itemId.
- query - Parameters that are appended to the URL. For example, in /items?id=###, the query parameter is id.
- header - Custom headers that are expected as part of the request. Note that RFC7230 states header names are case insensitive.
- cookie - Used to pass a specific cookie value to the API.

name and in are the 2 required properties.

In the context of the OpenAPI Specification (OAS), a schema in a parameter definition is used to describe the structure and data type of the parameter's value. It provides detailed information about the expected format and constraints of the parameter's value.

E.g.

```
paths:
  /v1/customers:
    get:
      parameters:
      - name: pageNumber
        in: query
        description: Page Number
        required: false
        style: form
        explode: true
        schema:
          type: integer
          format: int32
          default: 1
      - name: pageSize
        in: query
        description: Page Size
        required: false
        style: form
        explode: true
        schema:
          type: integer
          format: int32
          default: 25
      responses:
        200:
          description: List of Customers
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/CustomerPagedList'
```

We can define the components of parameter for reusability.

Overall the api looks like

```
openapi: 3.0.2
info:
  title: OpenAPI Course
  description: Specification for OpenAPI Course
  termsOfService: http://example.com/terms/
  contact:
    name: John Thompson
    url: https://springframework.guru
    email: john@springframework.guru
  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0.html
  version: "1.0"
servers:
- url: https://dev.example.com
  description: Development Server
- url: https://qa.example.com
  description: QA Server
- url: https://prod.example.com
  description: Production Server
paths:
  /v1/customers:
    get:
      parameters:
      - $ref : '#/components/parameters/PageNumberParameter'
      - $ref : '#/components/parameters/PageSizeParameter'
      responses:
        200:
          description: List of Customers
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/CustomerPagedList'
  /v1/beers:
    get:
```

```yaml
      parameters:
      - $ref : '#/components/parameters/PageNumberParameter'
      - $ref : '#/components/parameters/PageSizeParameter'
      responses:
        200:
          description: List of Beers
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/BeerPagedList'
        404:
          description: No Beers Found
components:
  schemas:
    Address:
      type: object
      properties:
        line1:
          type: string
          example: 123 main
        city:
          type: string
          example: St Pete
        stateCode:
          maxLength: 2
          minLength: 2
          type: string
          description: 2 Letter State Code
          enum:
          - AL
          - AK
          - AZ
          - AR
          - CA
        zipCode:
          type: string
          example: "33701"
    Customer:
      type: object
      properties:
        id:
          type: string
          format: uuid
        firstName:
          maxLength: 100
          minLength: 2
          type: string
          example: John
        lastName:
          maxLength: 100
          minLength: 2
          type: string
          example: Thompson
        address:
          $ref: '#/components/schemas/Address'
      description: customer object
    CustomerList:
      maxItems: 100
      minItems: 1
      type: array
      description: List of Customers
      items:
        $ref: '#/components/schemas/Customer'
    CustomerPagedList:
      type: object
      properties:
        content:
          $ref: '#/components/schemas/CustomerList'
      allOf:
      - $ref: '#/components/schemas/PagedResponse'
    Brewery:
      type: object
      properties:
        name:
          type: string
        location:
          type: string
    Beer:
      type: object
      properties:
        beerName:
          type: string
        style:
          type: string
          enum:
          - ALE
          - PALE_ALE
          - IPA
          - WHEAT
          - LAGER
        price:
          type: number
          format: float
        quantityOnHand:
```

```yaml
          type: integer
          format: int32
        brewery:
          $ref: '#/components/schemas/Brewery'
      description: Beer Object
    BeerList:
      type: array
      items:
        $ref: '#/components/schemas/Beer'
    BeerPagedList:
      type: object
      properties:
        content:
          $ref: '#/components/schemas/BeerList'
      allOf:
      - $ref: '#/components/schemas/PagedResponse'
    PagedResponse:
      type: object
      properties:
        pageable:
          $ref: '#/components/schemas/PagedResponse_pageable'
        totalPages:
          type: integer
          format: int32
        last:
          type: boolean
        totalElements:
          type: integer
          format: int32
        size:
          type: integer
          format: int32
        number:
          type: integer
          format: int32
        numberOfElements:
          type: integer
          format: int32
        sort:
          $ref: '#/components/schemas/PagedResponse_pageable_sort'
        first:
          type: boolean
    PagedResponse_pageable_sort:
      type: object
      properties:
        sorted:
          type: boolean
      unsorted:
        type: boolean
    PagedResponse_pageable:
      type: object
      properties:
        sort:
          $ref: '#/components/schemas/PagedResponse_pageable_sort'
        offset:
          type: integer
          format: int32
        pageNumber:
          type: integer
          format: int32
        pageSize:
          type: integer
          format: int32
        paged:
          type: boolean
        unpaged:
          type: boolean
  parameters:
    PageNumberParameter:
      name: pageNumber
      in: query
      description: Page Number
      required: false
      style: form
      explode: true
      schema:
        type: integer
        format: int32
        default: 1
    PageSizeParameter:
      name: pageSize
      in: query
      description: Page Size
      required: false
      style: form
      explode: true
      schema:
        type: integer
        format: int32
        default: 25
```

Describing path/url parameters:

```
  /v1/customer/{customerId}:
    get:
      parameters:
      - name: customerId
        in: path
        required: true
        description: Customer Id
        schema:
          type: string
          format: uuid
      responses:
        '200':
          description: Found Customer
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Customer'
```

Required has to be true of path parameter.

OpenAPI requests:

See how to define rest api methods. We will look out request body, response body, response code, etc.

OpenAPI operation summary and description:

Along with the parameters, responses operation object takes up summary and description to specify what the endpoint actually does (overview).

```
get:
        summary: List of Customer
        description: returns a list of customer objects in the application
```

In description we can have markdown (html like text where we can bold, italics, multi-line description etc. like readme of GitHub)

Writing the text in within single Asterix -> italics and double -> bold

OpenAPI tags:
Help group the operations within the documentation. By default it names default and puts everything there onwards.

So operation object takes a property tags and it takes up list of string

```
get:
        summary: List of Customer
        description: returns a list of customer objects in the application
        tags:
                - Customer
```

OpenAPI operationId:
It's a unique string used to identify the operation. This id must be unique among all the operations described in the API. It is case-sensitive.
It will be used by code generation tools to identify it's a unique operation.

```
get:
        summary: List of Customer
        description: returns a list of customer objects in the application
        tags:
                - Customer
        operationId: listCustomersV1
```

Using requestBody in operation object:

```
paths:
  /v1/customers:
    get:
      tags:
      - Customers
      summary: List Customers
      description: Get a list of customers in the system
      operationId: listCustomersV1
      parameters:
      - name: pageNumber
        in: query
        description: Page Number
        required: false
        style: form
        explode: true
        schema:
          type: integer
          format: int32
          default: 1
      - name: pageSize
        in: query
        description: Page Size
        required: false
        style: form
        explode: true
        schema:
          type: integer
          format: int32
          default: 25
      responses:
        200:
          description: List of Customers
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/CustomerPagedList'
    post:
      tags:
      - Customers
      summary: New Customer
      description: Create a new customer
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Customer'
        required: true
      responses:
        201:
          description: Customer Created
```

In the context of the OpenAPI Specification (OAS), the `explode` property in the parameters object is used specifically for query parameters that accept arrays or objects as values. It controls how array or object values are serialized into the URL query string.

When `explode` is set to `true`, each value of the array or object is serialized into a separate parameter with the same name. This means that the parameter name will appear multiple times in the URL query string, once for each value. This is the default behavior when `explode` is not explicitly specified.

When `explode` is set to `false`, the array or object is serialized into a single parameter, and its values are combined in some way, typically by using a delimiter such as a comma for arrays or an ampersand for objects.

Here's an example to illustrate the difference:

```yaml
parameters:
  - name: colors
    in: query
    schema:
      type: array
      items:
        type: string
    explode: true
```

With `explode: true`, if the array `colors` contains `["red", "blue", "green"]`, the URL query string would look like this:

```
?colors=red&colors=blue&colors=green
```

On the other hand, if `explode: false`, the URL query string might look like this:

```
?colors=red,blue,green
```

The choice between `explode: true` and `explode: false` depends on how you want array or object values to be serialized in your API requests.

But now we have to tell the user where we can find the customer created (that is the location), we can do by setting this in headers property of responses object of operation object.

```
post:
      tags:
      - Customers
      summary: New Customer
      description: Create a new customer
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Customer'
        required: true
      responses:
        201:
          description: Customer Created
          headers:
            Location:
              description: Location of the created customer
              style: simple
              explode: false
```

```
       schema:
         type: string
         format: uri
         example: http://example.com/v1/customers/{assignedIdValue}
```

The purpose of {assignedIdValue} is to indicate that this part of the URI will be dynamically replaced with the actual ID value assigned to the newly created customer. For example, if the ID value assigned to the newly created customer is 123, then {assignedIdValue} in the URI http://example.com/v1/customers/{assignedIdValue} would be replaced with 123, resulting in http://example.com/v1/customers/123.

Headers take a map so we can add additional properties as sub objects in them.

Now there is one more problem we have provided schema for requestBody content type as Customer which also mentions the id but this is system generated. We have to take care of that.

We can make that id property in component/schemas/Customer as readOnly: true

This makes it not to be passed as part of request. It's gonna be provided only in the response body.

Put request:

```
put:
   tags:
   - Customers
   summary: Update Customer
   description: Update customer by id.
   parameters:
   - name: customerId
     in: path
     description: Customer Id
     required: true
     style: simple
     explode: false
     schema:
       type: string
       format: uuid
   requestBody:
     content:
       application/json:
         schema:
           $ref: '#/components/schemas/Customer'
```

```
    required: true
  responses:
    204:
      description: Customer Updated
```

We can also append multiple responses

```
400:
      description: Bad Request
409:
      description: Conflict
```

Like this under responses object for post method

```
404:
      description: Not found
```

For get method

```
responses:
    204:
      description: Customer Updated
    400:
      description: Bad Request
    409:
      description: Conflict
```

For put method

```
delete:
  tags:
  - Customers
  summary: Delete Customer By ID
  description: Delete a customer by its Id value.
  operationId: deleteCustomerV1
  parameters:
  - name: customerId
    in: path
    description: Customer Id
    required: true
    style: simple
    explode: false
    schema:
      type: string
      format: uuid
  responses:
    200:
      description: Customer Delete
    404:
      description: Not found
```

For delete method

OpenAPI callbacks

https://swagger.io/docs/specification/callbacks/

In the age of programmable web, we are seeing more often where we are getting communications setup with web hooks or also known as callbacks in the OpenAPI.
So what we are saying is that when we use this api we can say i wanna receive a response back from a specific hour to a specific url.
We are gonna do the post to the url using the request body and expect response accordingly.

```
  /v1/customers/{customerId}/orders:
    post:
      tags:
      - Order Service
      description: Place Order
      parameters:
      - name: customerId
        in: path
        description: Customer Id
        required: true
        style: simple
        explode: false
        schema:
          type: string
          format: uuid
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/BeerOrder'
        required: false
      responses:
        201:
          description: Order Created
          headers:
            Location:
              description: Reference to created Order
              style: simple
            explode: false
            schema:
              type: string
              format: uri
        400:
          description: Bad Reqeust
        404:
          description: Not Found
        409:
          description: Conflict
      callbacks:
        orderStatusChange:
          ${request.body#/orderStatusCallbackUrl}:
            description: Webhook for order status change notifications
            post:
              requestBody:
                content:
                  application/json:
                    schema:
                      type: object
                      properties:
                        orderId:
                          type: string
                          format: uuid
                        orderStatus:
                          type: string
              responses:
                200:
                  description: Okay
```

BeerOrder looks like

```
BeerOrder:
      required:
      - customerId
      type: object
      properties:
        id:
          type: string
          format: uuid
          nullable: true
          readOnly: true
        customerId:
          type: string
          format: uuid
        customerRef:
```

```
        type: string
        nullable: true
      beerOrderLines:
        type: array
        items:
          $ref: '#/components/schemas/BeerOrderLine'
      orderStatusCallbackUrl:
        type: string
        format: uri
```

Upon order status change we gonna call back to url specified in the request body of the orderStatusCallbackUrl.

Full configuration:

```
openapi: 3.0.2
info:
  title: OpenAPI Course
  description: Specification for OpenAPI Course
  termsOfService: http://example.com/terms/
  contact:
    name: John Thompson
    url: https://springframework.guru
    email: john@springframework.guru
  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0.html
  version: "1.0"
servers:
- url: https://dev.example.com
  description: Development Server
- url: https://qa.example.com
  description: QA Server
- url: https://prod.example.com
  description: Production Server
paths:
  /v1/customers:
    get:
      tags:
      - Customers
      summary: List Customers
      description: Get a list of customers in the system
      operationId: listCustomersV1
      parameters:
      - name: pageNumber
        in: query
```

```yaml
    description: Page Number
    required: false
    style: form
    explode: true
    schema:
      type: integer
      format: int32
      default: 1
  - name: pageSize
    in: query
    description: Page Size
    required: false
    style: form
    explode: true
    schema:
      type: integer
      format: int32
      default: 25
  responses:
    200:
      description: List of Customers
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/CustomerPagedList'
post:
  tags:
  - Customers
  summary: New Customer
  description: Create a new customer
  requestBody:
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Customer'
    required: true
  responses:
    201:
      description: Customer Created
      headers:
        Location:
          description: Location of the created customer
          style: simple
          explode: false
          schema:
            type: string
            format: uri
```

```
       example: http://example.com/v1/customers/{assignedIdValue}
   400:
    description: Bad Request
   409:
    description: Conflict
/v1/customers/{customerId}:
 get:
  tags:
  - Customers
  summary: Get Customer By ID
  description: Get a single **Customer** by its Id value.
  operationId: getCustomerByIdV1
  parameters:
  - name: customerId
   in: path
   description: Customer Id
   required: true
   style: simple
   explode: false
   schema:
    type: string
    format: uuid
  responses:
   200:
    description: Found Customer
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/Customer'
   404:
    description: Not found
 put:
  tags:
  - Customers
  summary: Update Customer
  description: Update customer by id.
  parameters:
  - name: customerId
   in: path
   description: Customer Id
   required: true
   style: simple
   explode: false
   schema:
    type: string
    format: uuid
  requestBody:
```

```yaml
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Customer'
        required: true
      responses:
        204:
          description: Customer Updated
        400:
          description: Bad Request
        409:
          description: Conflict
    delete:
      tags:
      - Customers
      summary: Delete Customer By ID
      description: Delete a customer by its Id value.
      operationId: deleteCustomerV1
      parameters:
      - name: customerId
        in: path
        description: Customer Id
        required: true
        style: simple
        explode: false
        schema:
          type: string
          format: uuid
      responses:
        200:
          description: Customer Delete
        404:
          description: Not found
  /v1/customers/{customerId}/orders:
    post:
      tags:
      - Order Service
      description: Place Order
      parameters:
      - name: customerId
        in: path
        description: Customer Id
        required: true
        style: simple
        explode: false
        schema:
          type: string
          format: uuid
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/BeerOrder'
        required: false
      responses:
        201:
          description: Order Created
          headers:
            Location:
              description: Reference to created Order
              style: simple
              explode: false
              schema:
                type: string
                format: uri
        400:
          description: Bad Reqeust
        404:
          description: Not Found
        409:
          description: Conflict
      callbacks:
        orderStatusChange:
          ${request.body#/orderStatusCallbackUrl}:
            description: Webhook for order status change notifications
            post:
              requestBody:
                content:
                  application/json:
                    schema:
                      type: object
                      properties:
                        orderId:
                          type: string
                          format: uuid
                        orderStatus:
                          type: string
              responses:
                200:
                  description: Okay
  /v1/beers:
    get:
      tags:
      - Beers
      summary: List Beers
      description: List all beers in system.
      operationId: listBeersV1
      parameters:
      - name: pageNumber
        in: query
        description: Page Number
        required: false
        style: form
        explode: true
        schema:
          type: integer
          format: int32
          default: 1
      - name: pageSize
        in: query
        description: Page Size
        required: false
        style: form
        explode: true
        schema:
          type: integer
          format: int32
          default: 25
      responses:
        200:
          description: List of Beers
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/BeerPagedList'
        404:
          description: No Beers Found
    post:
      tags:
      - Beers
      summary: New Beer
      description: Create a new Beer Object
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Beer'
        required: true
      responses:
        201:
          description: Beer Created
```

```yaml
          headers:
            Location:
              description: Beer Object created
              style: simple
              explode: false
              schema:
                type: string
                format: uri
                example: http://example.com/v1/beers/{assignedIdValue}
        400:
          description: Bad Request
        409:
          description: Conflict
  /v1/beers/{beerId}:
    get:
      tags:
      - Beers
      summary: Get Beer by ID
      description: Get a single beer by its ID value.
      operationId: getBeerByIdV1
      parameters:
      - name: beerId
        in: path
        description: Beer Id
        required: true
        style: simple
        explode: false
        schema:
          type: string
          format: uuid
      responses:
        200:
          description: Found Beer by Id
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Beer'
        404:
          description: Not Found
    put:
      tags:
      - Beers
      summary: Update Beer by ID
      description: Update a beer by its ID value.
      operationId: updateBeerByIdV1
      parameters:
      - name: beerId
```

```yaml
        in: path
        description: Beer Id
        required: true
        style: simple
        explode: false
        schema:
          type: string
          format: uuid
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Beer'
        required: true
      responses:
        204:
          description: Beer Updated
        400:
          description: Bad Request
        404:
          description: Not Found
        409:
          description: Conflict
    delete:
      tags:
      - Beers
      summary: Delete Beer by Id
      description: Delete a beer resource by its ID value.
      operationId: deleteBeerV1
      parameters:
      - name: beerId
        in: path
        description: Beer Id
        required: true
        style: simple
        explode: false
        schema:
          type: string
          format: uuid
      responses:
        200:
          description: Beer Resource Deleted
        404:
          description: Not Found
components:
  schemas:
    Address:
```

```yaml
    type: object
    properties:
      line1:
        type: string
        example: 123 main
      city:
        type: string
        example: St Pete
      stateCode:
        maxLength: 2
        minLength: 2
        type: string
        description: 2 Letter State Code
        enum:
        - AL
        - AK
        - AZ
        - AR
        - CA
      zipCode:
        type: string
        example: "33701"
  Customer:
    type: object
    properties:
      id:
        type: string
        format: uuid
        readOnly: true
      firstName:
        maxLength: 100
        minLength: 2
        type: string
        example: John
      lastName:
        maxLength: 100
        minLength: 2
        type: string
        example: Thompson
      address:
        $ref: '#/components/schemas/Address'
    description: customer object
  CustomerList:
    maxItems: 100
    minItems: 1
    type: array
    description: List of Customers
  items:
    $ref: '#/components/schemas/Customer'
CustomerPagedList:
  type: object
  properties:
    content:
      $ref: '#/components/schemas/CustomerList'
  allOf:
  - $ref: '#/components/schemas/PagedResponse'
Brewery:
  type: object
  properties:
    name:
      type: string
    location:
      type: string
Beer:
  type: object
  properties:
    id:
      type: string
      format: uuid
      readOnly: true
    beerName:
      type: string
    style:
      type: string
      enum:
      - ALE
      - PALE_ALE
      - IPA
      - WHEAT
      - LAGER
    price:
      type: number
      format: float
    quantityOnHand:
      type: integer
      format: int32
    brewery:
      $ref: '#/components/schemas/Brewery'
  description: Beer Object
BeerList:
  type: array
  items:
    $ref: '#/components/schemas/Beer'
BeerPagedList:
  type: object
  properties:
    content:
      $ref: '#/components/schemas/BeerList'
  allOf:
  - $ref: '#/components/schemas/PagedResponse'
BeerOrder:
  required:
  - customerId
  type: object
  properties:
    id:
      type: string
      format: uuid
      nullable: true
      readOnly: true
    customerId:
      type: string
      format: uuid
    customerRef:
      type: string
      nullable: true
    beerOrderLines:
      type: array
      items:
        $ref: '#/components/schemas/BeerOrderLine'
    orderStatusCallbackUrl:
      type: string
      format: uri
BeerOrderLine:
  required:
  - orderQuantity
  - upc
  type: object
  properties:
    id:
      type: string
      format: uuid
      nullable: true
      readOnly: true
    beerId:
      type: string
      format: uuid
      readOnly: true
    upc:
      type: string
    orderQuantity:
```

```yaml
      maximum: 999
      minimum: 1
      type: integer
    quantityAllocated:
      type: integer
      nullable: true
      readOnly: true
PagedResponse:
  type: object
  properties:
    pageable:
      $ref: '#/components/schemas/PagedResponse_pageable'
    totalPages:
      type: integer
      format: int32
    last:
      type: boolean
    totalElements:
      type: integer
      format: int32
    size:
      type: integer
      format: int32
    number:
      type: integer
      format: int32
    numberOfElements:
      type: integer
      format: int32
    sort:
      $ref: '#/components/schemas/PagedResponse_pageable_sort'
    first:
      type: boolean
PagedResponse_pageable_sort:
  type: object
  properties:
    sorted:
      type: boolean
    unsorted:
      type: boolean
PagedResponse_pageable:
  type: object
  properties:
    sort:
      $ref: '#/components/schemas/PagedResponse_pageable_sort'
    offset:
      type: integer
```

```yaml
          format: int32
        pageNumber:
          type: integer
          format: int32
        pageSize:
          type: integer
          format: int32
        paged:
          type: boolean
        unpaged:
          type: boolean
  parameters:
    PageNumberParam:
      name: pageNumber
      in: query
      description: Page Number
      required: false
      style: form
      explode: true
      schema:
        type: integer
        format: int32
        default: 1
    PageSizeParam:
      name: pageSize
      in: query
      description: Page Size
      required: false
      style: form
      explode: true
      schema:
        type: integer
        format: int32
        default: 25
    CustomerIdPathParm:
      name: customerId
      in: path
      description: Customer Id
      required: true
      style: simple
      explode: false
      schema:
        type: string
        format: uuid
    BeerIdPathParm:
      name: beerId
      in: path
```

```
description: Beer Id
required: true
style: simple
explode: false
schema:
  type: string
  format: uuid
```

- This section defines callbacks, which are a way for the server to notify the client about events. Here, there's a callback named orderStatusChange, which specifies a webhook URL ($ {request.body#/orderStatusCallbackUrl}) to be notified when the order status changes. The webhook is triggered by a POST request with a JSON payload containing the orderId and orderStatus.

1. **Callbacks**:
    - This section introduces a callback mechanism named orderStatusChange, which is a way for the server to notify the client about changes in the order status asynchronously. It provides a description of the purpose of this callback and outlines the structure of the callback payload expected in the request body when the callback is triggered.

OpenAPI Security

https://swagger.io/docs/specification/authentication/

There is global and operation object level security, so defining the local level will override the global one.
At global level everything is authenticated and authorised.

We have to specify securityScheme using components.
After you have defined the security schemes in the securitySchemes section, you can apply them to the whole API or individual operations by adding the security section on the root level or operation level, respectively.

Basic Authentication:

```
components:
 securitySchemes:
     BasicAuth:
       type: http
       scheme: basic
```

We can apply globally:
security:
- BasicAuth: []

Only OpenId Connect and OAuth2 expects scope, for the rest we have to provide empty bracket.
So to access the endpoints now we have to prove the username and password.

JWT Bearer Token Auth:

```
components:
 securitySchemes:
     BasicAuth:
       type: http
       scheme: basic
     JwtAuthToken:
        type: http
        scheme: bearer
        bearerFormat: JWT
```

We can apply globally:

```
security:
- BasicAuth: []
- JwtAuthToken: []
```

Now we can authenticate either of them.

To setup no security for the endpoint:

We can have security: [] under operation object.

OpenAPI Code Gen

https://openapi-generator.tech/

With *50+* client generators, you can easily generate code to interact with any server which exposes an OpenAPI document.
Maintainers of APIs may also automatically generate and distribute clients as part of official SDKs.
Each client supports different options and features, but all templates can be replaced with your own Mustache-based templates.

Getting started with server development can be tough, especially if you're evaluating technologies. We can reduce the burden when you bring your own OpenAPI document.

Generate server stubs for 40+ different languages and technologies, including Java, Kotlin, Go, and PHP.
Some generators support *Inversion of Control*, allowing you to iterate on design via your OpenAPI document without worrying about blowing away your entire domain layer when you regenerate code.

To download the OpenAPI Generated Java Client
Click on export at right top corner of swagger hub and select client sdk and select java.

Similarly for server stubs can be dowloaded.