Swagger - OpenAPI

Editor, UI and Specification

https://www.openapis.org/
https://swagger.io

NOTE: We use swagger's specification to write our application's RESTFul services specification. Please don't get confused by the word specification. This document interchanges these 2 types of specification.

- Swagger/OpenAPI specification: Specifies how swagger's YAML file is written
- Our Services specification: Specifies how our application's RESTFul services will work

Swagger has variety of solutions to develop RESTFul webservice

- **Swagger specification** - Specification to **Design, Describe, and Document** RESTFul webservices in YAML or JSON
- **Swagger Editor** - Swagger specification file editor
- **Swagger UI** - Test RESTFul application using as defined in Swagger specification file editor

There are other Swagger tools available. Here is the complete list:
https://github.com/swagger-api

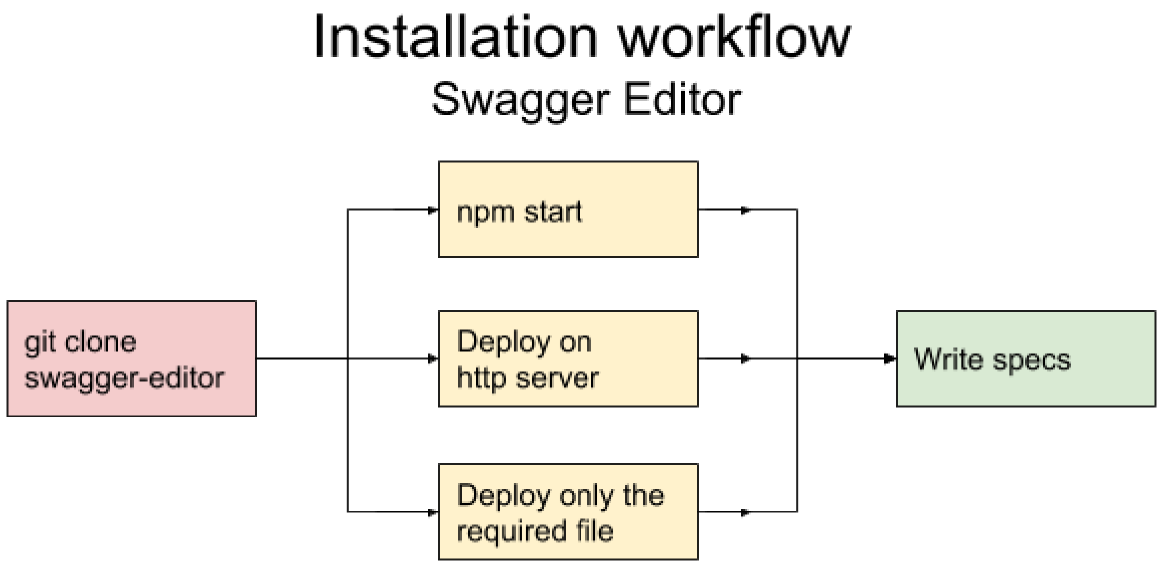
Swagger and Open API

https://www.openapis.org/
https://github.com/OAI/OpenAPI-Specification
https://swagger.io/introducing-the-open-api-initiative/
https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md
Swagger specification is adapted by Open API community (Linux Foundation) and named it Open API specification. Current and all future releases of Swagger Editor and UI will support Open API specification.

Swagger Editor

https://swagger.io/swagger-editor/
https://swagger.io/docs/swagger-tools/

Installation



Prerequisites

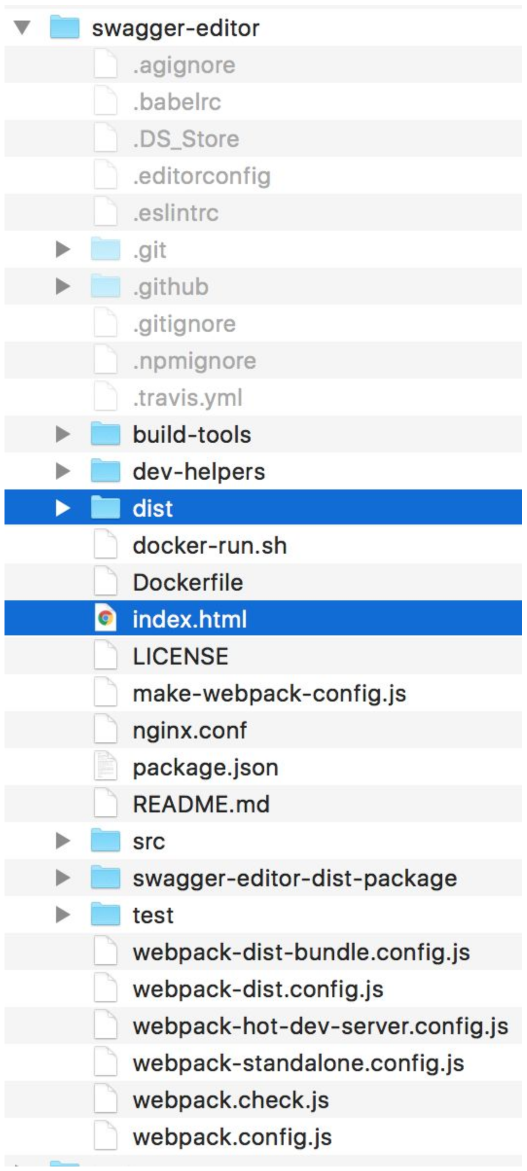
- Nodejs
- Any http server (e.g. npm http-server, tomcat, apache)

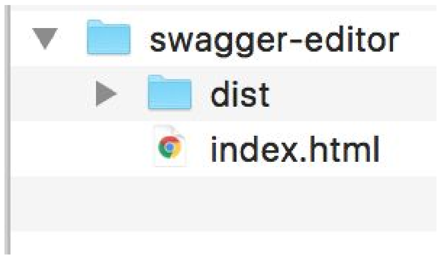
Clone Swagger Editor

$ git clone https://github.com/swagger-api/swagger-editor

Copy dist and index.html

Pull **dist** and **index.html** file in a separate folder.


```
swagger-editor
    .agignore
    .babelrc
    .DS_Store
    .editorconfig
    .eslintrc
    .git
    .github
    .gitignore
    .npmignore
    .travis.yml
    build-tools
    dev-helpers
    dist
    docker-run.sh
    Dockerfile
    index.html
    LICENSE
    make-webpack-config.js
    nginx.conf
    package.json
    README.md
    src
    swagger-editor-dist-package
    test
    webpack-dist-bundle.config.js
    webpack-dist.config.js
    webpack-hot-dev-server.config.js
    webpack-standalone.config.js
    webpack.check.js
    webpack.config.js
```
E.g. create a **swagger-editor** folder in another location and copy dist and index.html

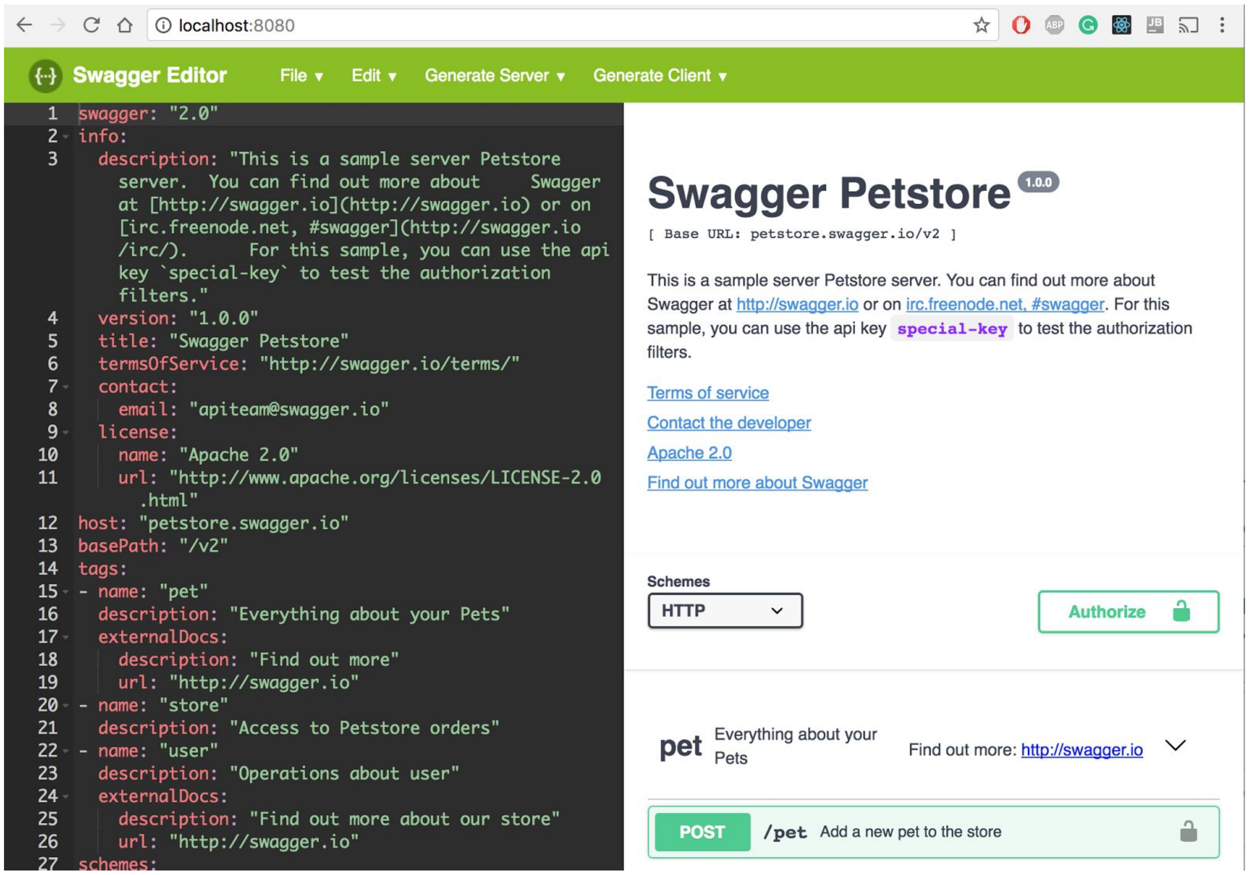

```
swagger-editor
    dist
    index.html
```

Install and run http-server

Install http-server
$ npm install http-server -g
Navigate to new swagger-editor folder and start http-server

$ hs .

Open http://localhost:8080 in browser



```
swagger: "2.0"
info:
  description: "This is a sample server Petstore
    server.  You can find out more about     Swagger
    at [http://swagger.io](http://swagger.io) or on
    [irc.freenode.net, #swagger](http://swagger.io
    /irc/).      For this sample, you can use the api
    key `special-key` to test the authorization
    filters."
  version: "1.0.0"
  title: "Swagger Petstore"
  termsOfService: "http://swagger.io/terms/"
  contact:
    email: "apiteam@swagger.io"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0
      .html"
host: "petstore.swagger.io"
basePath: "/v2"
tags:
- name: "pet"
  description: "Everything about your Pets"
  externalDocs:
    description: "Find out more"
    url: "http://swagger.io"
- name: "store"
  description: "Access to Petstore orders"
- name: "user"
  description: "Operations about user"
  externalDocs:
    description: "Find out more about our store"
    url: "http://swagger.io"
schemes:
```

Swagger Petstore 1.0.0

[Base URL: petstore.swagger.io/v2]

This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

Terms of service

Contact the developer

Apache 2.0

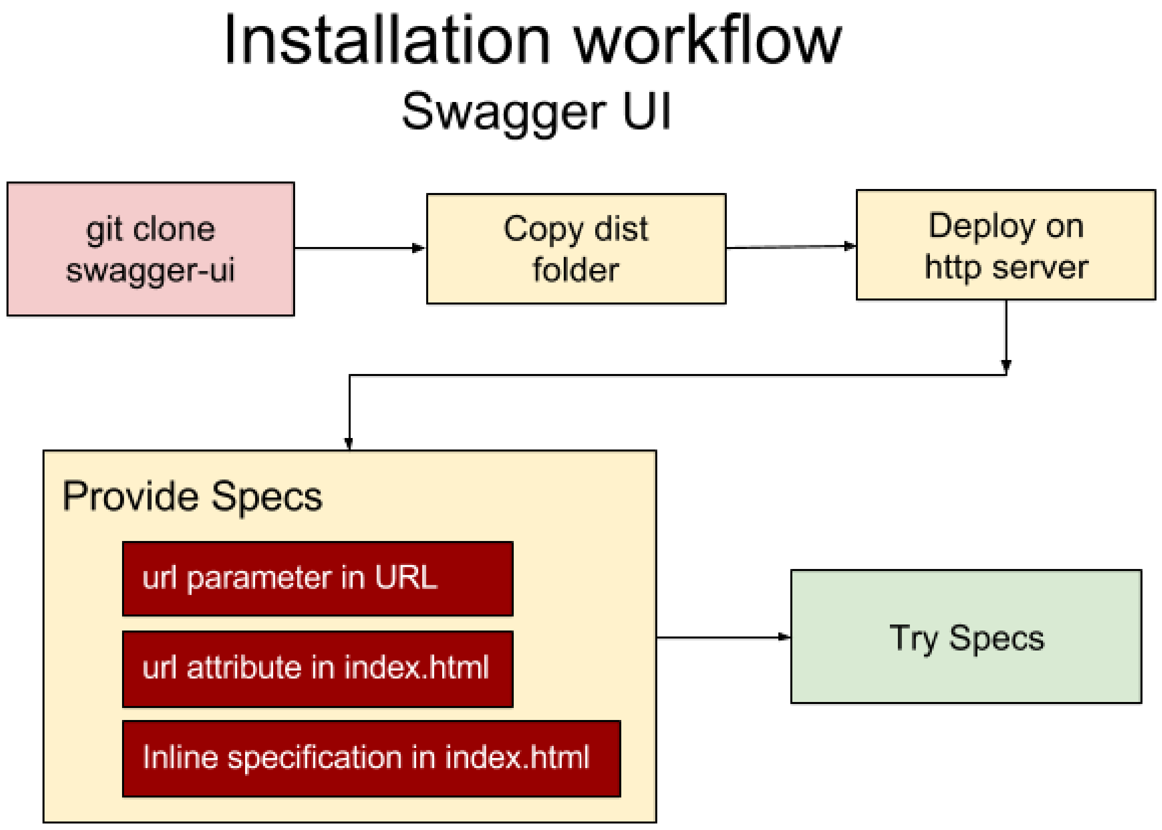
Find out more about Swagger

Schemes
HTTP
Authorize
pet Everything about your Pets
Find out more: http://swagger.io
POST /pet Add a new pet to the store

Swagger UI

https://swagger.io/swagger-ui/
https://swagger.io/download-swagger-ui/

Installation



Prerequisites

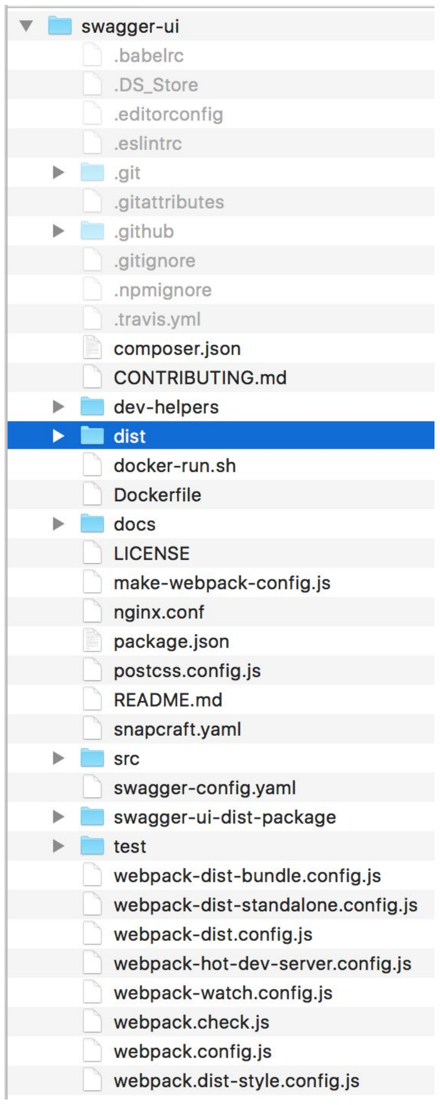
- Nodejs
- Any http server (e.g. npm http-server, tomcat, apache)

Clone Swagger UI

$ git clone https://github.com/swagger-api/swagger-ui.git

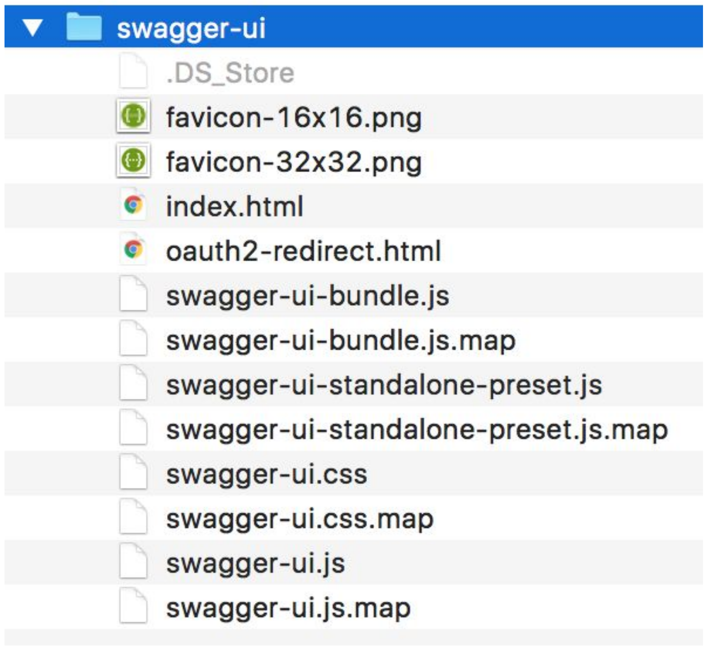
Copy dist content

Copy files in **dist** folder in a separate folder where you want to keep swagger-ui installation.



swagger-ui
.babelrc
.DS_Store
.editorconfig
.eslintrc
.git
.gitattributes
.github
.gitignore
.npmignore
.travis.yml
composer.json
CONTRIBUTING.md
dev-helpers
dist
docker-run.sh
Dockerfile
docs
LICENSE
make-webpack-config.js
nginx.conf
package.json
postcss.config.js
README.md
snapcraft.yaml
src
swagger-config.yaml
swagger-ui-dist-package
test
webpack-dist-bundle.config.js
webpack-dist-standalone.config.js
webpack-dist.config.js
webpack-hot-dev-server.config.js
webpack-watch.config.js
webpack.check.js
webpack.config.js
webpack.dist-style.config.js

E.g. create a **swagger-ui** folder in another location and copy dist and index.html


```
swagger-ui
    .DS_Store
    favicon-16x16.png
    favicon-32x32.png
    index.html
    oauth2-redirect.html
    swagger-ui-bundle.js
    swagger-ui-bundle.js.map
    swagger-ui-standalone-preset.js
    swagger-ui-standalone-preset.js.map
    swagger-ui.css
    swagger-ui.css.map
    swagger-ui.js
    swagger-ui.js.map
```

Install and run http-server

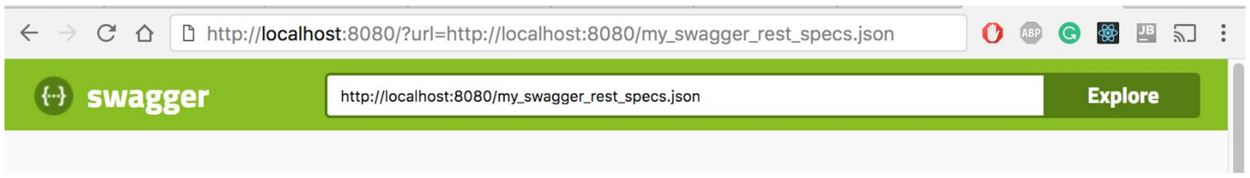
Install http-server
$ npm install http-server -g
Navigate to new swagger-ui folder and start http-server
$ hs .
Open http://localhost:8080 in browser

Note: To change the port, use -p flag
$ hs . -p 7070

Provide Specs

We can provide swagger specs in several ways:

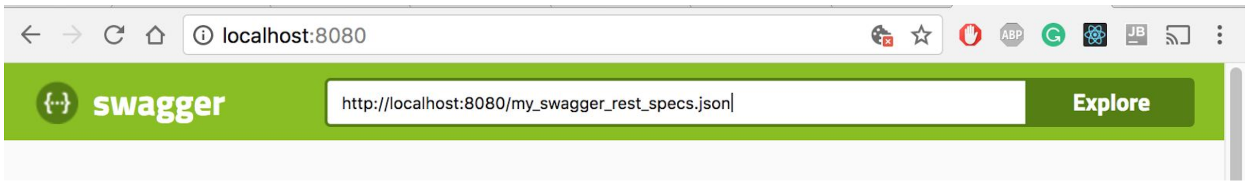
1. Pass specs URL in url parameter


http://localhost:8080/?url=http://localhost:8080/my_swagger_rest_specs.json

http://localhost:8080/my_swagger_rest_specs.json

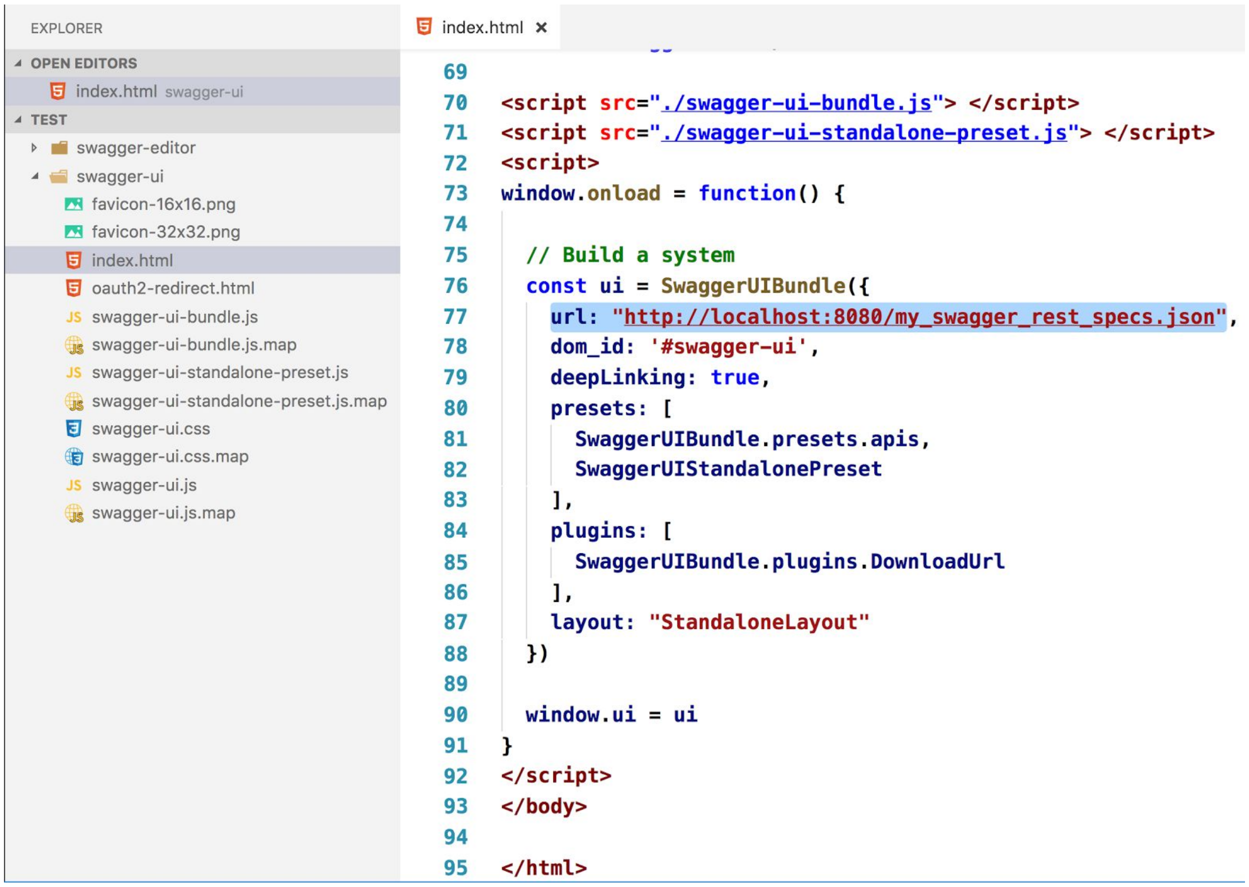
E.g.
http://localhost:8080?url=http://localhost:8080/my_swagger_rest_specs.json

2. Enter specs url in the text field and click **Explore** button



```
http://localhost:8080/my_swagger_rest_specs.json
```

3. Give Specs URL in index.html



```

<script src="./swagger-ui-bundle.js"> </script>
<script src="./swagger-ui-standalone-preset.js"> </script>
<script>
window.onload = function() {

  // Build a system
  const ui = SwaggerUIBundle({
    url: "http://localhost:8080/my_swagger_rest_specs.json",
    dom_id: '#swagger-ui',
    deepLinking: true,
    presets: [
      SwaggerUIBundle.presets.apis,
      SwaggerUIStandalonePreset
    ],
    plugins: [
      SwaggerUIBundle.plugins.DownloadUrl
    ],
    layout: "StandaloneLayout"
  })

  window.ui = ui
}
</script>
</body>

</html>
```

CORE issue

Look at how dealing with CORE in
https://swagger.io/docs/swagger-tools/#download-33

And to understand CORE
https://www.w3.org/TR/cors/

CommonMark

Swagger descriptions can be written in CommonMark
http://spec.commonmark.org/

NOTE: Go over this tutorial
http://commonmark.org/help/tutorial/

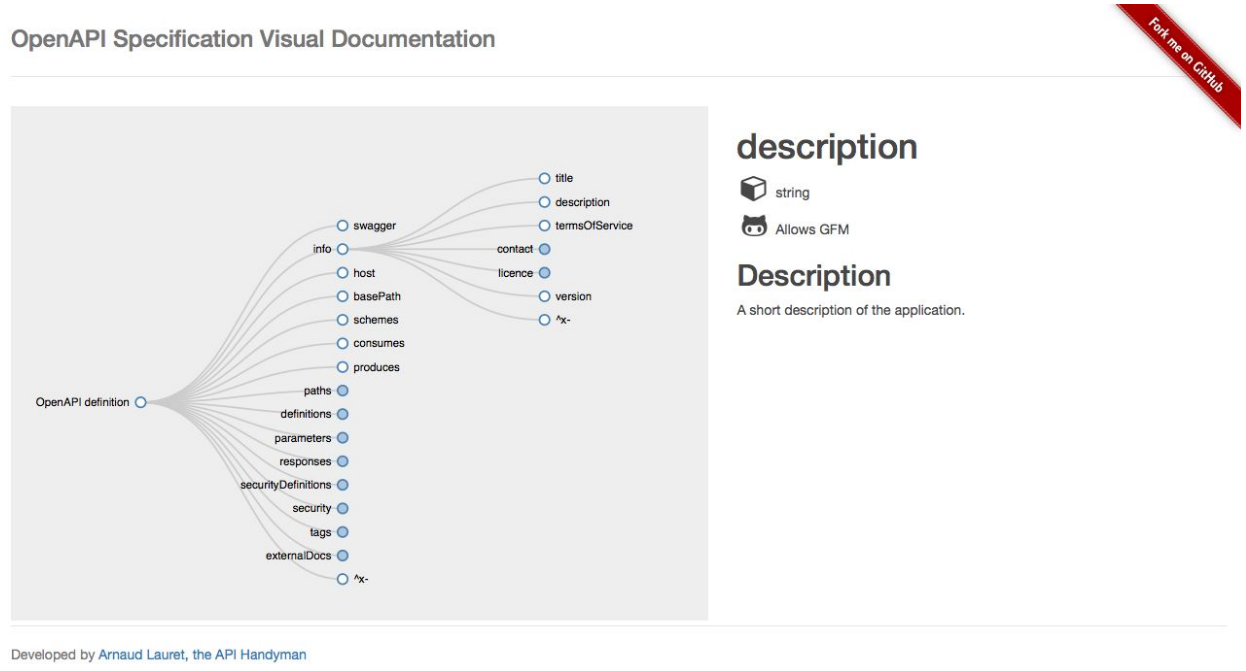
Swagger/OpenAPI Specification

https://swagger.io/specification/
https://swagger.io/docs/specification/
We can write our service specs in Swagger 2.0 or OpenAPI 3.0.0

All the future specs should be documented in OpenAPI 3.0.0
https://www.openapis.org/blog/2017/03/01/openapi-spec-3-implementers-draft-released

Swagger UI and Swagger Editor supports both Swagger 2.0 and OpenAPI 3.0.0

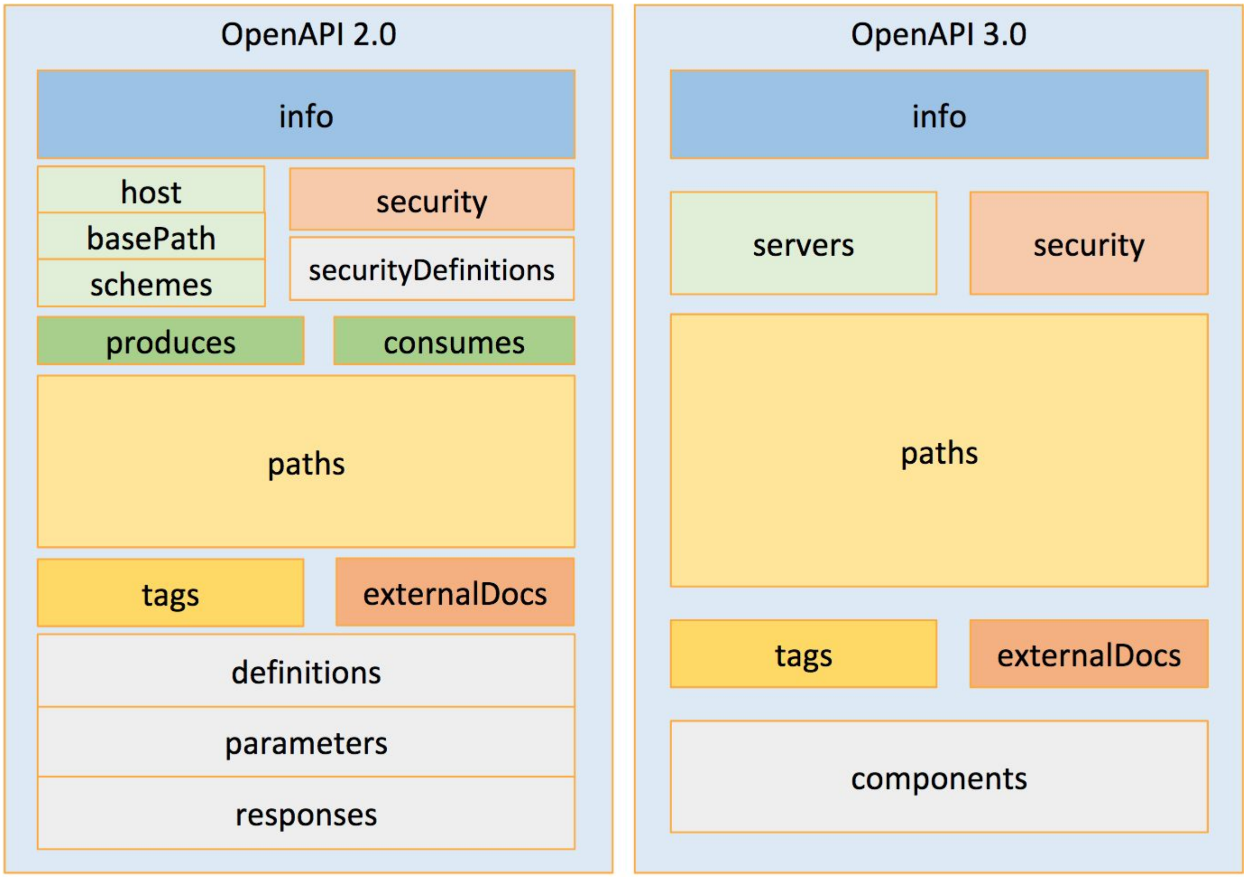
Below is Swagger 2.0 schema

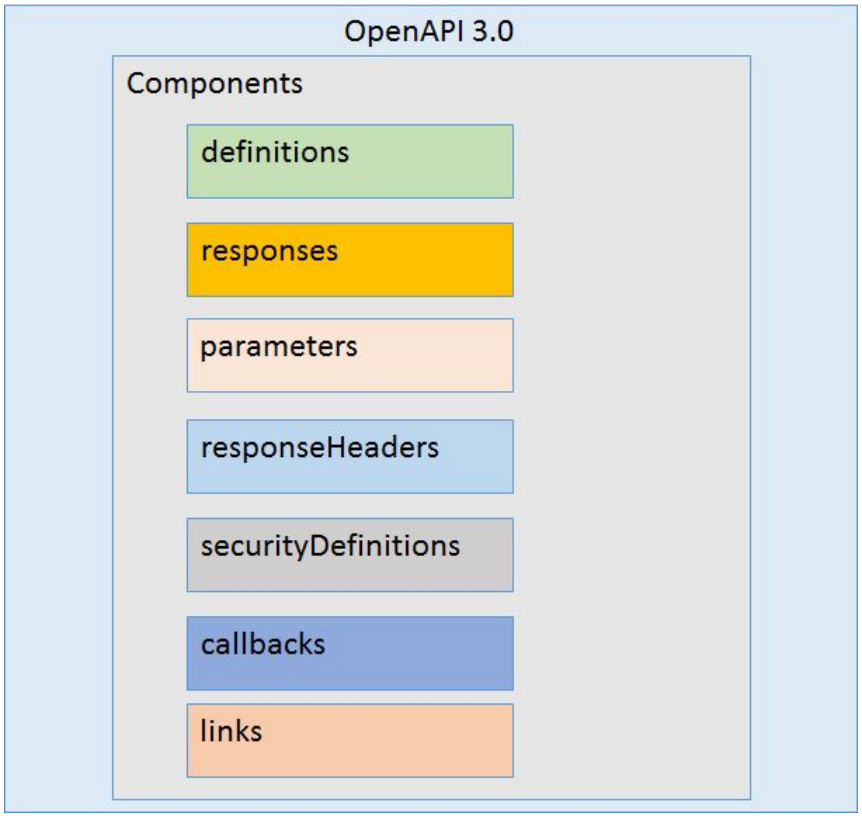


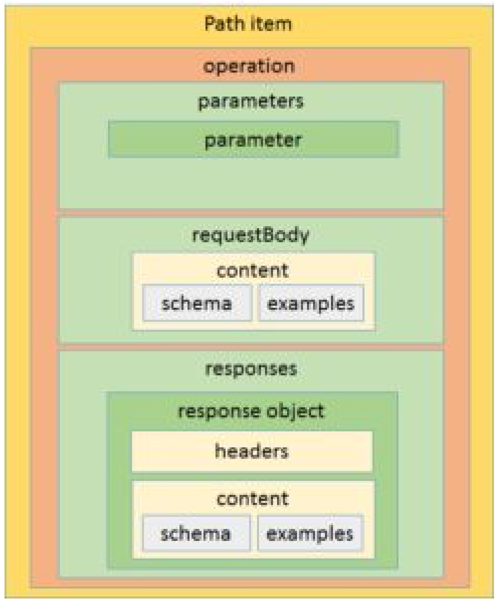
https://apihandyman.io/writing-openapi-swagger-specification-tutorial-part-1-introduction/
https://apihandyman.io/images/writing-openapi-swagger-specification-tutorial/openapi-specification-visual-documentation.png

Swagger 2.0 and OpenAPI 3.0.0 Comparision

https://www.openapis.org/specification/v3insights
https://blog.readme.io/an-example-filled-guide-to-swagger-3-2/


OpenAPI 2.0
info
host
basePath
schemes
security
securityDefinitions
produces
consumes
paths
tags
externalDocs
definitions
parameters
responses
OpenAPI 3.0
info
servers
security
paths
tags
externalDocs
components


OpenAPI 3.0
Components
definitions
responses
parameters
responseHeaders
securityDefinitions
callbacks
links



Minimum Configuration

At minimum 3 element are required. **openapi, info** and **path**
* **swagger or openapi:** contains version (swagger: '2.0' or openapi: '3.0.0')
* **info:** contains service metadata

- **title:**
- **version:**
- **description:**

servers: List of server
* **paths:** define service endpoints, request and response structure, methods (get, post, put, delete), response codes, links, callbacks
components: Define reusable objects that could be used in various parts of specification

```
openapi: 3.0.0
info:
 version: 1.0.0
 title: My API
 description: My API Description
paths: {}
```

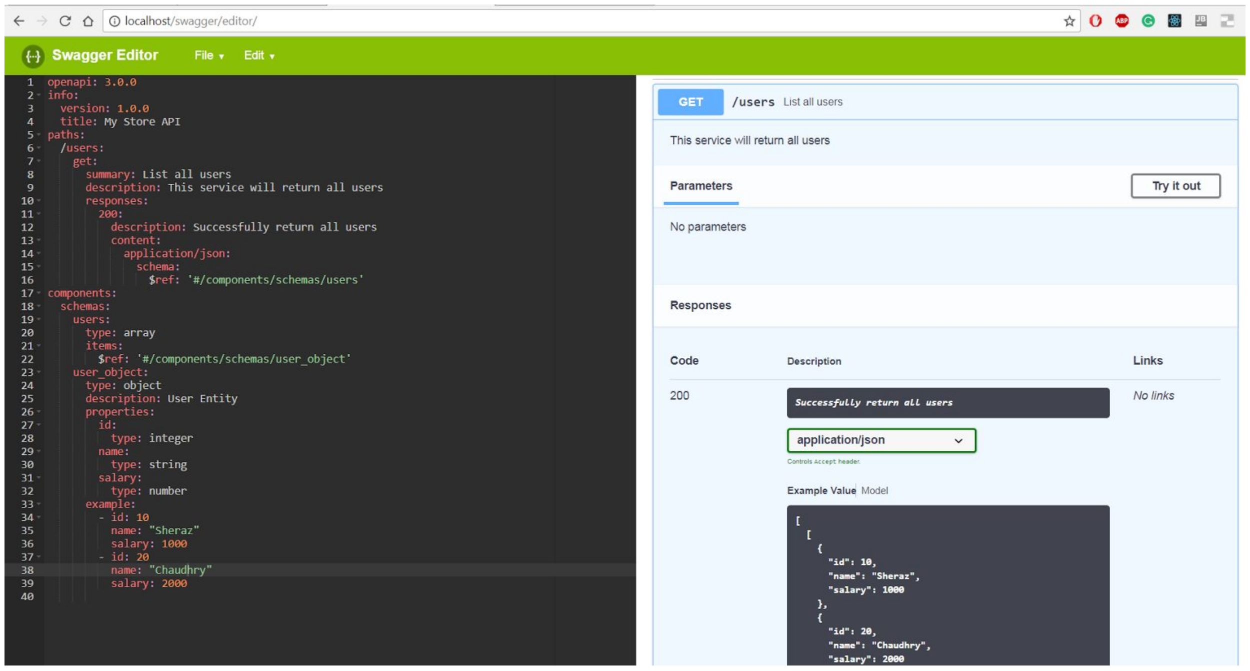
NOTE: {} object is given for **path:** because this is YAML rule.

Servers

paths - Service endpoints

All service endpoints are defined under **paths** element
E.g.
Let's define a /users service that

- On successful (200) GET will respond with an array of user object.
- Each user object have properties: id, name and salary
- And we also want to give example of user object



```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: My Store API
paths:
  /users:
    get:
      summary: List all users
      description: This service will return all users
      responses:
        200:
          description: Successfully return all users
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/users'
components:
  schemas:
    users:
      type: array
      items:
        $ref: '#/components/schemas/user_object'
    user_object:
      type: object
      description: User Entity
      properties:
        id:
          type: integer
        name:
          type: string
        salary:
          type: number
      example:
        - id: 10
          name: "Sheraz"
          salary: 1000
        - id: 20
          name: "Chaudhry"
          salary: 2000
```

This service will return all users

No parameters

Code	Description	Links
200	Successfully return all users	No links

```
[
  [
    {
      "id": 10,
      "name": "Sheraz",
      "salary": 1000
    },
    {
      "id": 20,
      "name": "Chaudhry",
      "salary": 2000
```

```yaml
openapi: 3.0.0
info:
 version: 1.0.0
 title: My Store API
paths:
 /users:
   get:
     summary: List all users
     description: This service will return all users
     responses:
       200:
         description: Successfully return all users
         content:
           application/json:
             schema:
               $ref: '#/components/schemas/users'
```

```yaml
components:
 schemas:
   users:
     type: array
     items:
       $ref: '#/components/schemas/user_object'
   user_object:
     type: object
     description: User Entity
     properties:
       id:
         type: integer
       name:
         type: string
       salary:
         type: number
     example:
       - id: 10
         name: "Sheraz"
         salary: 1000
       - id: 20
         name: "Chaudhry"
         salary: 2000
```

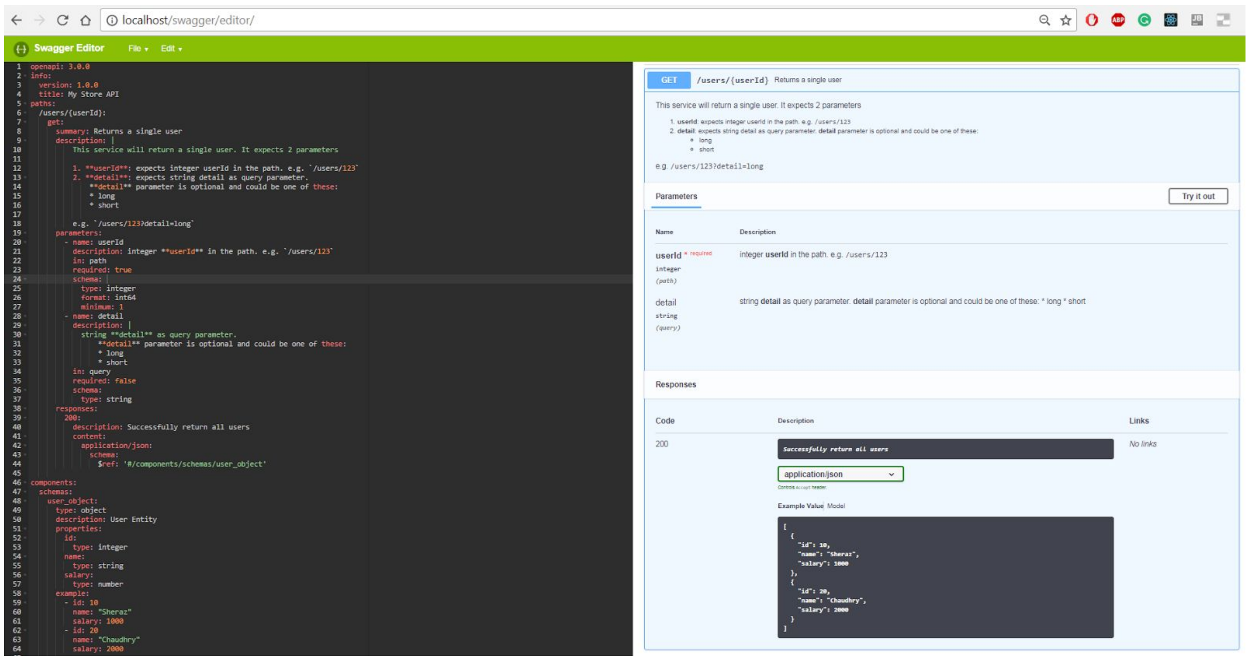
Parameters

Parameters are key value pair that RESTFul services can use. There are 4 types of parameters

1. **path:** values passed in path. e.g. /users/{userId}
2. **query:** values passed in query. e.g. /users?limit=10
3. **header:**
4. **cookie:**

Parameters are specified in **parameters** element under HTTP method.
E.g.
Let's define a /users/{userId} service that

- expects **userId** in path which is required.
- expects **detail** in query which is optional
- **detail** could be **long** or **short**
- Returns user object have properties: id, name and salary



```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: My Store API
paths:
  /users/{userId}:
    get:
      summary: Returns a single user
      description: |
          This service will return a single user. It expects 2 parameters

          1. **userId**: expects integer userId in the path. e.g. `/users/123`
          2. **detail**: expects string detail as query parameter.
              **detail** parameter is optional and could be one of these:
              * long
              * short

          e.g. `/users/123?detail=long`
      parameters:
        - name: userId
          description: integer **userId** in the path. e.g. `/users/123`
          in: path
          required: true
          schema:
            type: integer
            format: int64
            minimum: 1
        - name: detail
          description: |
            string **detail** as query parameter.
                **detail** parameter is optional and could be one of these:
                * long
                * short
          in: query
          required: false
          schema:
            type: string
      responses:
        200:
          description: Successfully return all users
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/user_object'

components:
  schemas:
    user_object:
      type: object
      description: User Entity
      properties:
        id:
          type: integer
        name:
          type: string
        salary:
          type: number
      example:
        - id: 10
          name: "Sheraz"
          salary: 1000
        - id: 20
          name: "Chaudhry"
          salary: 2000
```

GET `/users/{userId}` Returns a single user

This service will return a single user. It expects 2 parameters

1. **userId**: expects integer userId in the path. e.g. `/users/123`
2. **detail**: expects string detail as query parameter. **detail** parameter is optional and could be one of these:
   - long
   - short

e.g. `/users/123?detail=long`

Parameters

Try it out

Name	Description
userId * required integer *(path)*	integer **userId** in the path. e.g. `/users/123`
detail string *(query)*	string **detail** as query parameter. **detail** parameter is optional and could be one of these: * long * short

Responses

Code	Description	Links
200	Successfully return all users application/json Controls Accept header. Example Value Model	*No links*

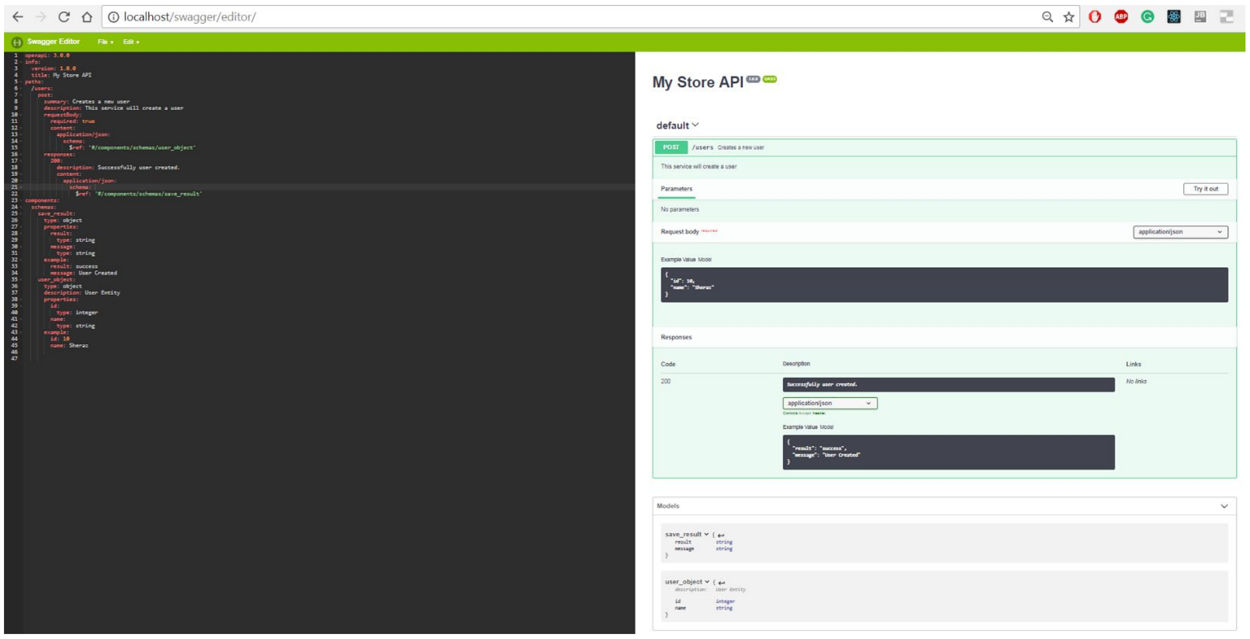
```
[
  {
    "id": 10,
    "name": "Sheraz",
    "salary": 1000
  },
  {
    "id": 20,
    "name": "Chaudhry",
    "salary": 2000
  }
]
```

```yaml
openapi: 3.0.0
info:
 version: 1.0.0
 title: My Store API
paths:
 /users/{userId}:
   get:
     summary: Returns a single user
     description: |
         This service will return a single user. It expects 2 parameters

         1. **userId**: expects integer userId in the path. e.g. `/users/123`
         2. **detail**: expects string detail as query parameter.
             **detail** parameter is optional and could be one of these:
             * long
             * short

         e.g. `/users/123?detail=long`
     parameters:
       - name: userId
         description: integer **userId** in the path. e.g. `/users/123`
         in: path
         required: true
         schema:
           type: integer
           format: int64
           minimum: 1
       - name: detail
         description: |
           string **detail** as query parameter.
               **detail** parameter is optional and could be one of
these:
               * long
               * short
         in: query
         required: false
         schema:
           type: string
     responses:
       200:
         description: Successfully return all users
         content:
           application/json:
             schema:
               $ref: '#/components/schemas/user_object'
components:
 schemas:
   user_object:
     type: object
     description: User Entity
     properties:
       id:
         type: integer
       name:
         type: string
       salary:
         type: number
     example:
       - id: 10
         name: "Sheraz"
         salary: 1000
       - id: 20
         name: "Chaudhry"
         salary: 2000
```

Request Body

To define a service that uses HTTP method (POST, and PUT) that sends request body, we use **requestBody** element.
E.g.
Create a /users service that:

- accepts user_object object, that contain id, and name
- returns save_result object, that contain result (success) and message
- give examples of both user_object and save_result



```
openapi: 3.0.0
info:
  version: 1.0.0
  title: My Store API
paths:
  /users:
    post:
      summary: Creates a new user
      description: This service will create a user
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/user_object'
      responses:
        200:
          description: Successfully user created.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/save_result'
components:
  schemas:
    save_result:
      type: object
      properties:
        result:
          type: string
        message:
          type: string
      example:
        result: success
        message: User Created
    user_object:
      type: object
      description: User Entity
      properties:
        id:
          type: integer
        name:
          type: string
      example:
        id: 10
        name: Sheraz
```

My Store API

default

POST /users Creates a new user

This service will create a user

Parameters   Try it out

No parameters

Request body required   application/json

Example Value Model

```
{
  "id": 10,
  "name": "Sheraz"
}
```

Responses

Code	Description	Links
200	Successfully user created.	No links

application/json

Controls Accept header.

Example Value Model

```
{
  "result": "success",
  "message": "User Created"
}
```

Models

```
save_result {
  result     string
  message    string
}

user_object {
  description:   User Entity
  id         integer
  name       string
}
```

```
openapi: 3.0.0
info:
 version: 1.0.0
 title: My Store API
paths:
 /users:
   post:
     summary: Creates a new user
     description: This service will create a user
     requestBody:
       required: true
       content:
         application/json:
           schema:
             $ref: '#/components/schemas/user_object'
     responses:
       200:
         description: Successfully user created.
         content:
           application/json:
             schema:
               $ref: '#/components/schemas/save_result'
components:
 schemas:
   save_result:
     type: object
     properties:
       result:
         type: string
       message:
         type: string
     example:
       result: success
       message: User Created
   user_object:
     type: object
     description: User Entity
     properties:
       id:
         type: integer
       name:
         type: string
     example:
       id: 10
       name: Sheraz
```

Reuseable Request Body

Reuseable request body can be created to use the same request body for multiple service endpoints.
e.g.
Create (POST) or update (PUT) a user.

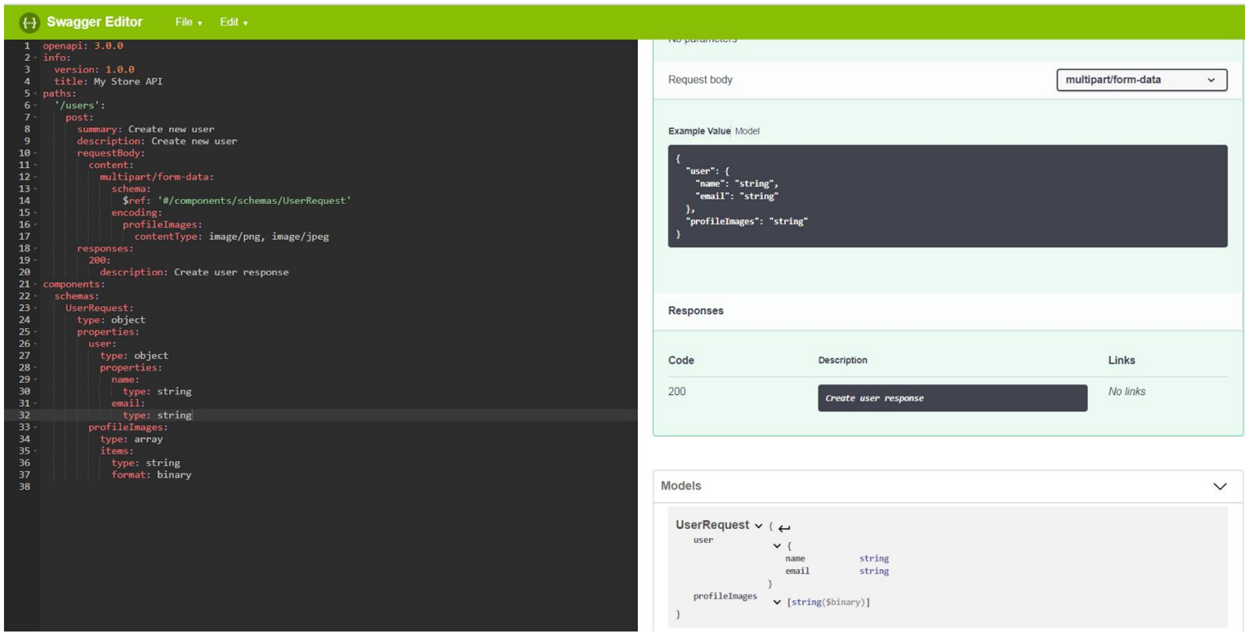
- Create endpoint /user and update endpoint /user/{userId}
- Both create a update user should use same request body
- update user should use accept userId in path
- Both should respond with a string message

Upload single file

Multipart Request Body

Multipart Request Body can be created to send multipart/form-data request body.
e.g.
Create (POST) a user on /users service

- Request should accept user object with name and email
- Request should accept profile images of type PNG and JPEG



```
openapi: 3.0.0
info:
  version: 1.0.0
  title: My Store API
paths:
  '/users':
    post:
      summary: Create new user
      description: Create new user
      requestBody:
        content:
          multipart/form-data:
            schema:
              $ref: '#/components/schemas/UserRequest'
            encoding:
              profileImages:
                contentType: image/png, image/jpeg
      responses:
        200:
          description: Create user response
components:
  schemas:
    UserRequest:
      type: object
      properties:
        user:
          type: object
          properties:
            name:
              type: string
            email:
              type: string
        profileImages:
          type: array
          items:
            type: string
            format: binary
```

Request body: multipart/form-data

Example Value Model

```
{
  "user": {
    "name": "string",
    "email": "string"
  },
  "profileImages": "string"
}
```

Responses

Code	Description	Links
200	Create user response	No links

Models

```
UserRequest {
  user            {
                    name      string
                    email     string
                  }
  profileImages   [string($binary)]
}
```

```
openapi: 3.0.0
info:
 version: 1.0.0
 title: My Store API
paths:
 '/users':
   post:
     summary: Create new user
     description: Create new user
     requestBody:
       content:
         multipart/form-data:
           schema:
             $ref: '#/components/schemas/UserRequest'
           encoding:
             profileImages:
              contentType: image/png, image/jpeg
     responses:
       200:
         description: Create user response
components:
 schemas:
   UserRequest:
     type: object
     properties:
       user:
         type: object
         properties:
           name:
             type: string
           email:
             type: string
       profileImages:
         type: array
         items:
           type: string
           format: binary
```

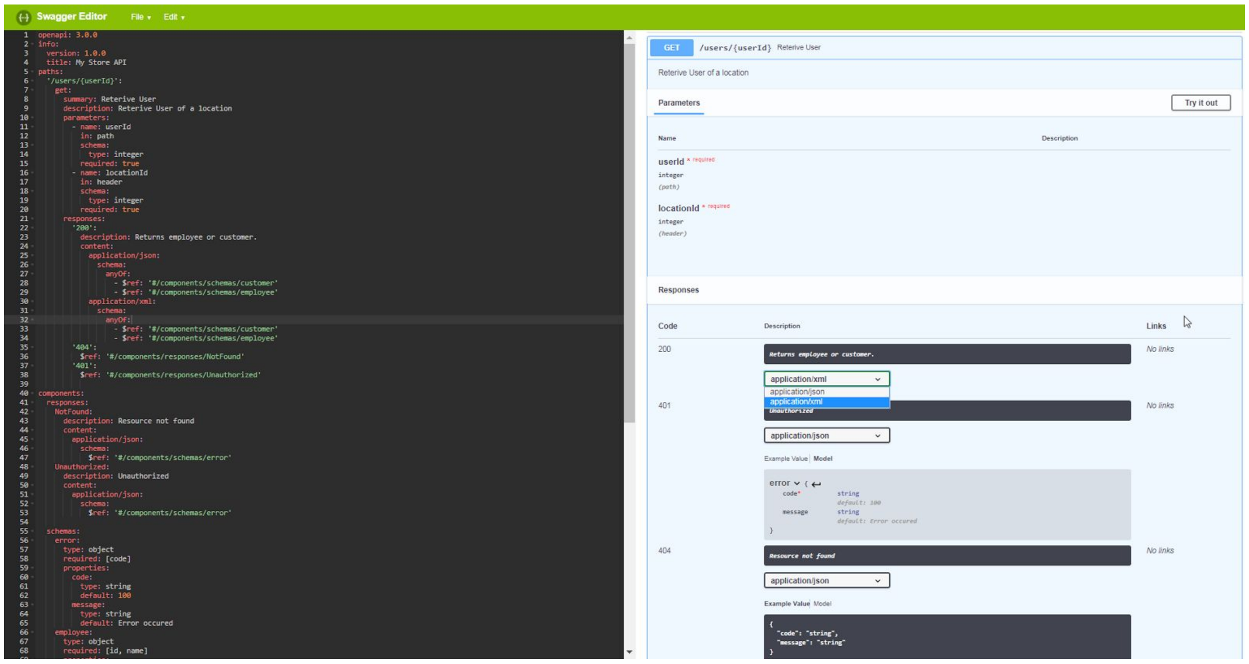
Responses

API response is defined under **responses** element. There are various response types, and parts of each response

- Response code. HTTP code. e.g. 200, 201, 404, 5XX...
- Response Body. Response body. Response content.
- Empty Response body
- Respond with a file
- Response Header

E.g.
Define a /users/{userId} API that

- Request should accept location ID in header
- Request should accept userId in path
- Response could be 200(success), 404(not found), 401(Unauthorized)
- Use 404 and 401 reuseable responses
- Use error message for both 404, and 401
- Error message should have code and message. Code and message should have default value.
- 200 could return customer or employee object
- 200 response could be JSON or XML



```
openapi: 3.0.0
info:
  version: 1.0.0
  title: My Store API
paths:
  '/users/{userId}':
    get:
      summary: Reterive User
      description: Reterive User of a location
      parameters:
        - name: userId
          in: path
          schema:
            type: integer
          required: true
        - name: locationId
          in: header
          schema:
            type: integer
          required: true
      responses:
        '200':
          description: Returns employee or customer.
          content:
            application/json:
              schema:
                anyOf:
                  - $ref: '#/components/schemas/customer'
                  - $ref: '#/components/schemas/employee'
            application/xml:
              schema:
                anyOf:
                  - $ref: '#/components/schemas/customer'
                  - $ref: '#/components/schemas/employee'
        '404':
          $ref: '#/components/responses/NotFound'
        '401':
          $ref: '#/components/responses/Unauthorized'

components:
  responses:
    NotFound:
      description: Resource not found
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/error'
    Unauthorized:
      description: Unauthorized
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/error'

  schemas:
    error:
      type: object
      required: [code]
      properties:
        code:
          type: string
          default: 100
        message:
          type: string
          default: Error occured
    employee:
      type: object
      required: [id, name]
```

GET `/users/{userId}` Reterive User

Reterive User of a location

Parameters

Name	Description
userId * required integer *(path)*	
locationId * required integer *(header)*	

Responses

Code	Description	Links
200	*Returns employee or customer.* application/xml application/json application/xml	*No links*
401	*Unauthorized* application/json Example Value | Model error { code* string default: 100 message string default: Error occured }	*No links*
404	*Resource not found* application/json Example Value | Model { "code": "string", "message": "string" }	*No links*

```
openapi: 3.0.0
info:
 version: 1.0.0
 title: My Store API
paths:
 '/users/{userId}':
   get:
     summary: Retrieve User
     description: Retrieve User of a location
     parameters:
       - name: userId
         in: path
         schema:
           type: integer
         required: true
       - name: locationId
         in: header
         schema:
           type: integer
         required: true
     responses:
       '200':
         description: Returns employee or customer.
         content:
           application/json:
             schema:
               anyOf:
```

```yaml
                  - $ref: '#/components/schemas/customer'
                  - $ref: '#/components/schemas/employee'
            application/xml:
              schema:
                anyOf:
                  - $ref: '#/components/schemas/customer'
                  - $ref: '#/components/schemas/employee'
        '404':
          $ref: '#/components/responses/NotFound'
        '401':
          $ref: '#/components/responses/Unauthorized'

components:
 responses:
   NotFound:
     description: Resource not found
     content:
       application/json:
         schema:
           $ref: '#/components/schemas/error'
   Unauthorized:
     description: Unauthorized
     content:
       application/json:
         schema:
           $ref: '#/components/schemas/error'
 schemas:
   error:
     type: object
     required: [code]
     properties:
       code:
         type: string
         default: 100
       message:
         type: string
         default: Error occurred
   employee:
     type: object
     required: [id, name]
     properties:
       id:
         type: integer
       name:
         type: string
        department:
          type: string
    customer:
      type: object
      required: [id, name]
      properties:
        id:
          type: integer
        name:
          type: string
        orders:
          type: array
          items:
            type: string
```

Data Models

https://swagger.io/docs/specification/data-models/

Authentication

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

Grouping

https://swagger.io/docs/specification/grouping-operations-with-tags/

Links

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

Callbacks

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

```
http-server . > http.log 2>&1 &
```