# **What Is OpenAPI?**

**OpenAPI Specification** (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI file allows you to describe your entire API, including:

* Available endpoints and operations on each endpoint
* Operation parameters Input and output for each operation
* Authentication methods
* Contact information, license, terms of use and other information.

API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines.

# **What Is Swagger?**

**Swagger** is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs. The major Swagger tools include:

* [Swagger Editor](http://editor.swagger.io/) – browser-based editor where you can write OpenAPI specs.
* [Swagger UI](https://swagger.io/swagger-ui/) – renders OpenAPI specs as interactive API documentation.
* [Swagger Codegen](https://github.com/swagger-api/swagger-codegen) – generates server stubs and client libraries from an OpenAPI spec.

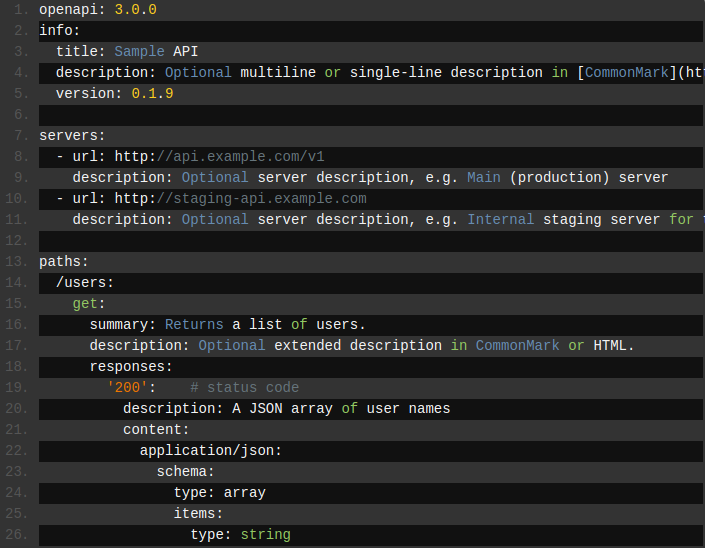
# **Why Use OpenAPI?**

The ability of APIs to describe their own structure is the root of all awesomeness in OpenAPI. Once written, an OpenAPI specification and Swagger tools can drive your API development further in various ways:

* Design-first users: use [Swagger Codegen](https://swagger.io/swagger-codegen/) to **generate a server stub** for your API. The only thing left is to implement the server logic – and your API is ready to go live!
* Use [Swagger Codegen](https://swagger.io/swagger-codegen/) to **generate client libraries** for your API in over 40 languages.
* Use [Swagger UI](https://swagger.io/swagger-ui/) to generate **interactive API documentation** that lets your users try out the API calls directly in the browser.
* Use the spec to connect API-related tools to your API. For example, import the spec to [SoapUI](https://soapui.org/) to create automated tests for your API.

# **Basic Structure**

We can write OpenAPI definitions in [YAML](https://en.wikipedia.org/wiki/YAML) or [JSON](https://en.wikipedia.org/wiki/JSON). In this guide, we use only YAML examples but JSON works equally well. A sample OpenAPI 3.0 definition written in YAML looks like:



All keyword names are **case-sensitive**.

# **Metadata**

Every Swagger specification starts with the Swagger version, 2.0 being the latest version. A Swagger version defines the overall structure of an API specification – what we can document and how we document it.

1. swagger: "2.0"

Then, we need to specify the API info – **title**, **description** (optional), **version** (API version, not file revision or Swagger version).

1. info:
2. title: Sample API
3. description: API description in Markdown.
4. version: 1.0.0

**version** can be a random string. We can use major.minor.patch (as in [semantic versioning](http://semver.org/)), or an arbitrary format like 1.0-beta or 2016.11.15. **description** can be [multiline](http://stackoverflow.com/a/21699210) and supports [GitHub Flavored Markdown](https://guides.github.com/features/mastering-markdown/) for rich text representation. **info** also supports other fields for contact information, license and other details.

# **Base URL**

The base URL for all API calls is defined using **schemes**, **host** and **basePath**:

1. host: api.example.com
2. basePath: /v1
3. schemes:
4. - https

# **Consumes, Produces**

The **consumes** and **produces** sections define the MIME types supported by the API. The root-level definition can be overridden in individual operations.

1. consumes:
2. - application/json
3. - application/xml
4. produces:
5. - application/json
6. - application/xml

All API paths are relative to the base URL. For example, /users actually means <https://api.example.com/v1/users>.

# **Paths**

The **paths** section defines individual endpoints (paths) in our API, and the HTTP methods (operations) supported by these endpoints. For example, GET /users can be described as:

1. paths:
2. /users:
3. get:
4. summary: Returns a list of users.
5. description: Optional extended description in Markdown.
6. produces:
7. - application/json
8. responses:
9. **200:**
10. description: OK

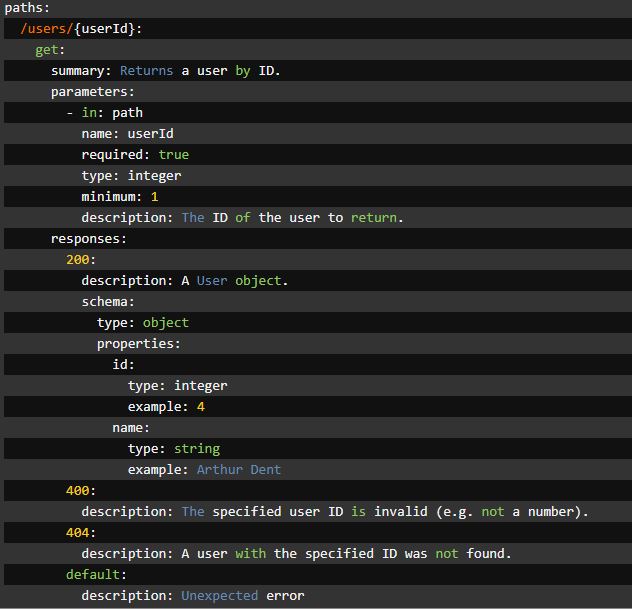
# **Parameters**

Operations can have parameters that can be passed via URL path (**/users/{userId}**), query string (**/users?role=admin**), headers (**X-CustomHeader: Value**) and request body. We can define the parameter types, format, whether they are required or optional, and other details:

1. paths:
2. /users/{userId}:
3. get:
4. summary: Returns a user by ID.
5. parameters:
6. - in: path
7. name: userId
8. required: true
9. type: integer
10. minimum: 1
11. description: Parameter description in Markdown.
12. responses:
13. 200:
14. description: OK

# **Responses**

For each operation, we can define possible status codes, such as 200 OK or 404 Not Found, and **schema** of the response body. Schemas can be defined inline or referenced from an external definition via **$ref**. You can also provide example responses for different content types.

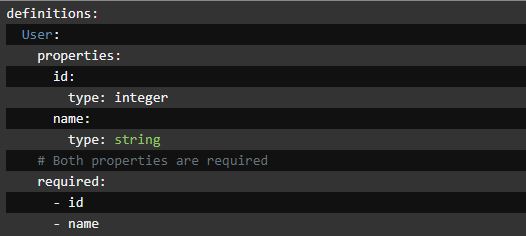


# **Input and Output Models**

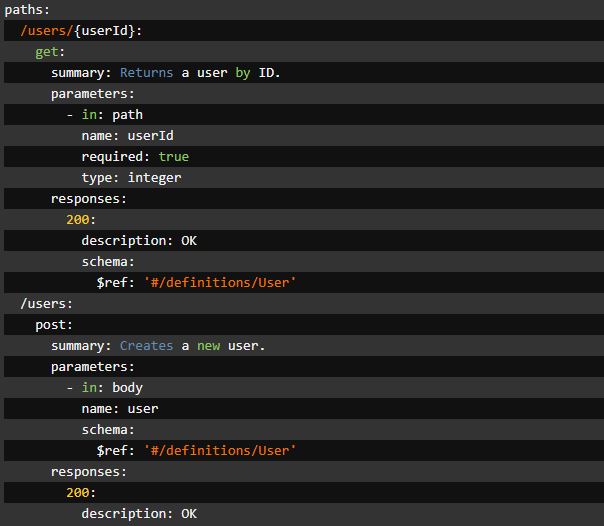
The global **definitions** section lets us define common data structures used in our API. They can be referenced via **$ref** whenever a **schema** is required – both for request body and response body. For example, this JSON object:



can be represented as:

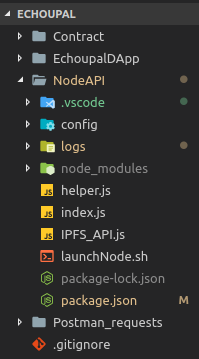


and then referenced in the request body schema and response body schema as follows:

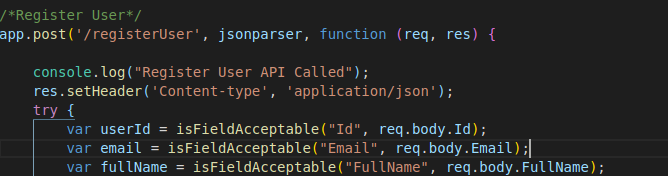
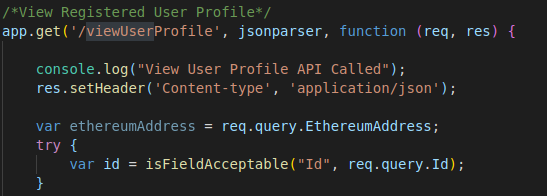


# **Creating API Documentation for Echoupal using Swagger**

## Clone the Echoupal project from Bitbucket

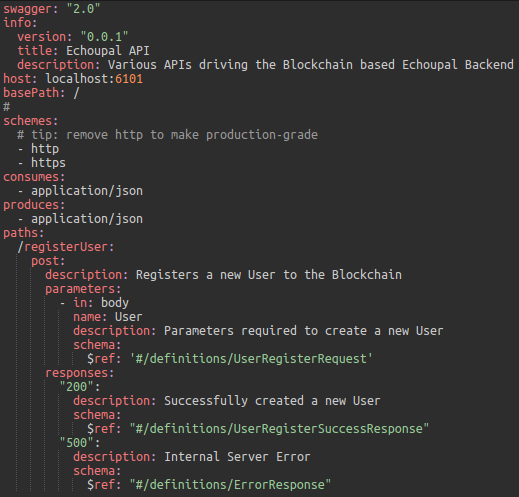


## From our existing API ‘**index.js’**, we select two routes (a POST and a GET) for further demonstration



## Generating Swagger Specification ‘swagger.yaml’ file for our selected APIS

1. Head over to : <https://editor.swagger.io/>
2. We will see a website which is split into 2 panes: **left** one is the **Editor** where we will create Swagger Specification for our selected APIs and the **right** one is the **live render** of our Swagger Specification.
3. A **sample** Swagger Specification will already be provided to us on the left hand side. We can modify the sample skeleton and add specification for our selected APIs the live render of which we will see on the right hand side.
4. Let’s start writing Swagger Specification for our **POST** API as follows:



1. Similarly, we can write Swagger Specification for our GET API.
2. Once done, our Swagger Specification should look like,

swagger: "2.0"

info:

version: "0.0.1"

title: Echoupal API

description: Various APIs driving the Blockchain based Echoupal Backend

host: localhost:6101

basePath: /

#

schemes:

# tip: remove http to make production-grade

- http

- https

consumes:

- application/json

produces:

- application/json

paths:

/registerUser:

post:

description: Registers a new User to the Blockchain

parameters:

- in: body

name: User

description: Parameters required to create a new User

schema:

$ref: '#/definitions/UserRegisterRequest'

responses:

"200":

description: Successfully created a new User

schema:

$ref: "#/definitions/UserRegisterSuccessResponse"

"500":

description: Internal Server Error

schema:

$ref: "#/definitions/ErrorResponse"

/viewUserProfile:

get:

description: Returns an User Profile based on the given ID

parameters:

- name: EthereumAddress

in: query

description: Ethereum Address of the User calling this API

required: false

type: string

- name: Id

in: query

description: ID for the Agent Profile to be fetched

required: true

type: string

responses:

"200":

description: Successfully returned a User Profile JSON

schema:

$ref: "#/definitions/ProfileResponse"

"500":

description: Internal Server Error

schema:

$ref: "#/definitions/ErrorResponse"

definitions:

UserRegisterRequest:

type: "object"

required:

- Id

- Email

- FullName

- PhoneNumber

- FirstName

- LastName

- UserName

- Address

- StatusID

- CreatedBy

- CreatedDate

- ModifiedBy

- ModifiedDate

- LocationID

- ParticipantType

properties:

Id:

type: "string"

description: "The ID of the User"

Email:

type: "string"

description: "The Email of the User"

FullName:

type: "string"

description: "The Full Name of the User"

PhoneNumber:

type: "string"

description: "The Phone Number of the User"

FirstName:

type: "string"

description: "The First Name of the User"

LastName:

type: "string"

description: "The Last Name of the User"

UserName:

type: "string"

description: "The User Name of the User"

Address:

type: "string"

description: "The Address of the User"

StatusID:

type: "string"

description: "The Status ID of the User"

CreatedBy:

type: "string"

description: "Who Created the User"

CreatedDate:

type: "string"

description: "When the User was Created"

ModifiedBy:

type: "string"

description: "Who Modified the User's account"

ModifiedDate:

type: "string"

description: "When the User's account was Modified"

LocationID:

type: "string"

description: "The Location ID of the User"

ParticipantType:

type: "number"

description: "The Participant Type of the User"

ProfileResponse:

type: "object"

properties:

Id:

type: "string"

description: "The ID of the User"

Email:

type: "string"

description: "The Email of the User"

FullName:

type: "string"

description: "The Full Name of the User"

PhoneNumber:

type: "string"

description: "The Phone Number of the User"

FirstName:

type: "string"

description: "The First Name of the User"

LastName:

type: "string"

description: "The Last Name of the User"

UserName:

type: "string"

description: "The User Name of the User"

Address:

type: "string"

description: "The Address of the User"

StatusID:

type: "number"

description: "The Status ID of the User"

CreatedBy:

type: "string"

description: "Who Created the User"

CreatedDate:

type: "string"

description: "When the User was Created"

ModifiedBy:

type: "string"

description: "Who Modified the User's account"

ModifiedDate:

type: "string"

description: "When the User's account was Modified"

LocationID:

type: "string"

description: "The Location ID of the User"

ParticipantType:

type: "string"

description: "The Participant Type of the User"

EthereumAddress:

type: "string"

description: "The Ethereum Address of the User"

UserRegisterSuccessResponse:

required:

- Success

- EthereumAddress

- RequestType

properties:

Success:

type: boolean

EthereumAddress:

type: string

RequestType:

type: string

ErrorResponse:

required:

- Success

- Message

properties:

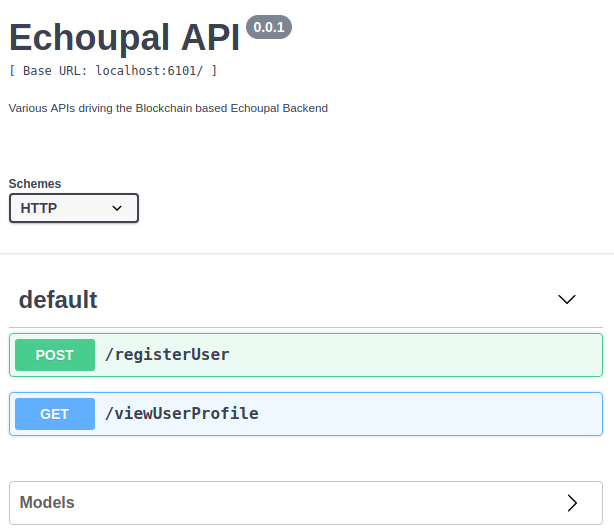
Success:

type: boolean

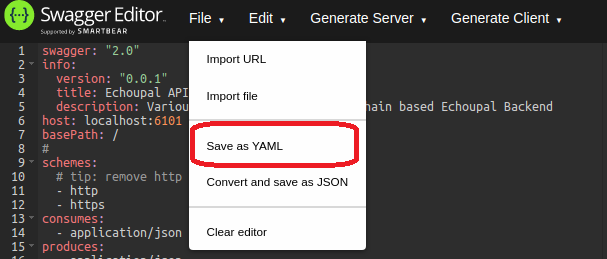
Message:

type: string

1. And the live render of our specification would look like:



1. Now click on the **File** menu and select ‘**Save as YAML’** option. This should download our Swagger Specification to the local storage.



## Creating Swagger UI for our Swagger Specification

* **Git clone or download** the following url:

<https://github.com/swagger-api/swagger-ui.git>

* Browse into the cloned folder, copy the **dist** folder and paste it in a convenient location. Other files and folders are not required by us.
* Fetch the ‘**swagger.yaml’** file that we created previously and paste it into this folder.



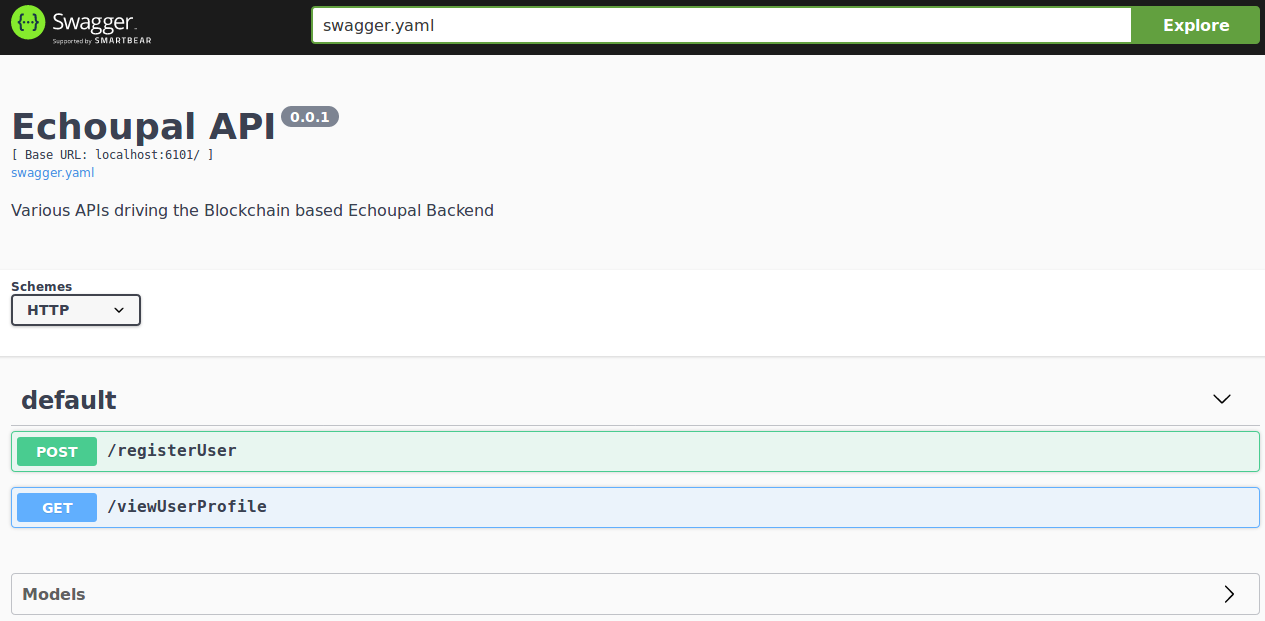
* Inside the dist folder, look for an **index.html** file.
* Open this file in a text editor and look for the following code:



* Change the url value from <http://petstore.swagger.io/v2/swagger.json> to a relative path to our YAML file, and then save the file. For example

url: "swagger.yaml",

* View the **index.html** file locally in our browser. Note that Chrome’s security restrictions (CORS objections) prevent us from viewing the Swagger UI file locally.
* It should look like:



* Well we have successfully generated a Swagger Documentation with an Interactive UI which can be used to test our APIs.

# **References**

* <https://editor.swagger.io/>
* <https://swagger.io/docs/specification/about/>
* <https://idratherbewriting.com/learnapidoc/pubapis_swagger.html>
* <https://github.com/swagger-api/swagger-ui>