**API Simulation**

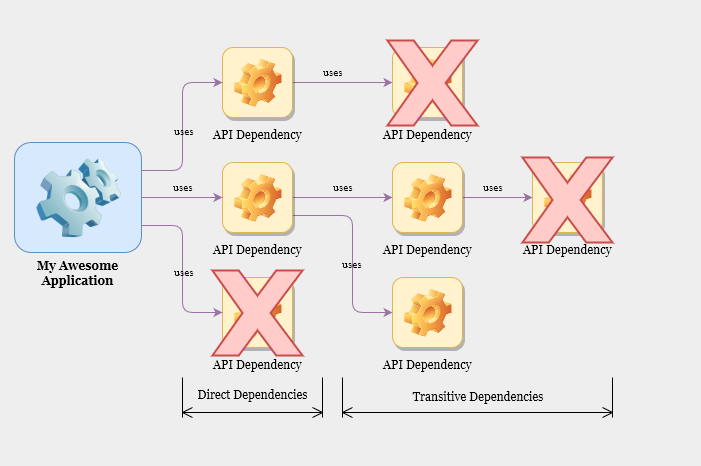
**Problem Statement**

Today, as we extensively work on Microservices Architecture which helps developers to develop & deploy services independently, but still there is a tight dependency between Front End & Backend REST API’s as well as between Backend REST API’s and External API’s.

**Scenario**

For instance, let consider a situation where backend developers build a REST API which collects some data from front end & post it to the External API’s. Now both front end & Back end API's are dependent on External API's & only when those External API's are completed & deployed, we shall be able to integrate & test the App End to End. So, Development & Testing come to a halt when any one of the direct and transitive API dependencies is needed but unavailable.

Also, in situations where there is a frequent outage in External API’s, in-house development & testing is blocked or when they have limitations in number of free requests per day, it tends to increase the development cost!



**Solution ? API Simulation!**

API Simulation mimic real API’s or even API’s which is not built yet!

Using API Simulation techniques, developers can mimic the direct & transitive dependent API’s & obtain response objects like real API’s. Also, API’s mocked using Simulators are always available & fast!

**Key Objective of API Simulation**

* Removes the direct & transitive dependencies in API’s
* Fast, Reliable & Independent of API outages
* Reduces the idle time of developers
* Cost effective when the external API’s have limitations in number of requests per day
* Provides an elegant way of Mocking API’s instead of hardcoding the response objects in Project Source code

Some of the Open Source API Simulation Products available in Market,

* API Simulator
* WireMock
* Hoverfly
* Karate

Out of these products, tried API Simulations using **API Simulator & WireMock**.

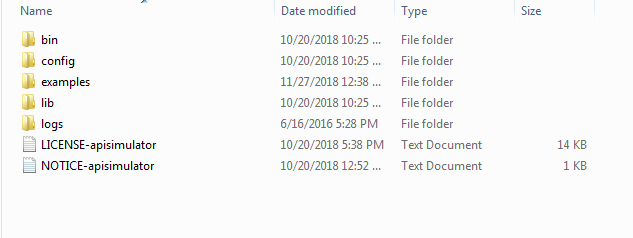
**API Simulator**

API Simulator is a Free & Open-Source Software helps to mimic the real API’s which is still in development or for API’s not even built!

**Steps to mimic REST Operations using Standalone API Simulator**

* Download API Simulator as a zip file from below link & unzip it

<https://apisimulator.io/download/>

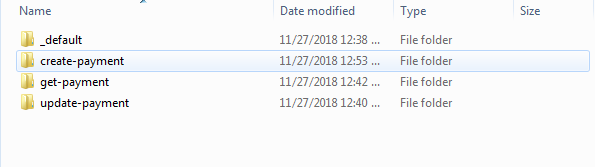


This will be the folder structure once you unzip the downloaded file.

* API Simulator accepts the Request & Response Stubbing in form of yaml files.

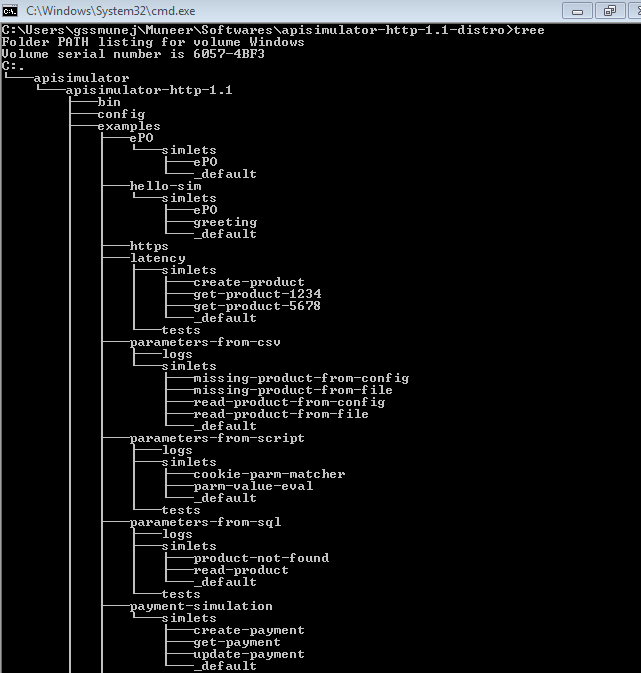
cd into examples folder & create a folder for your simulation. Here, we will try to mimic payment-simulation API. So, create a folder **payment-simulation** inside examples folder

* + Inside payment-simulation folder, create a folder called **simlets.** This is the naming convention provided by API Simulator. So, this is mandatory folder for every Mock API.
  + Inside simlets folder, create folders for your REST Operations like below



The above folder contains Request/Response Stubbing (in the form of yaml files) & the folder **\_default** contains the Stubbing for default Http Request!

So finally, folder structure will be like below [payment-simulation is at the last]



* create-payment folder performs Mocking for POST Request. Inside that folder, create a simlet.yaml file **[do not change the name of the file].** Content of simlet.yaml file is

#----------------------------------------------------------------------

# Simlet-specific configuration.

#----------------------------------------------------------------------

**matchers**:

- method: POST

- where: UriPath

equals: **"/flex/payments/createPayment**"

- header: "Content-Type"

equals: "application/json"

- where: parameter

named: PaymentID

exists: true

- where: body

element: ".payment.name"

exists: true

PaymentID:

is: parameter

from: body

element: ".payment.id"

**response**:

from: template

template: Simula

status: **201**

headers:

- "Location: /flex/payments/${PaymentID}/details.json"

body: **'{"payment":{"Message":"Payment Done!","Result":"Success"}}'**



**Attached the file for reference**

From the above yaml file, REST End point (Highlighted in yellow) will be exposed once API Simulator is started & when the request is posted with Body Content (Since this is a POST Request) using POSTMAN or any REST Client matches/satisfies the condition provided in yaml file, we will get 201 Response Status with Response Body as provided in the above yaml file (highlighted in pink)

* Start the API Simulator Server using the below commands
  + cd into the bin folder & execute the below command from Command Prompt

**syntax** :: apisimulator start *<relative path of simlets>*

**Example** :: apisimulator start examples/payment-simulation

Once executed, it will start the API Simulator & logs will be available in the corresponding folder [examples/payment-simulation/apisimulator.log]

Default port for API Simulator is **6090**  & admin port is **6190**

So, the REST Endpoint for create-payment will be,

<http://localhost:6090/flex/payments/createPayment>

* To Execute the POST Request, Open POSTMAN or any REST Client.
  + Create a New Request & Select POST method
  + Enter the REST End Point as <http://localhost:6090/flex/payments/createPayment>
  + Headers :: Content-Type – application/json
  + Body [Raw Body as json] ::

{

"payment": {

"id": "1234",

"name": "Muneer",

"category": "Electicity Bill",

"subCategory": "TNEB",

"Mode": "Debit Card"

}

}

Above body content will match with yaml file since condition provided in yaml file (Highlighted in blue) states that body of json should contain Id & name

Posting the above request will hit the API Simulator Mock Server & returns with 201 status & Success Body Content provided in the yaml file (Highlighted in pink)

Similarly Mocking can be done for GET & PUT REST Methods too!

Attaching the yaml files for GET & PUT

GET Request PUT Request

To Start API Simulator Mock Server in a different port,

**Syntax**: apisimulator start ***<relative path of simlets>*** **-p *<port number>* -admin\_port *<admin port number>***

**Example:** apisimulator start examples/payment-simulation -p 8080 -admin\_port 8081

**WireMock**

WireMock is a Free & Open-Source Software helps to simulate REST API’s. It enables you to stay productive when an API you depend on doesn't exist or isn't complete. It supports testing of edge cases and failure modes that the real API won't reliably produce.

**Steps to mimic REST Operations using Standalone WireMock Service**

* Download WireMock as a jar file from the below link,

<http://wiremock.org/docs/running-standalone/>

* Start the WireMock server using the below command,

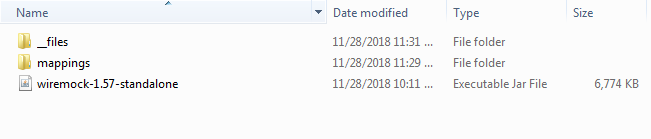
**Syntax** :: java -jar ***<wiremock jar file name>*** --verbose

**Example** :: java -jar wiremock-1.57-standalone.jar --verbose

By Default, WireMock boots up on port 8080. To run on a different port, use -port option

java -jar wiremock-1.57-standalone.jar --verbose **-port 8454**

* Once WireMock is booted up, it will create the below folder structures in the source location



All the Mock Requests & Response should be created inside mappings folder.

* WireMock accepts mocking in the form of Json files. Here, we will try to mimic Expense Manager API Simulation. So inside mappings folder, create a json for Mocking Expense Manager API’s

Let’s look out for CreateExpense API which supports POST Request.

Create a createExpense.json file with the below contents

{

"request":

{

"urlPattern": "/createExpense",

"method": "POST",

"bodyPatterns" : [{

"equalToJson" : "[{ \"name\" : \"Muneer\", \"expenseType\" : \"Personal\", \"expenseAmount\" : \"5000\", \"expenseDate\" : \"28th Nov 2018\" }, { \"name\" : \"Demo\", \"expenseType\" : \"Electricity\", \"expenseAmount\" : \"2000\", \"expenseDate\" : \"26th Nov 2018\" }]"

}]

},

"response":

{

"status": 200,

"headers":

{

"Content-Type" : "application/json"

},

"body": "{\"Result\" : \"SUCCESS\", \"Message\" : \"Expense Record created successfully!\"}"

}

}

In the above Mock Json file, REST API (Highlighted in Yellow) will be exposed once WireMock is started! The Body Content of the Request posted against this API (from POSTMAN or any other REST Clients) should match with Request available in the above json file (Highlighted in Green).



Attaching the createExpense.json file for reference

* Once done, stop the WireMock server & restart it again!
* To Execute the POST Request, Open POSTMAN or any REST Client.
  + Create a New Request & Select POST method
  + Enter the REST End Point as <http://localhost:8080/createExpense>
  + Headers :: Content-Type – application/json
  + Body [Raw Body as json] ::

[

{

"name": "Muneer",

"expenseType": "Personal",

"expenseAmount": "5000",

"expenseDate": "28th Nov 2018"

},

{

"name": "Demo",

"expenseType": "Electricity",

"expenseAmount": "2000",

"expenseDate": "26th Nov 2018"

}

]

Above Body Content will match with Mock Request & it will return 200 Status Code & Response Json(Highlighted in Blue) as provided in the Mock Response Json file.

Similarly Mocking can be done for GET & PUT REST Methods too!

Attaching the json files for GET & PUT

GET Request PUT Request

**Note:**

* Whenever Mock Request/Response is newly added or modified, Mock Server needs to be restarted. This is applicable for API Simulator as well as WireMock!

**To Discuss more on API Simulation, Contact**

* Sudhakar Ganesan - [sudhakar.ganesan@flex.com](mailto:sudhakar.ganesan@flex.com)
* Muneer Ahmed - [MuneerAhmed.J@flex.com](mailto:MuneerAhmed.J@flex.com)