Mayank Mishra (198648)

WireMOCK

WireMock is a flexible API mocking tool for fast, comprehensive, and reliable testing of edge cases and failure modes in your code. It allows you to stub out (mock) an HTTP API, and then test your code’s HTTP interactions with that API.



WireMock is particularly useful when you’re developing a client that depends on a third-party HTTP API, and you want to test that your client code is correct without actually sending real HTTP requests to the API server.

WireMock is also useful for integration testing. For example, if you have a microservices architecture, you can use WireMock to mock out the APIs of dependent services and test that your service is integrating correctly with those APIs.

WireMock is available as a standalone JAR file that you can run from the command line. It is also available as a library that you can include in your code and use from within your tests.

**Setting up WireMock**

To use WireMock, you’ll need to have Java 8 or later installed on your machine. Then, download the latest version of the WireMock JAR file from the [WireMock releases page](https://github.com/tomakehurst/wiremock/releases" \t "_blank) and run it with the following command:

java -jar wiremock-standalone-2.26.3.jar

This will start the WireMock server on port 8080 by default. You can change the port by specifying the --port option:

java -jar wiremock-standalone-2.26.3.jar --port 9999

You can also specify a different root directory for the WireMock files (stubs, mappings, etc.) with the --root-dir option:

java -jar wiremock-standalone-2.26.3.jar --root-dir /path/to/root/dir

**Using WireMock**

Once the WireMock server is running, you can create stubs and mappings for your HTTP API. A stub is a canned response that WireMock will return when a specific HTTP request is made. A mapping is a set of rules that tells WireMock what request to match, and what stub to return in response.

Here’s an example of a simple stub that returns a 200 OK response with a JSON body:

{  
 "request": {  
 "method": "GET",  
 "url": "/api/users/1"  
 },  
 "response": {  
 "status": 200,  
 "headers": {  
 "Content-Type": "application/json"  
 },  
 "body": "{\"id\": 1, \"name\": \"Alice\"}"  
 }  
}

To create this stub, you can use the WireMock API or the WireMock GUI.

**Using the WireMock API**

To create a stub using the WireMock API, send a POST request to the /\_\_admin/mappings endpoint with the stub definition in the request body. For example, using curl:

curl -X POST -d @stub.json http://localhost:8080/\_\_admin/mappings

**Using the WireMock GUI**

To create a stub using the WireMock GUI, open a browser and go to http://localhost:8080/\_\_admin/. You'll see the WireMock admin dashboard, where you can create, edit, and delete stubs and mappings. To create a new stub, click on the “Add new stub” button and enter the details of the stub.

**Testing with WireMock**

Once you’ve created your stubs, you can test them by making HTTP requests to the WireMock server. WireMock will match the request to the appropriate stub and return the specified response.

For example, to test the stub we created earlier, you can use curl to send a GET request to the /api/users/1 endpoint:

curl http://localhost:8080/api/users/1

This should return a 200 OK response with the JSON body {"id": 1, "name": "Alice"}.

You can also use WireMock to verify that your code is making the expected HTTP requests. WireMock will keep track of all requests that it receives, and you can use the /\_\_admin/requests endpoint to view the request history.

For example, to view the request history, you can use curl to send a GET request to the /\_\_admin/requests endpoint:

curl http://localhost:8080/\_\_admin/requests

This will return a list of all requests that have been made to the WireMock server, along with details about each request (method, URL, headers, etc.).

WireMock is a powerful tool for testing HTTP APIs and integration points in your code. It allows you to easily stub out HTTP responses and test your code’s behavior without making actual HTTP requests. This can make your tests faster, more reliable, and more deterministic.