

Speer.io Backend Technical Assessment
Mock Twitter API

Author: Jacob Michels
Email: jmichels@uoguelph.ca

Assigned: Friday February 12th @ 4:28PM
Due: Sunday February 14th @ 4:28PM

Documentation: How to run

I've uploaded an image of my API to docker hub, at this link:

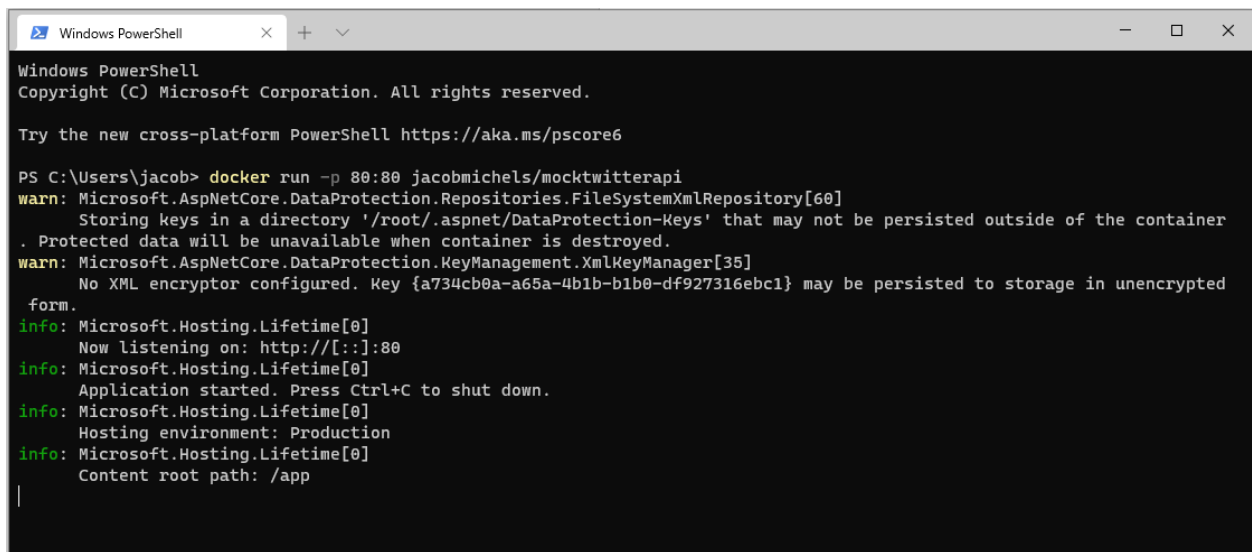
<https://hub.docker.com/repository/docker/jacobmichels/mocktwitterapi>

Prerequisites: Docker

Getting the app running is fairly easy, simply copy and paste this line into your terminal.

```
docker run -p 80:80 jacobmichels/mocktwitterapi
```

This will pull my docker image from the link above and run it in the terminal. You should see output similar to the example below soon after running the above command.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\jacob> docker run -p 80:80 jacobmichels/mocktwitterapi
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
      Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container
      . Protected data will be unavailable when container is destroyed.
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key {a734cb0a-a65a-4b1b-b1b0-df927316ebc1} may be persisted to storage in unencrypted
      form.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
|
```

Congratulations! The API is now running on your computer and is ready for requests.

Documentation: How to test

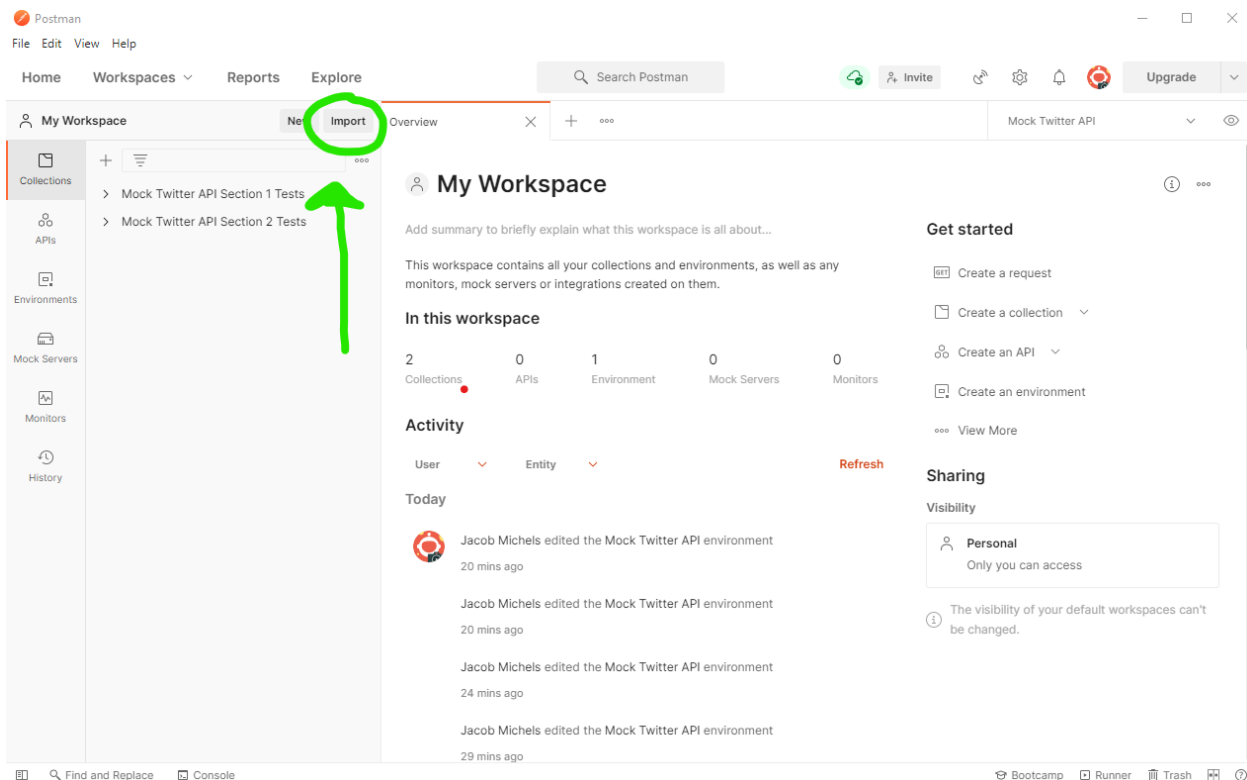
I've created a semi-automated test suite using Postman collections. Here, I will show you how to import the test suite and run it.

Prerequisites: Postman

Note: The web client for Postman is probably suitable for running my tests, but I have not tested it. I recommend the desktop client.

Step 1: Download the postman.json files required from the API's GitHub repository. The files are located in the directory `/MockTwitterAPI/PostmanTests/`. I suggest placing them somewhere easily accessible like the desktop. [Here is a link to the required files.](#)

Step 2: Open Postman and click the import button. We are going to import the files we just downloaded, so you may wish to create a new workspace to prevent cluttering up your existing workspace, but this is optional.



Step 3: Click Upload files and select the files. It should recognize that there are two test folders and one environment variable collection. Click import at the bottom to finalize the process.

Select files to import · 3/3 selected

	NAME	FORMAT	IMPORT AS
<input checked="" type="checkbox"/>	Mock Twitter API	Postman Environment	Environment
<input checked="" type="checkbox"/>	Mock Twitter API Section 1 Tests	Postman Collection v2.1	Collection
<input checked="" type="checkbox"/>	Mock Twitter API Section 2 Tests	Postman Collection v2.1	Collection

Step 4: Now that we've imported the environment and tests, we need to select the environment. Click the environment dropdown button at the top right and select Mock Twitter API.

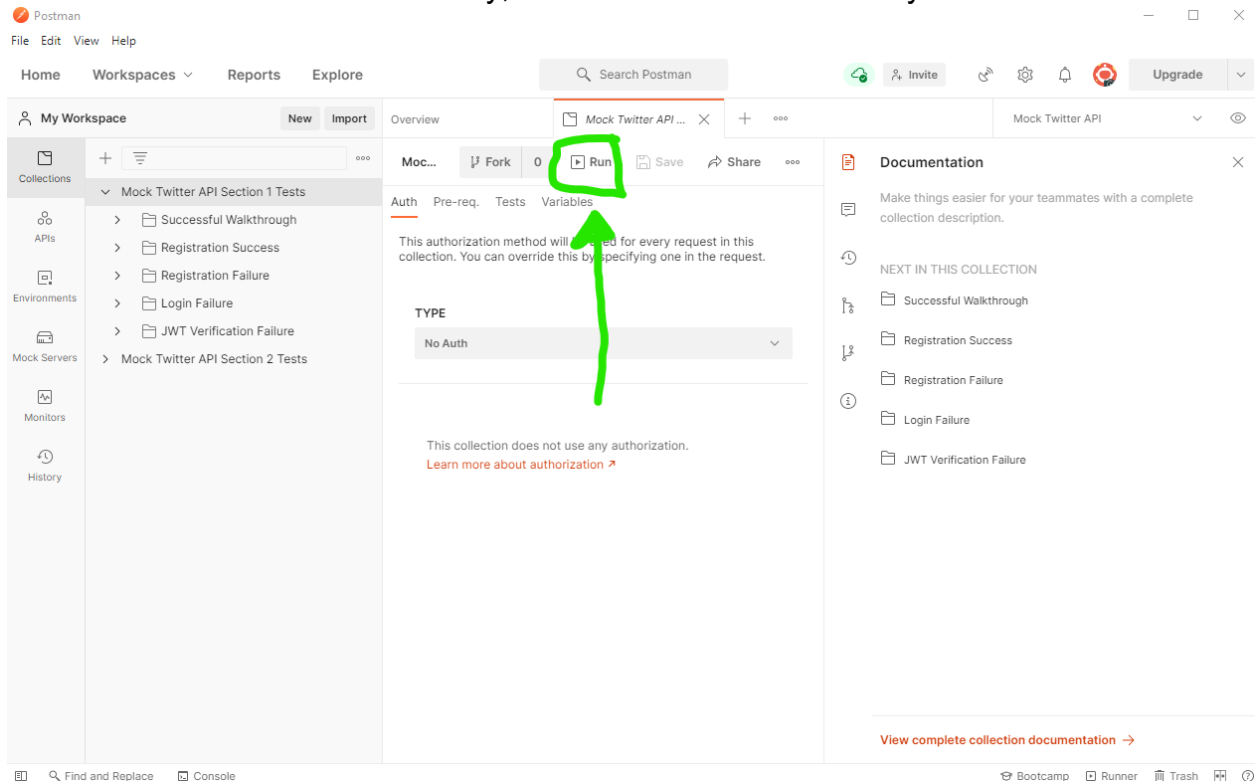
The screenshot shows the Postman application interface. On the left sidebar, the 'Environments' tab is selected, and 'Mock Twitter API' is chosen. The main panel displays the 'Mock Twitter API' environment variables table. A green circle highlights the environment dropdown menu at the top right, which currently shows 'Mock Twitter API'. A green arrow points to the 'Persist All' button in the table's header row. The table contains the following variables:

	VARIABLE	INITIAL VALUE	CURRENT VALUE	
<input checked="" type="checkbox"/>	Username	NewUser	NewUser	
<input checked="" type="checkbox"/>	Password	Abc12	Abc12	
<input checked="" type="checkbox"/>	JWT	Placeholder	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZWQ9.eyJzdWIiOiI2YjAxZjAx...	
<input checked="" type="checkbox"/>	NewChatRecipient	Receiver	Receiver	
<input checked="" type="checkbox"/>	ChatID	Placeholder	e0200655-1031-42a0-a385-60f851947f95	
<input checked="" type="checkbox"/>	TweetID	Placeholder	3ca7f5f9-a58f-4cc4-9818-c89016932840	
<input checked="" type="checkbox"/>	Address	http://localhost	http://localhost	
<input checked="" type="checkbox"/>	Port	80	80	
	Add a new variable			

At the bottom of the interface, there is a notification: 'Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. Learn more about variable values'.

Step 5: Now let's click the environments tab on the far left of Postman and inspect the environment variables. Depending on how you're running the API, you may need to change the address and port variables. If you're running the API through the docker run command above, you shouldn't need to change the port. Depending on your operating system, you may need to change `localhost` to `127.0.0.1`.

Step 6: We should now be ready to run the tests. Please ensure that the API is running. In Postman, click the collections tab on the left. Then click the collection named “Mock Twitter API Section 1 Tests”. This opens a new tab. Click the run button as shown below. The tests for section 1 will begin to run. If you see errors, or the tests don’t start, please review environment variables and ensure the API is running. Please contact me if the tests do not function correctly, I’m sure we can find out why.



Step 7: Once all the tests are completed for the first collection, repeat for the second collection, the one named “Mock Twitter API Section 2 Tests”. The process is the same, just click the run button and watch the tests run.

Note: There is no special environment that the tests use, the http requests the tests make are treated as real by the API. This has the side effect that running the tests more than once or out of order may produce different results. An example of this happening is one of the tests creating a new user with the name NewUser. If this test is ran again with the same data, it will fail because there is already a user named NewUser. To remedy this, there is an endpoint DELETE /Debug that you can hit that will completely clear the database. Obviously, this is not a good idea for a production API, but I thought it was a decent enough solution to the problem.

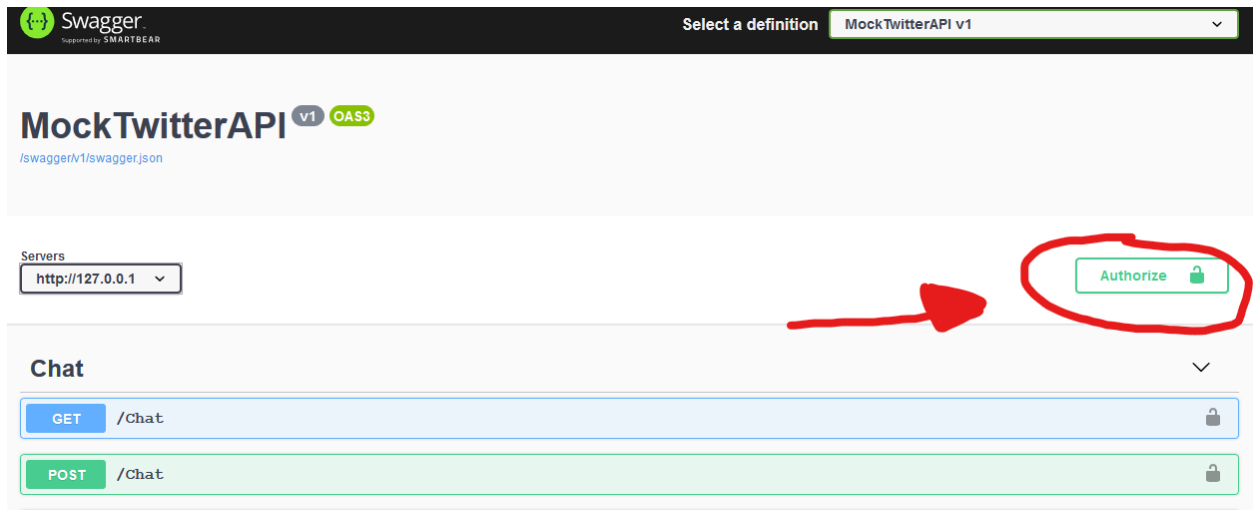
Feel free to play with the tests and try to break the API. I do think my testing was thorough but it’s totally possible I missed some big cases.

Documentation: Swagger

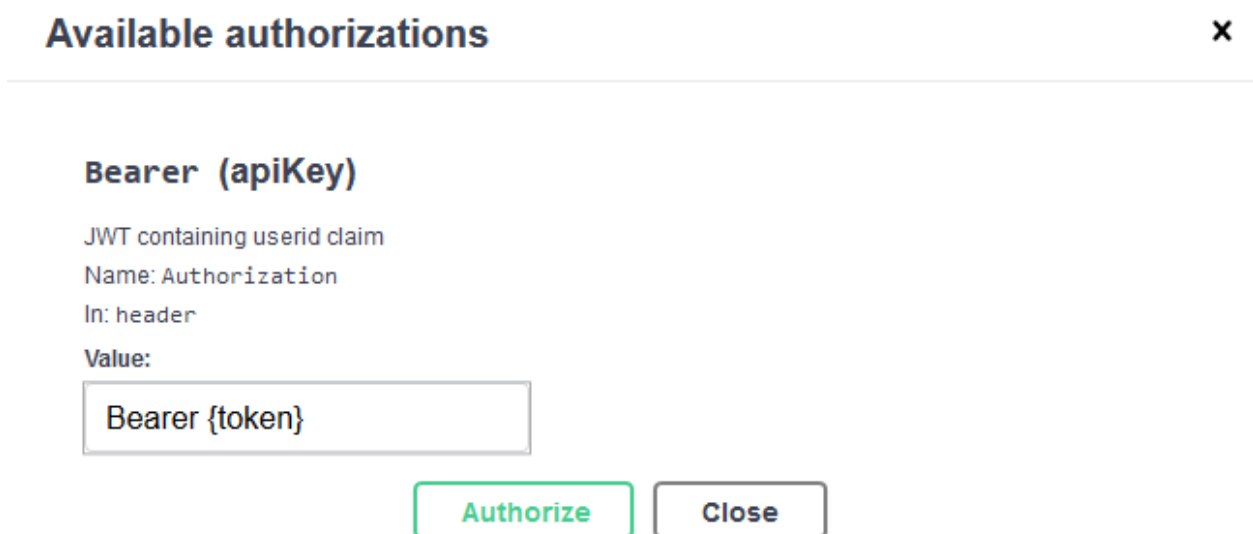
I've included Swagger in this application for an easy use way to visualize and play with the API.

To access it, assuming you're running the application with the docker run command specified above, simply navigate to <http://localhost/swagger> or <http://127.0.0.1/swagger>

For many of the endpoints in the API, users need to be authorized to use them. To authorize in Swagger, simply register, then login. The response of the login endpoint should be some JSON and a JWT string. Copy the JWT, scroll to the top of the page and click the green Authorize button.



Before pasting the token into the textbox, type the word "Bearer". This is a required part of the authorization process.



Project Highlights

Asynchronous code

Wherever possible instead of blocking while waiting for a database query, my codes execute asynchronously. This is important in high throughput applications when blocking takes up too many resources and can severely slow the response time of the system. Asynchronous code is less important in a small-scale application but understanding how to write asynchronous code and why it's important is a good skill to know.

ASP.NET Identity

Instead of writing my own solution for accounts and authorization I used the Identity API that ships with ASP.NET core. I believe this is a good idea because I'm not a security expert and would probably leave accidental security holes in my code. By choosing a tested authentication/authorization library, I am not only saving time but likely creating a more secure application.

JSON Web Tokens

To manage logins, I decided to use JSON web tokens (JWTs). I had never used this kind of system before, but it was surprisingly simple to implement. From what I've read, JWTs are a modern industry standard for user authentication, so I'm glad I was able to learn about and implement them here.

Entity Framework Core

Instead of interacting with a database directly via SQL, I chose to instead use Entity Framework (EF) Core ORM. EF Core comes with ASP.NET core and makes it simple to interact with a relational database without writing SQL. While I am comfortable with SQL, I decide to use an ORM to save me some time building the application.

Docker

I opted to use Docker to greatly simplify the deployment process. Being able to run code in a cross-platform container is extremely helpful for speeding up deployment. When using Docker together with Visual Studio, deployment to docker hub is only a few clicks away. Due to these reasons, I knew it was a no brainer to go for Docker.

Returning JSON

Like a real backend, all my endpoints return JSON. When I started the project, I started off by responding to requests with prettily formatted strings. I quickly realized that this approach makes absolutely no sense for a backend system. It's not my job to make things look pretty, it's to serve data in an efficient way, which includes ease of parsing.