Table of Contents

[Anatomy of a Flask route 2](#_Toc159942929)

[What is JSON? 2](#_Toc159942930)

[How to interact with your REST API 2](#_Toc159942931)

[How to create stores 2](#_Toc159942932)

[How to create items in each store 2](#_Toc159942933)

[URL parameters 3](#_Toc159942934)

[Working with JWTs 3](#_Toc159942935)

[What is a JWT? 3](#_Toc159942936)

[How are JWTs used? 3](#_Toc159942937)

[Getting an access token: 3](#_Toc159942938)

[Example of using access tokens: 4](#_Toc159942939)

[How long do JWTs last for? 4](#_Toc159942940)

# Anatomy of a Flask route

There are two parts to a Flask route:

* The endpoint decorator
* The function that should run

The endpoint decorator (@app.get("/store")) registers the route's endpoint with Flask. That's the /store bit. The Flask app knows that when it receives a request for /store, it should run the function that was put there. The function's job is to do everything that it should, and in the end return something. Most REST APIs return JSON format but can return anything that can be represented as text.

# What is JSON?

Is usually a long string that contains a specific format.

Remember that "turning it into JSON" means two things:

* Change Python keywords and values so they match the JSON standard (e.g. True to true).
* Turn the whole thing into a single string that our API can return.

# How to interact with your REST API

There are two main ways of interacting with it:

* With automated tests
* With manual exploratory testing

Normally you want to start with exploratory first and then make the automated testes based of the manual ones.

A Request Collection is made up of different parts:

* A method: such as GET or POST request is a type of method that is a piece of data sent to the server, but certain methods are used for certain things.
* The URL: that you want to request which in my case is formed of the "Base URL" (for Flask apps, that's http://127.0.0.1:5000), and the endpoint (e.g. /store).
* The body: or any data that you want to send in the request which could look like when creating stores or items we might send some data.
* The headers: which are other pieces of data with specific names, that the server can use and could be sent to help the server understand who is making the request.

# How to create stores

In order to create a store, we receive JSON from our client which will be sent by Insomnia but could be sent other ways such as another Python app or JavaScript.

We start with a POST method and it is usually used to receive data from clients and either use it in some way or create resources with it.

Some important status codes:

* 404- means “Not found”
* 200- means “OK”
* 201- means "Created”

# How to create items in each store

Creating items in a given store will go in the following order:

* The client will send us the store name where they want their new item to go into and be stored.
* They will also send us the name and price of the new item.
* We'll go through the stores one at a time, until we find the correct one (whose name matches what the user gave us).
* We'll append a new item dictionary to that store's items.

## URL parameters

Data can be found in a few more places besides being in the JSON:

* The body (as JSON, form data, plain text).
* Inside the URL which can also be part dynamic.
* At the end of the URL which uses query string arguments.
* In the request headers.

# An introduction to Docker

## What are Docker images and containers?

A virtual machine is an emulation of an operating system, and it is basically a copy of that operating system. Docker containers are different than virtual machines because they don’t emulate an operating system they use the operating system kernel of your computer running as a process within your host machine. They are also lighter because even though they have storage and networking they don’t have the weight of having to be a copy of an operating system.

### What is a Kernel?

An operating system consists of two things:

* The kernel
* Files and programs that come with the operating system

Since containers are using the host kernel a Windows Docker container cannot be run with a MacOS host or a Linux host and vice versa.

### Why are containers more efficient than VMs?

Containers can start up faster as well as use fewer resources because it only uses the kernel, and the kernel is already running.

### What does a Docker container run?

To run the flask app there needs to be a Docker image that has all of the needed dependencies besides the OS Kernel

### What is a Docker image?

A Docker image is a snapshot of source code, libraries, dependencies, tools, and everything else but again it’s not the operating system kernel it is the rest that a container needs to run

## How to run a Docker Cont0ainer?

# Working with JWTs

## What is a JWT?

A JWT is a signed JSON object with a specific structure. Our Flask app will sign the JWTs with the secret key, proving that it generated them.

For example, if we want certain endpoints to only be accessible to logged-in users, all we do is require a JWT in them. Since the client can only get a JWT after logging in, we know that including a JWT is proof that the client logged in successfully at some point in the past.

## How are JWTs used?

### Getting an access token:

* The client sends authentication information to the API (usually a username and password)
* The API validates them and then generates an access token (in our case, a JWT)
* Inside the JWT the users unique ID is stored
* The access token is sent back to the client for storage and later use

A diagram of a client

Description automatically generated

### Example of using access tokens:

* Imagine you’re making an API with an endpoint, /my-info, which should return information about the currently logged-in user.
* Imagine the client is a website. On the website there is a button, “See my info”, which when clicked sends a request to /my-info to get the information.
* Let’s say a user arrives at the website without having logged in, what does the flow look like?

### How long do JWTs last for?

Or how often do users need to authenticate?

* Access tokens don’t last forever as that would be a security risk (if the user forgets to log out in a shared computer)
* Shorter expiry times are safer but if users have to re-authenticate every few minutes, it’s incredibly annoying
* That is where token refreshing comes into play