Goals

- Create a versioned state server using *flask*
  - store key-value pairs of data with a version number
- Implement the API(s) that will allow users to interact with your web server
  - PUT, GET, DELETE request
- Explore two solutions to the same problem
  - store data locally in memory
  - store data remotely in a MongoDB database
Versioned State Server
API Requirements

PUT /<key>

- Add a versioned object to storage
- Version number starts at 1
- Return HTTP/200 if successful
- You can get the contents of the request as a UTF-8 string with `request.data.decode("utf-8")`
- Example: PUT /date with data content “2022-03-26”
API Requirements

GET /<key>

- Retrieve the latest version of a key
- Return a JSON containing the latest value stored for <key> and the corresponding version number
- JSON format: `{ "value": "<string>", "version": <number> }`
- Return HTTP/200 if successful, HTTP/404 if key not found
- Example JSON: `{ "value": "2022-03-28", "version": 3 }`
API Requirements

GET /<key>/<version>

- Retrieve a specific version of a key
- Similar to GET /<key> but for a specific version instead of the latest version
- Return HTTP/200 if successful, HTTP/404 if key not found
- Example: GET /date/2
- Example JSON: { "value": "2022-03-27", "version": 2 }
API Requirements

DELETE /<key>

- Completely delete all data associated with a key
- Delete the key and all version of the key
- Future versions of the key begin again with 1
- Return HTTP/200 if successful
Example

**PUT /date**

Content: “2022-03-26”

<table>
<thead>
<tr>
<th>Version</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022-03-26</td>
</tr>
</tbody>
</table>
Example

PUT /date

Content: “2022-03-27”

<table>
<thead>
<tr>
<th>Version</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022-03-26</td>
</tr>
<tr>
<td>2</td>
<td>2022-03-27</td>
</tr>
</tbody>
</table>
Example

GET /date

Result:

```
{ "value": "2022-03-27", "version": 2 }
```
Example

GET /date/1

Result:

```json
{
  "value": "2022-03-26",
  "version": 1
}
```

<table>
<thead>
<tr>
<th>Version</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022-03-26</td>
</tr>
<tr>
<td>2</td>
<td>2022-03-27</td>
</tr>
</tbody>
</table>
**Example**

GET /date/3

Result:

HTTP/404 Not Found

<table>
<thead>
<tr>
<th>Version</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022-03-26</td>
</tr>
<tr>
<td>2</td>
<td>2022-03-27</td>
</tr>
</tbody>
</table>
Example

DELETE /date

Result:

HTTP/200 Success
Part 1: Local Store
Local Storage

Complete app.py in local-store/

- Store the data locally using any Python data structure
- Launch the app with `python -m flask run`
- The server should be hosted on localhost (127.0.0.1) on port 5000
Testing Part 1

Run your server and send HTTP requests using

- `curl` commands in the terminal
- applications such as Postman

To run the test suite

- `python -m pytest test_local.py`
Part 2: MongoDB
MongoDB Data Store

Complete `app.py` in `mongodb-nosql/`

- Store the data in a MongoDB database
- You can run a MongoDB server using docker
  - `docker run --rm -it -p 27017:27017 mongo`
- Install `PyMongo` to connect to MongoDB in flask
- The server should be hosted on `127.0.0.1:5001`
Testing Part 2

Run your server and send HTTP requests using

- `curl` commands in the terminal
- applications such as Postman

(Optional) MongoDB Compass

To run the test suite

- `python -m pytest test_mongo.py`
Question