Passwd as a Service

The idea of this challenge is to create a minimal HTTP service that exposes the user and group information on a UNIX-like system that is usually locked away in the UNIX /etc/passwd and /etc/groups files.

While this service is obviously a toy (and potentially a security nightmare), please treat it as you would a real web service. That means write production quality code per your standards, including at least: Unit Tests, and README documentation. Use any of the following languages and an idiomatic HTTP framework of your choosing: Java/Kotlin/Scala, C#/F#, Python, Ruby, Go, JavaScript, or Rust. Please post your solution to a public GitHub (or BitBucket or GitLab) repository and include instructions for running your service.

To aid testing and deployment, the paths to the passwd and groups file should be configurable, defaulting to the standard system path. If the input files are absent or malformed, your service must indicate an error in a manner you feel is appropriate.

This service is read-only but responses should reflect changes made to the underlying passwd and groups files while the service is running. The service should provide the following methods:

GET /users

Return a list of all users on the system, as defined in the /etc/passwd file.

Example Response:

```
[
  {“name”: “root”, “uid”: 0, “gid”: 0, “comment”: “root”, “home”: “/root”, “shell”: “/bin/bash”},
  {“name”: “dwoodlins”, “uid”: 1001, “gid”: 1001, “comment”: “”, “home”: “/home/dwoodlins”, “shell”: “/bin/false”}
]
```

GET /users/query[?name=<nq>][&uid=<uq>][&gid=<gq>][&comment=<cq>][&home=<hq>][&shell=<sq>]

Return a list of users matching all of the specified query fields. The bracket notation indicates that any of the following query parameters may be supplied:

- name
- uid
- gid
- comment
- home

- shell

Only exact matches need to be supported.

Example Query: `GET /users/query?shell=%2Fbin%2Ffalse`
Example Response:

```
[
  {“name”: “dwoodlins”, “uid”: 1001, “gid”: 1001, “comment”: “”, “home”:
“/home/dwoodlins”, “shell”: “/bin/false”}
]
```

GET /users/<uid>

Return a single user with <uid>. Return 404 if <uid> is not found.

Example Response:

```
{“name”: “dwoodlins”, “uid”: 1001, “gid”: 1001, “comment”: “”, “home”:
“/home/dwoodlins”, “shell”: “/bin/false”}
```

GET /users/<uid>/groups

Return all the groups for a given user.

Example Response:

```
[
  {“name”: “docker”, “gid”: 1002, “members”: [“dwoodlins”]}
]
```

GET /groups

Return a list of all groups on the system, a defined by /etc/group.

Example Response:

```
[
  {“name”: “_analyticsusers”, “gid”: 250, “members”:
[“_analyticsd’,”_networkd”,”_timed”]},
  {“name”: “docker”, “gid”: 1002, “members”: []}
]
```

GET /groups/query[?name=<nq>][&gid=<gq>][&member=<mq1>[&member=<mq2>][&...]]

Return a list of groups matching all of the specified query fields. The bracket notation indicates that any of the following query parameters may be supplied:

- name
- gid
- member (repeated)

Any group containing all the specified members should be returned, i.e. when query members are a subset of group members.

Example Query: `GET /groups/query?member=_analyticsd&member=_networkd`
Example Response:

```
[
  {"name": "_analyticsusers", "gid": 250, "members":
["_analyticsd","_networkd","_timed"]}
]
```

GET /groups/<gid>

Return a single group with <gid>. Return 404 if <gid> is not found.

Example Response:

```
{"name": "docker", "gid": 1002, "members": ["dwoodlins"]}
```