Code Challenge for Mercedes-Benz.io

Context

While we are huge fans of micro-service architecture this is not without issues, we expect that each and every micro-services we depend on are always up and running, but that's not always the case.

So, our developers decided to build a tool to help them monitor if a given micro-service/site is up or down and display it in their command line.

Your challenge is to build a command line interface (CLI), that is capable of checking if a given service is live or not and display that information. Notice that you are required to monitor https://status.bitbucket.org and at least another one of the following services:

- https://status.github.com
- https://status.slack.com
- http://status.gitlab.com
- https://www.google.com/appsstatus (select one of the sub-services to monitor)

Don't forget to provide a README.md file with:

- Your name, e-mail address, college name, degree and attendance year;
- Descrive your solution and relevant design decisions.

Build a CLI tool for checking if several services are UP or DOWN service availability

CLI Tool commands

***poll*:**

- Retrieves the status from of all configured services:
  - Outputs results of all services
  - Saves the result to local storage (don't use a database)
  - *Bonus*: Pass an argument **only** to *poll* in order to retrieve a specific set of services (eg: *--only=github,slack*)
  - *Bonus*: Pass an argument **exclude** to *poll* in order to exclude a specific set of services (eg: *--exclude=slack*)

***fetch*:**

- Retrieves the status from of all configured services with a given interval (default interval: 5 seconds):
  - Saves the result to local storage (don't use a database)
  - Outputs results of all services
  - Configurable polling interval, with default of 5 seconds (eg: *--refresh=60*)
  - *Bonus*: Pass the argument **only** to *fetch* in order to retrieve a specific set of services (eg: *--only=github,slack*)
  - *Bonus*: Pass the argument **exclude* to *fetch* in order to exclude a specific set of services (eg: *--exclude=slack*)

***history*:**

- Outputs all the data from the local storage:
  - *Bonus*: Pass the argument **only** to *history* in order to retrieve the history of a specific set of services (eg: *--only=github*)

***backup*:**

- Takes an argument (path with the file name) and saves the currect local storage:
  - *Bonus*: Save to a simple .txt file, with a custom format (eg: *--format=txt*)
  - *Bonus*: Save to a simple .csv file, with row data separated by commas (eg: *--format=csv*)

***restore*:**

- Takes an argument (path with the file name) and restores the data in the file into current local storage.
  - *Bonus*: Pass an argument to *restore* in order to merge the content of the input file instead of replacing it (eg: *--merge=true*)

> HINT: Don't forget to validate the content of the import file before starting the import.

***services*:**

- Outputs all services defined in the configuration file and their respective endpoint.

***help*:**

- Outputs all available CLI commands

***status* (bonus):**

- Summarizes data and displays it in a table-like fashion:
  - Print time since webservice has't been down
  - Mean time to failure (MTTF)

Technical challenges building the CLI Tool:

- Don't use a database as local storage. Create your own storage format, be it a simple TXT, JSON, XML file or even a binary file.
- Services can be down at any given time, be prepared for that.
- Services status pages can also be down at any given time, be prepared for that.
- Know how to process the service response. If it's HTML page, know where to fetch the availability status information.
- Know that you can face a broad type of errors, eg: timeout, network unreachable, invalid HTML/JSON structure, etc...

Usage Examples

Configuration file

Your service should use a text file from where your CLI tool will read the services defined there.

Below is an example of a configuration file (yours can be completely diferent, from format to syntax) and you are advised on chosing the format, structure and syntax that you would prefere.

Let's assume that in our configuration file we have the following services defined: *github*, *slack*, and *bitbucket*:

```
{
    "services": [
        {
            "id": "github",
            "name": "GitHub",
            "url": "https://status.github.com/messages"
        },
        {
            "id": "slack",
            "name": "Slack",
            "url": "https://status.slack.com/"
        }
        {
            "id": "bitbucket",
            "name": "Bitbucket",
            "url": "https://status.bitbucket.org/"
        }
    ]
}
```

Other example of a configuration file could be:

```
github|https://status.github.com/messages
slack|https://status.slack.com/
bitbucket|https://status.bitbucket.org/
```

Also considere the following output as an example of how things can be presented as oposite on how you should present:

Poll

Simple poll example polling all known services:

```
#> bot poll
[github] 2018-09-17T15:58:38+01:00 - up
[slack] 2018-09-17T15:58:38+01:00 - up
[bitbucket] 2018-09-17T15:58:38+01:00 - down
```

Simple poll example poll excluding *github*:

```
#> bot poll --exclude=github
[slack] 2018-09-17T15:58:38+01:00 - up
[bitbucket] 2018-09-17T15:58:38+01:00 - down
```

Simple poll example poll including only *bitbucket*

```
#> bot poll --only=bitbucket
[github] 2018-09-17T15:58:38+01:00 - up
```

backup

Simple backup from current storage to an external file:

```
#> bot backup /tmp/backup-file.txt
```

NOTE: In this example the current database is copied to `/tmp/backup-file.txt` without any file conversion.

restore

Restoring an external file in to the local app storage:

```
#> bot restore ~/Desktop/invalid-file.mp4
bot: Can't restore file is invalid
```

```
#> bot restore  ~/Desktop/valid-file.txt
bot: Restored 600 status lines from 2 services: slack, github
```

```
#> bot restore  ~/Desktop/valid-file.txt --merge
bot: Restored 600 status lines from 2 services: slack, github
```

NOTE: The output shown during the import is a mere example, you can display whatever you see fit.

help

Example of the help comand line output

```
#> bot help
bot command [args]

command:
    poll - Retrieves the status from of all configured services
    fetch - Retrieves the status from of all configured services
    services - Lists all known services
    backup - backups the current internal state to a file
    restore - Imports the internal state from another run or app
    history - Outputs all the data from the local storage
    status - Summarizes data and displays it in a table-like fashion
    help - This screen
```

history

Outputs all the data from the local storage

```
#> bot history
[github] 2018-09-17T15:58:38+01:00 - up
[slack] 2018-09-17T15:58:38+01:00 - up
[bitbucket] 2018-09-17T15:58:38+01:00 - down
```

status

Summarizes data and displays it in a table-like fashion

```
#> bot status

| service | uptime |   mttf   |
+---------+--------+----------+
| github  | 120min |  8min    |
```

Help

In case you need help or clarifications regarding the exercise please reach out to recruitment@mercedes-benz.io

---

MB.io Service availability challenge - v1.5.2