Service poller

# Requirements

#### 1. Server

1. Design and build a webserver that allows users to register HTTP(S) endpoints for the server to periodically poll, and report back with the status.
2. The server must be accessible over a network.
3. The server must return OK or FAIL for each service.
4. The server must persist the registered services beyond reboots.
5. The Server should handle situations where one registered service has multiple actions take against it at the same time
   1. E.g. one user deletes a service, another changes the service name.
   2. E.g. as the server updates the status of a service, a user deletes the service.
6. The Server should not be adversely impacted by registered services not returning valid response codes in short time.
   1. E.g. A service that takes 10 minutes to provide a response should not slow down the polling of registered service, or access to the server.
7. The server must be written in Java
8. The server must persist any data in an SQL-based database.

#### 2. UI

1. Design and build a UI through which the user can administer these registered endpoints.
2. The user must be able to view the registered services and their status in a basic UI.
3. The user must be able to register a new service, specifying the service name, and its URL.
4. The user must be able to view the name, URL, creation time, and status of all registered services in the UI.
5. The UI should allow users to update the name and URL of previously registered services.
6. The UI should allow users to delete (de-register) previously registered services.
7. The UI should automatically refresh with the latest status of each registered service i.e. the user should not have to manually refresh the page to view the latest status.
8. The UI should show informative, *nice looking* animations when services are added or removed.
9. The UI should prevent users from registering invalid services with invalid URLs.
10. The UI should support multiple users with different registered services.
    1. Each user should only see the services they registered.

#### Other

1. The UI and Server code must be accompanied by a README in English, containing instructions on how to run the application.

# Assumptions

It is an internal tool, so will not be open to the public on WWW.

This limits the number of services that must be supported, within reason the employees at Kry will only ever be tracking 10k services across all users. Each user will only be tracking 500 services maximum.

It also limits the number of security precautions that must be taken.

There will be one instance of the server running.

For requirement 2.10, the requirement states that users must not see services registered by other users. It does not provide information as to how users will login. As such I assume it is acceptable for users to login with just a username, and there will be no username / password authentication to access the application.

Maximum URL length is 2000.

Services that return 1xx (informational), 2xx (successful), 3xx(redirection) and 429 (too many requests) will be considered OK. All other responses will be considered FAIL.

# Solution Design

#### Server

The server will use Spring Boot to deploy a tomcat webserver.

Persisted data, and the persisting of data will be managed by JPA and Hibernate. The server will be configured to use a MySQL database running on port 3309, although this can easily be changed via configuration.

The server will serve 2 endpoints:

1. “/api/v1/monitored-services” with support for the following HTTP request methods
   1. Get
      1. Requires query parameter ‘username’
      2. Requests will return the registered services for the provided username
   2. Post
      1. Requires request body containing
         1. username
         2. name
         3. URL
      2. Requests will register a new service with the provided name, with the target URL, available to the specified user
   3. Patch
      1. Requires request body containing
         1. id
         2. name
         3. url
      2. Requests will update the service with the provided id, to have the specified service name, and target the specified URL.
   4. Delete
      1. Requires request body containing
         1. id
      2. Requests will delete the registered service with the provided id.
2. “/api/v1/monitored-services/all” with support for the following HTTP request methods
   1. Get
      1. Requests will return all registered services, across all users.

Once a services is registered to be monitored, it will be stored in the database table registered\_service. This table has the following columns:

* id num (PK)
* name str 255
* url str 2000
* status int
* last\_polled\_date date Indexed
* create\_date date
* username str 255

Periodically a scheduled task will query the top 100 records from the registered\_service table sorting by last\_polled\_date ascending (such that the 100 services polled least recently are selected).

We only load 100 services at a time to reduce memory footprint of the application. It also reduces the likelihood of us polling a service that has been deleted since it was queried from the database.

These services are then passed into a parallel stream, where the fork join pool will poll a service, and store the response, mapped to OK or FAIL, back in the database table.

Services that are yet to be polled for the first time, or have not been polled since their name/url was updated will be set to status WAITING.

#### UI

The UI will be coded in React js.

It will contain 3 main components:

* Login box
  + Contains a textbox in which the user can enter their username
* Polled Services
  + Contains a table of the polled services for the user, displaying the data held against that service (e.g. name, url, status, create date, last polled date)
  + The table has action buttons against each row to allow delete and update of the service
  + Contains a box used to register new services
* Edit Service Dialog
  + Contains fields to allow the user to modify the name and URL of a service
  + Is shown when the user presses Edit against a row in the Polled Services table.

All components will use Material UI design, and will use kry brand colours.

The application will store the provided username in the browser, such that repeat accesses from the same browser will be able to use this username and bypass login.

# Possible Improvements

#### Configuration

1. Extract all strings to config files to allow localisation.
2. Allow configuration of polled service batch size.
3. Allow configuration of polled service timeout interval.
4. Allow configuration of max URL length
5. Allow configuration of the response statuses that count as OK (vs those that count as FAIL)

#### Features

1. Implement passwords to prevent users accessing each other’s services.
2. Implement more status results e.g. Redirected

#### Design

1. Use custom fork-join pool for polling registered services with the parallel stream – allow for custom number of threads – server specific
2. Have the ScheduledPollingTask as a standalone server – instead of tying it to the webserver. This would allow spinning up of multiple instances of the server that handles webrequests, while having one server that executes the polling on the registered services.
3. Error handling in the UI – present relevant errors to the user.
4. If multiple users register the same endpoint, only poll it once. Do not poll the same endpoint multiple times per cycle.
5. Do not permit a user to register the same end point multiple times.
6. Move the Edit and Delete buttons to be hidden in a popup actions menu against each row, instead of permanently showing the buttons
7. Fixed width for columns in the Monitored Services table, so the data doesn’t jump around when there are different URL lengths on screen.
8. Only set the monitored service Status back to WAITING when the user edits the service URL, not when they only change the service name.
9. Colour indication for service status.
10. Use Idempotency Keys for the post request.
11. Prevent users from

#### Testing

1. Complete backend test suite
2. Extract testcases to csv

#### Bugs

1. Some services return 403 forbidden when the webserver queries them, when the browser is allowed access fine – need to better emulate browser requests with the scheduled service monitor.
2. Gradlew warnings
3. Monitored Services table filters do not render the whole selected filter status – the field is too small.