Test task: meeting room occupancy web app

Abstract

The goal of this task is to implement a simplified web application, consisting of a back-end and front-end, that may be used to collect and visualize the data from an IoT network.

Use case and assignment

Context

An office building is equipped with sensors at entrances and exits of every meeting room. These sensors count the number of people who go into or out of the room. These sensors are further referred to as “People Counters”. All People Counters are connected to the Internet. A People Counter can be configured with a “webhook listener URL”: when configured accordingly, a sensor will send HTTP requests with the results of the observations to this URL. An HTTP request is produced by the sensor every time it detects a single person or multiple people; such request indicates the number of people detected and their direction.
Building manager wants to monitor the number of people in the meeting rooms in real time.

Objective

The goal of this task is to implement one back-end application and one front-end application, which together would provide an HTTP API to collect and then expose sensor data, as well as a simple user interface to visualize it.

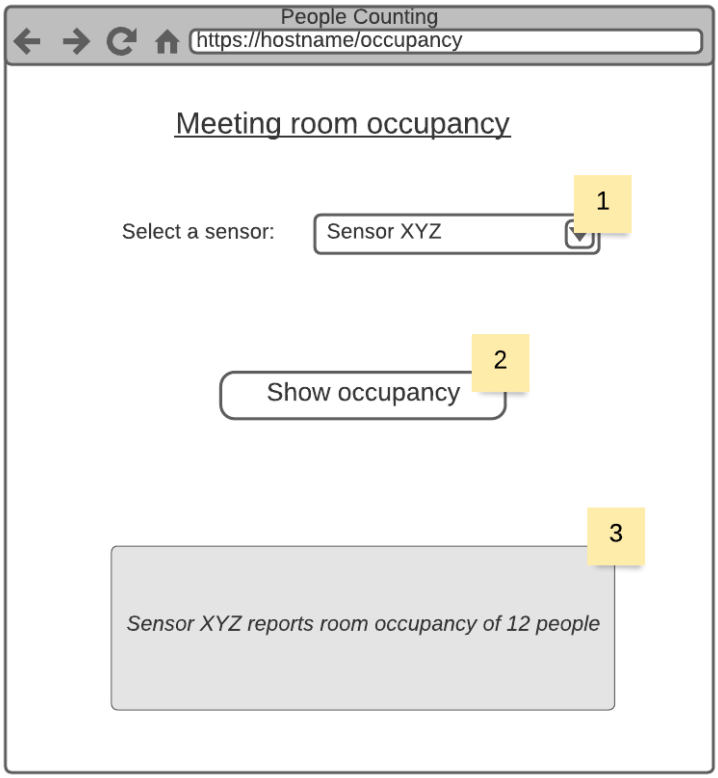
Back-end should provide an HTTP API with at least two endpoints:

1. An endpoint to be called by People Counters and where they may send the results of the observations, as described in the “*Context*” section above. Requests produced by People Counters may be emulated with the following command:
   ```
   curl --header "Content-Type: application/json" \
       --request POST --data \
       '{"sensor":"abc","ts":"2018-11-14T13:34:49Z","in":3,"out":2}' \
       http://hostname/api/webhook
   ```
   where `http://hostname/api/webhook` is this endpoint URL and is assumed to be configured within the sensor.

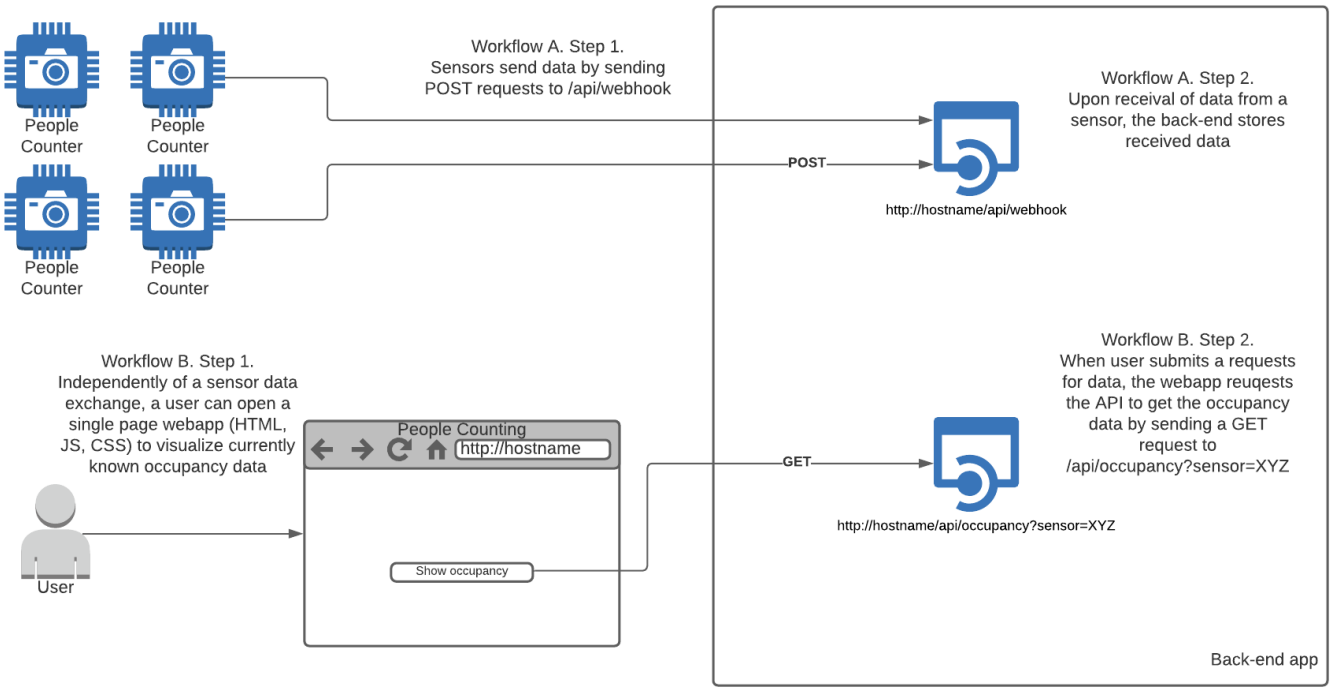
2. An endpoint that provides an aggregated result of the number of people currently present in the meeting room (an occupancy, based on the number of people who entered and left the room). This endpoint is to be used by the front-end application. Following is the command to emulate a request to this endpoint:
   ```
   curl --request GET http://hostname/api/occupancy
   ```
   and its sample output:
   ```
   { "inside": 42 }
   ```

If necessary for the implementation of the front-end, the back-end may provide any additional endpoints and APIs.

Front-end should be a single-page web application (HTML, CSS, JS) served by the same or another server as the back-end. User interface should allow the user selecting the sensor and displaying according meeting room occupancy. A user interface mockup is given below.



Following diagram present the entire system design and data flow:



Functional requirements

- At any point in time, occupancy of a meeting room is a sum of all entries minus a sum of all exits registered by a sensor since the start of the application. Occupancy is calculated for each room separately. Every meeting room is measured by a single dedicated sensor.

Technical requirements

- A solution to this exercise is the source code of two standalone executable applications (one back-end and one front-end) that may be eventually served from a web server.
- A solution may be written with any technology of choice and any programming languages, provided that the choice can be justified.
    - However, the front-end application should be a single-page Javascript application implemented with one of any modern JS frameworks, such as React, Vue, Angular, or a similar one.
- The application may keep all data in memory. For the purpose of this exercise, there is no need to implement any persistent storage or use a database.
- The implementation or both back-end and front-end should be accompanied by unit tests.

Optional assignments

Following assignments are not mandatory. They may be implemented at will.

- `occupancy` endpoint may take an optional query parameter: `atInstant`. When this parameter is defined the API is expected to provide the number of people that were present in the room at a given moment in time:
  `curl --request GET http://hostname/occupancy?sensor=XYZ&atInstant=2018-11-14T14:00:00Z`
  example output:
  `{ "inside": 9 }`
- User interface may leverage the new `atInstant` feature of the `occupancy` endpoint by providing users a selection of the date and time to show the occupancy for.

Solution evaluation

Please, note that a solution will be evaluated for the following criteria, in the given order (former items have more importance than the latter ones):

- Correctness (the solution corresponds to the given requirements).
- Code quality.
- Simplicity, yet effectiveness of the software design.

The choice of a technology used to implement a solution is not a part of evaluation criteria, provided that it meets the Technical Requirements above.