Frankenhaus - Hospital app

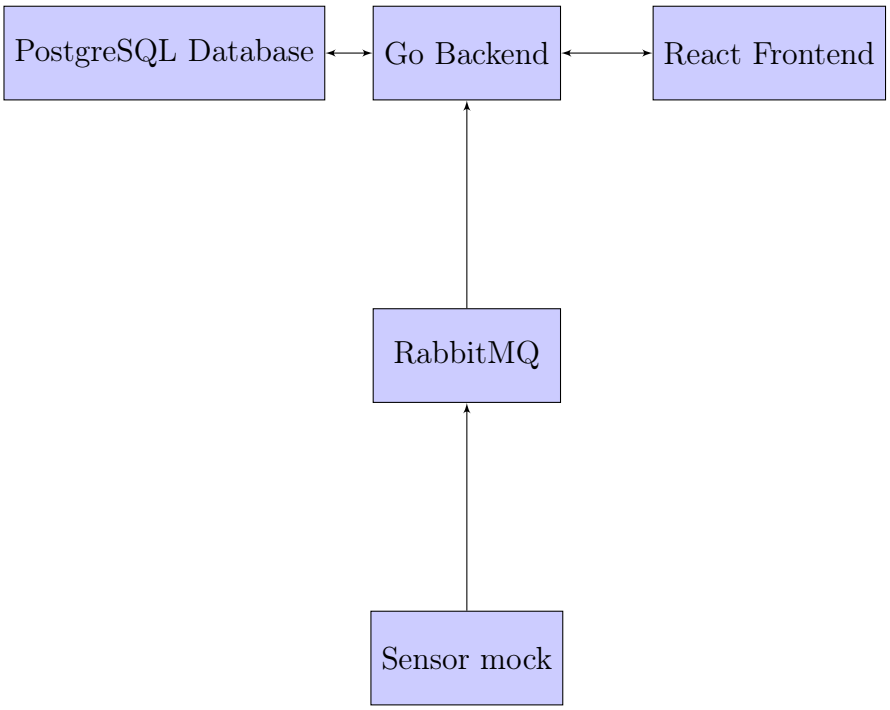
Ghitulescu David

November 7, 2019

30243

Contents

1 Architecture



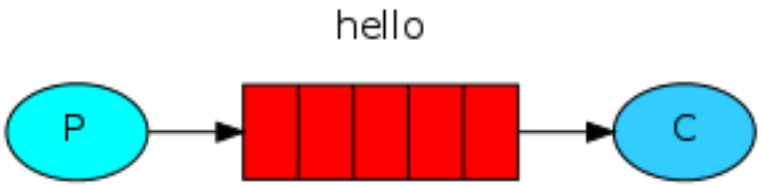
1.1 RabbitMQ

RabbitMQ is a separate server application which runs on my localhost with the sole purpose to collect sensor data and make them available when the backend is ready to process that data.

It has a queue named "hello" which is used the the process described above.

Not much configuration is needed for RabbitMQ especially on Ubuntu. Once the RabbitMQ service is started, everything should run.

The RabbitMQ architecture I have chosen is the following:



Where P is my producer - The Sensor mocking app described in section 3, the red box is the queue name "hello" and C is the Consumer - the backend from Assignment 1 which has been modified according to section 2.

1.2 Backend

The backend from the first application has been modified to now implement the RabbitMQ Consumer with a simple class which simply declares the queue it want and then subscribes to its events.

Once a message has arrived it is unmarshalled into a go object, filtered and send to the corresponding websocket listeners which have already been registered(open websocket connections).

The filtering is done using the 3 rules of the assignment:

1. A patient has been sleeping for more than 12 hours
2. A patient has been outside for more than 12 hours
3. A patient has been sitting on the toilet for more than 1 hour

If any of the above conditions is met a notification message is generated and sent via WebSockets

1.3 Sensor mock app

The physical sensors are mocked(simulated) using a simple go application which reads a file of activities and sends a message to a designated RabbitMQ queue. Below a sample output can be seen with the messages it send via the Rabbit Queue

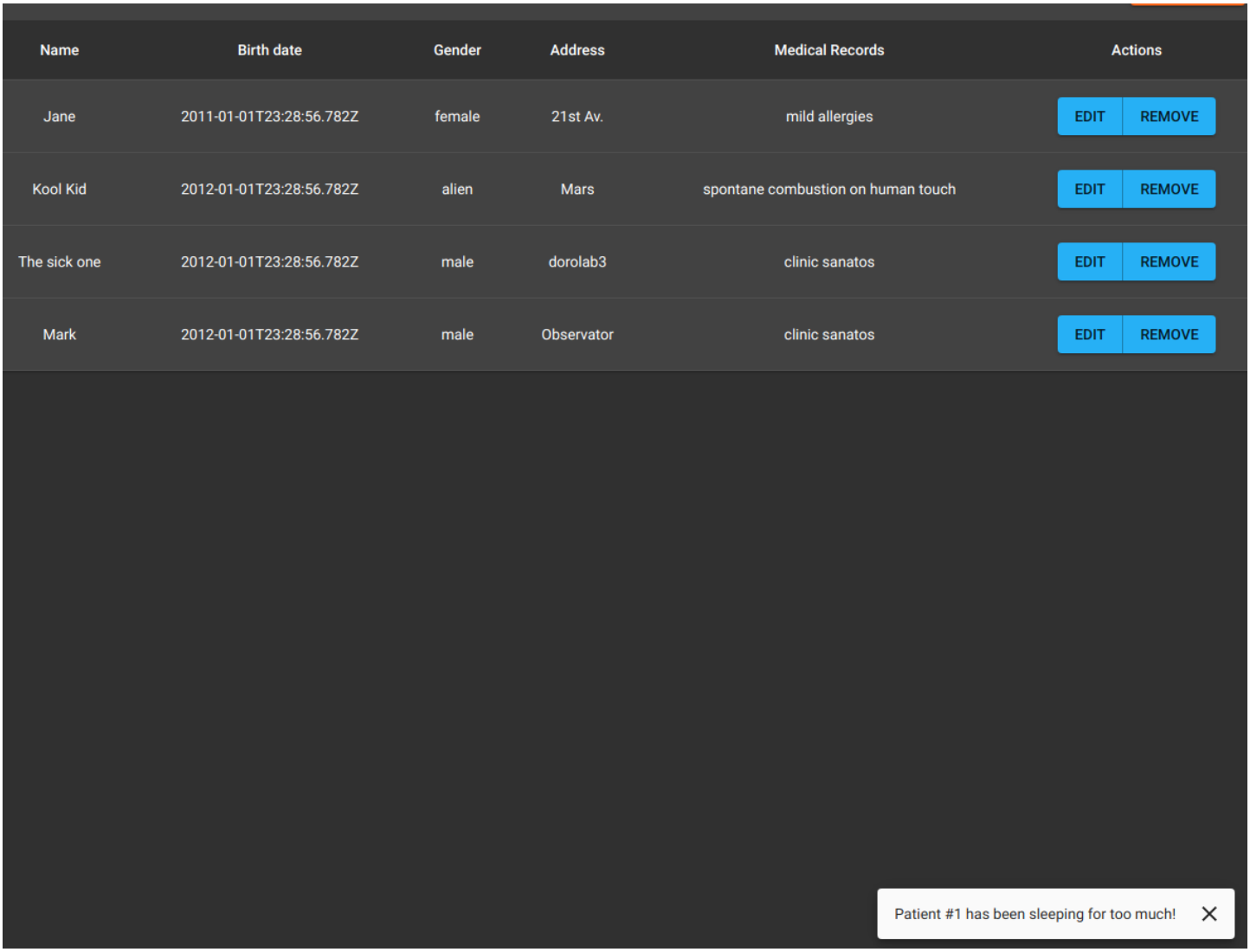
```
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Sleeping", "start": 1322447279, "end": 1322561891}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Toileting", "start": 1322475684, "end": 1322475816}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Showering", "start": 1322475944, "end": 1322476380}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Breakfast", "start": 1322476463, "end": 1322476980}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Grooming", "start": 1322477388, "end": 1322477473}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Spare_Time/TV", "start": 1322477501, "end": 1322485507}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Toileting", "start": 1322485564, "end": 1322485591}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Leaving", "start": 1322485771, "end": 1322486949}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Spare_Time/TV", "start": 1322487520, "end": 1322490100}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Toileting", "start": 1322490158, "end": 1322490427}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Lunch", "start": 1322490431, "end": 1322492640}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Grooming", "start": 1322492699, "end": 1322492789}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Spare_Time/TV", "start": 1322492821, "end": 1322511600}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Snack", "start": 1322511655, "end": 1322511659}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Spare_Time/TV", "start": 1322511675, "end": 1322532360}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Sleeping", "start": 1322532960, "end": 1322566260}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Toileting", "start": 1322566315, "end": 1322566615}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Grooming", "start": 1322566658, "end": 1322567334}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Showering", "start": 1322567397, "end": 1322567473}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Breakfast", "start": 1322568508, "end": 1322569080}
2019/11/07 16:01:50  [x] Sent {"patient_id":1, "activity":"Grooming", "start": 1322569141, "end": 1322569320}
```

In order to parse the file and extract the relevant data, a regular expression is used with three groups corresponing to the three double-tab separated sections:

1. Activity Name
2. Activity Start time - parsed into a unix timestamp
3. Activity End time - parsed into a unix timestamp

1.4 Frontend

A notification component has been added which get spawned automatically when a WebSockets message is received. It can be closed with the X button.



2 Installation

To run this application the following **dependencies** must be met:

1. **RabbitMQ** (see installing RabbitMQ)
2. **Go** (see installing go)
3. **Node** with textbfNPM (see installing npm)
4. Port 5432 is not blocked(DB access)

The textbfsetup is pretty straight-forward:

1. Run in the **backend** folder:
   (a) **./run.sh**
2. Run in the **frontend** folder:
   (a) **sudo npm i**
   (b) **npm run start**
3. Run in the **act_sensors** folder:
   (a) **go build main.go**
   (b) **go run main.go**

If the application doesn't start automatically, access localhost:3000