Report

Assignment 3

2016 - 11 - 17

Björn Hansson
Simon Gullstrand
Jonas Wahlfrid

Prerequisites

- Mongodb will need to be installed and running with default settings.
- Nodejs will need to be installed.
- Chrome should optimally be installed and used for viewing the webpage. (Firefox will be fine but the date pickers works better in Chrome).

Inside our source code project, run *npm install* in a terminal to download the external Nodejs dependencies and libraries. Start the server by running *node bin/www*. Mongodb will need to be running already for the server to start correct. Open http://localhost:3000/ in Chrome to view the client webpage. Then “fake” a camera event by open http://localhost:3000/api/camera/ or set up a real Axis camera to send notifications to this URL.

Description of the system

Axis camera

The axis camera works as a client to the server in this setup with a single goal to send HTTP notifications to the server when some kind of motion is detected. We managed this by adding an event and a event trigger though the cameras GUI interface (Axis Video Motion Detection), which is strongly recommended by the assignment instructions.

Server

The server code is written in JavaScript. It runs on Nodejs, with standard setup libraries like express for routing and mongoose for Mongodb management, etc. Mongodb was chosen because of the ease of use with the mongoose driver and for a full-stack JavaScript solution. The server acts as a middleware for the camera and the user client. It stores the motion data acquired from the camera and then allows the client to retrieve it by a simple HTTP GET request for it. The server is similar to a REST web service, but a very simple one with GET and POST operations. We did not see any reason to add something more advanced for this assignment. However, it can easily be extended if needed.

Client

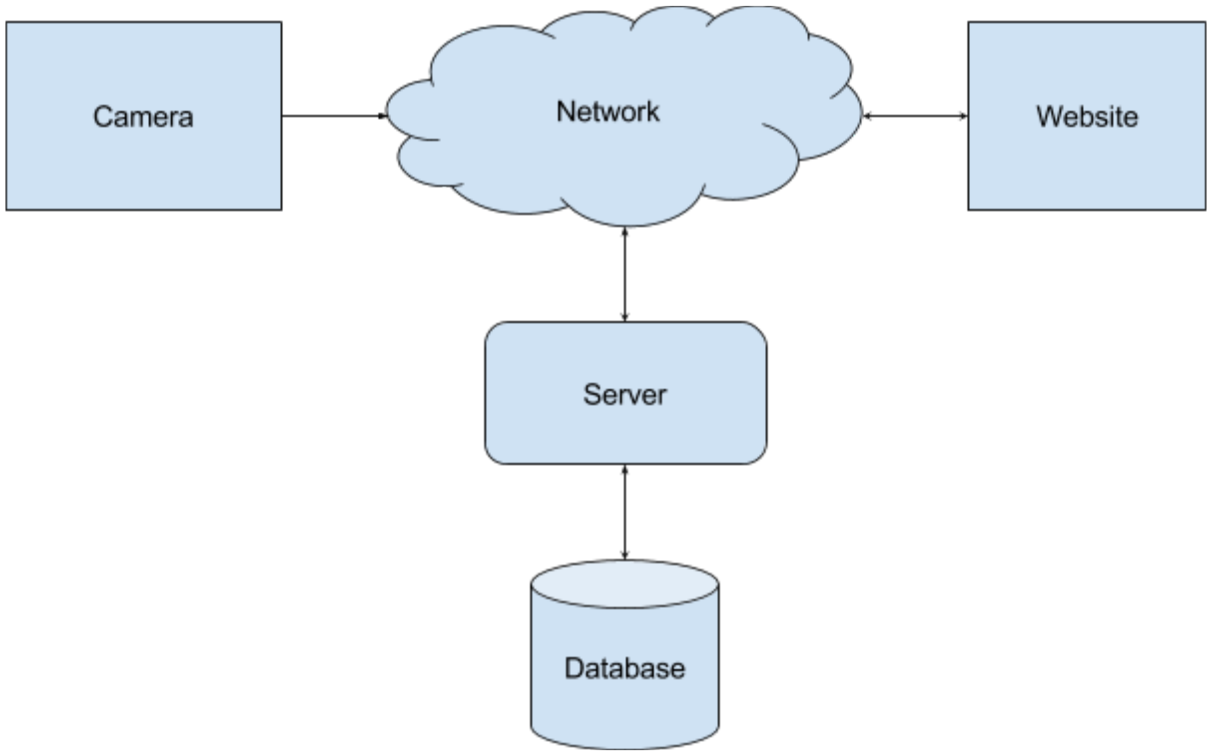
The second client in our solution is a website developed using HTML, CSS and JavaScript. We did also include the jquery library to increase productivity. We chose to create a website since it allows easy access from anywhere and on any device. The client's purpose is to illustrate the detected motion by the camera over a specified time period to the user. This time period can be set in the client and the data, in JSON, for that period is received from the server and passed to a Google Chart Timeline to make the data more comprehensive. We think the timeline chart illustrates the data to the user in a simple and easy to understand manner.

Protocol

When some kind of motion in front of the camera presents it self, a HTTP notification is send from the camera to our Nodejs server. The server then processes the notification by saving a timestamp on when the notification was received to a Mongodb. This then allows the website client to retrieve the data from the database using Ajax HTTP requests. When the data, in JSON, is retrieved we show the data to the user with the help of Google Charts Timeline. Regarding that every detected motion only is illustrated for one second in the timeline is related to that the camera only sends a response when the motions started. We would need to conduct more investigation on the Axis Video Motion Detection to see if this is supported. Maybe the interface provides a way to also send a response when the motion stops or maybe we could retrieve the information some other way, for example though a request to a log file on the camera or something similar.

Discussion about the solution

We think that the client-server solution is indeed the best solution for this setup. Because the fact that the camera has to detect motion, then log it and be able to show it to the our users.



The system could be designed in different ways tho. One direction would be to instead of a separate server and database we could have configured the camera to create some kind of log file on the camera, and then let the client access the data in the file directly on the camera. This would give us a problem with cross original access problem. Which means that the client will be calling from a different ip address than the camera. In our solution the “middleware” (Nodejs server) also hosts our website client which means that this cross original access problem will not occur.

The reason we chose Nodejs to be our server platform is because it allows us to develop in one language (JavaScript) regardless of where we are writing code. Also Nodejs is built on the principle of asynchronous which allows callback, this is very powerful as well as in our type of application saves a lot of development time. This together with the database Mongodb, results in a full-stack JavaScript solution. No need to learn other languages and the same IDE can be used for everything.

Discussion about the assignment

We think that the assignment is interesting and also open for us to be a bit creative. It gives us the opportunity to develop a solution however we want, as long as the assignment is solved. Since we had a seminar on how to use SoupUI with the Axis Vapix, it would be interesting to try that out. We did find that it was harder to learn SoupUI with the user guide from Axis and implement it, than just coming up with our own solution. So on that part we would like some more in depth and practical description on how it works, and why we should use it. Also we think the assignment was too small to implement something this advanced. For this small communication it feels that a simple REST-like web service doing HTTP transaction is sufficient enough. In short we didn't think that the project architecture benefited from SoupUI.