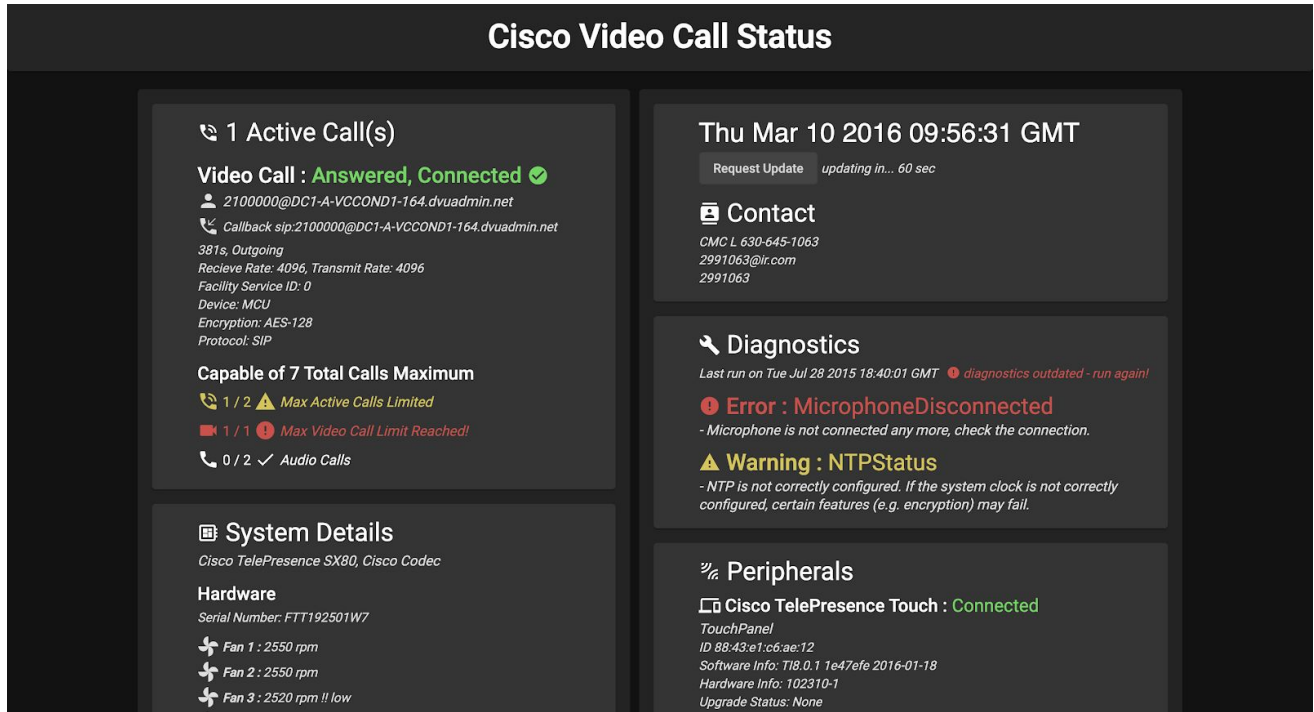


IR Graduate Developer Assignment Report

Michael Merola, michaelr.merola@gmail.com

Solution

My solution utilizes *Angular* and *Node.js* to read in the `status.xml` file and display critical and pertinent information on a sleek client-side utility. Green notices indicate good/connected status, yellow indicates a warning, and red a critical error that requires attention by the user.



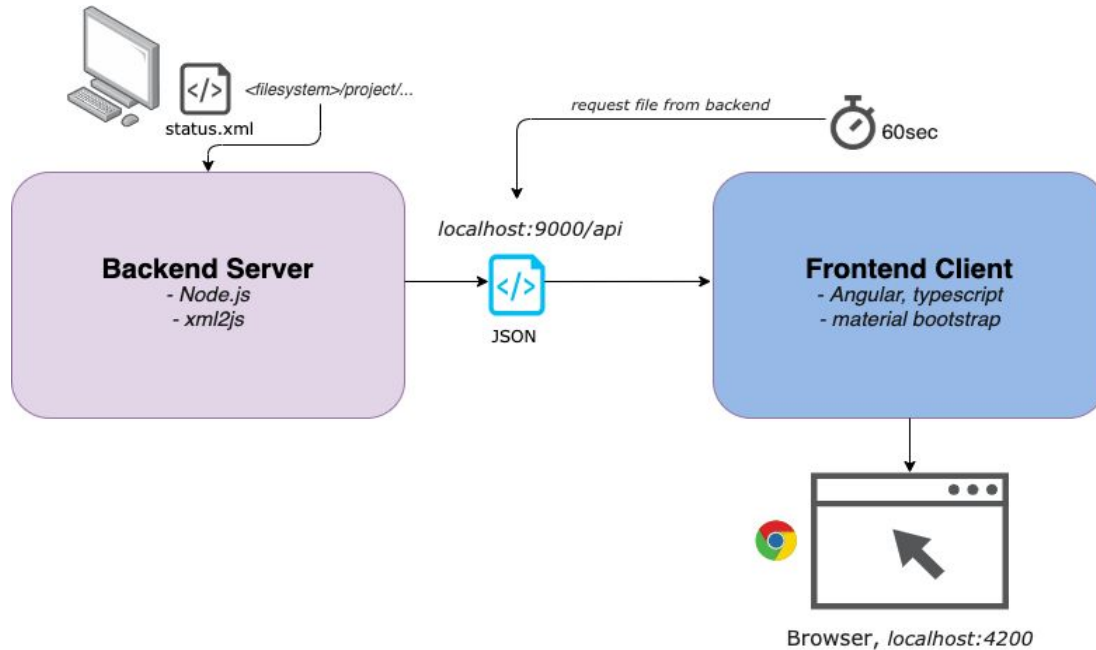
The screenshot displays the 'Cisco Video Call Status' web application. The interface is dark-themed and organized into several panels:

- Top Header:** 'Cisco Video Call Status' in white text on a dark background.
- Left Panel (Call Status):**
 - 1 Active Call(s)**: Shows a video call as 'Answered, Connected' with a green checkmark. Details include: 2100000@DC1-A-VCCOND1-164.dvuadmin.net, Callback sip: 2100000@DC1-A-VCCOND1-164.dvuadmin.net, 381s, Outgoing, Receive Rate: 4096, Transmit Rate: 4096, Facility Service ID: 0, Device: MCU, Encryption: AES-128, Protocol: SIP.
 - Capable of 7 Total Calls Maximum**: Shows 1/2 calls active (yellow warning icon) and 1/1 video call limit reached (red error icon). Audio calls are 0/2.
- Right Panel (System & Diagnostics):**
 - System Details:** Cisco TelePresence SX80, Cisco Codec. Hardware: Serial Number: FTT192501W7. Fans: Fan 1 (2550 rpm), Fan 2 (2550 rpm), Fan 3 (2520 rpm, low).
 - Timestamp:** Thu Mar 10 2016 09:56:31 GMT. Request Update: updating in... 60 sec.
 - Contact:** CMC L 630-645-1063, 2991063@ir.com, 2991063.
 - Diagnostics:** Last run on Tue Jul 28 2015 18:40:01 GMT. Shows a red error: 'Error : MicrophoneDisconnected' (Microphone is not connected) and a yellow warning: 'Warning : NTPStatus' (NTP is not correctly configured).
 - Peripherals:** Cisco TelePresence Touch : Connected. Details: TouchPanel, ID 88:43:e1:c6:ae:12, Software Info: T18.0.1 1e47efe 2016-01-18, Hardware Info: 102310-1, Upgrade Status: None.

Technology

I chose to approach this assignment by developing a full-stack web app. I chose Node.js as a backend because it is a standard technology in web development that is very useful for handling data requests and easily serves most frontend frameworks. Angular was a natural choice for the frontend because it provides a flexible component system for creating a clean, usable client. The two technologies integrate smoothly and maintain javascript across the stack. Developing with this stack also increases scalability, as it is simple to add REST functionality to request the .xml file or later add on a database such as mongoDB.

Design



The solution uses Node to read `status.xml` from the file system, and I used the common node module '[xml2js](#)' to convert to a JSON file before making the data available to the frontend via express api (see `<project>/api/routes/data-api.js`). Because my app is javascript full-stack, it is much more simple to parse through the data as JSON which has native support in both Node and Angular. On the frontend, the 'data-display' component handles all functionality including requesting the data from the api, parsing the JSON object for relevant information, and displaying data on the client via html/css (see `<project>/client/src/app/data-display/..` for relevant ts, html, and css files). The app will request the data from the backend and reparse every 60 seconds or when the user clicks the request button.

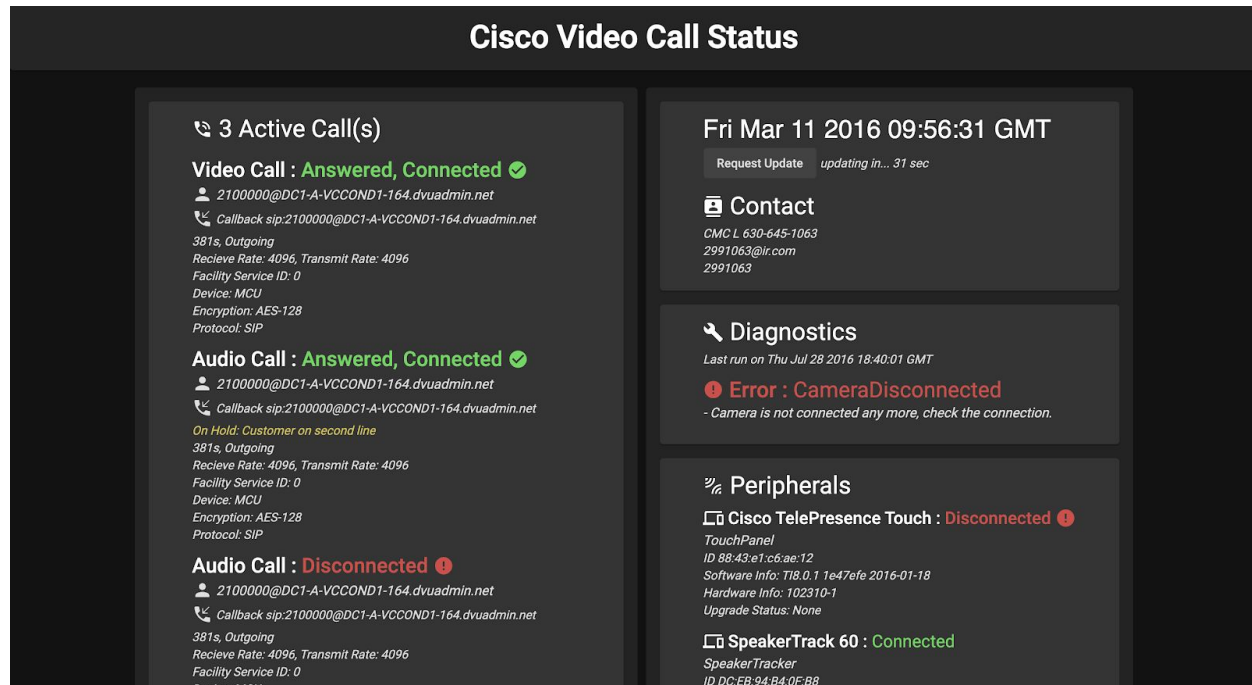
Thu Mar 10 2016 09:56:31 GMT

Request Update updating in... 59 sec

For the style of the app, I opted to follow Google's guidelines on dark-themed material design (<https://material.io/design/color/dark-theme.html>). I utilized the Angular CLI's material module, which includes a variety of pre-designed css components, in order to create the basic structure and feel of the app (modules used: `MatCard`, `MatIcon`, `MatButton`).

Testing

To test that my solution will dynamically display the data received, I created a duplicate xml file (test.xml, included in project) with edited attributes. The data altered includes multiple call objects with varying status, different diagnostics, system time, and peripherals. The screenshot below highlights how the app handles different changes in the data.



Improvements

Ideally, the app would request the .xml response from a REST endpoint and not the filesystem, which would be easily implemented through the built Node.js backend. If I had more time, I would further the frontend by making it more interactive. For example, the ability to collapse big data blocks to reduce clutter, or introduce a tab system to switch between viewing different aspects of the system.

Relevant Scripts

```
<project>/api/status.xml (test.xml)
<project>/api/routes/data-api.js .xml Handler / API (Node.js / Express)
<project>/client/src/app/data-display/...
- data-display.component.ts      main functionality (typescript)
- data-display.component.html
- data-display.component.css
```

Steps to Deploy Solution

- 1) Please ensure that you have installed both *Node.js* and *Angular* on your system.

Node.js: <https://nodejs.org/en/download/>

Angular: \$ npm install -g @angular/cli

- 2) Download attached .zip from email, or clone my repo from GitHub.

\$ git clone https://github.com/MichaelMerola/ir-dev-project.git

- 3) Open two tabs in your terminal:

Tab 1

\$ cd <project>/app/api (backend folder)

\$ npm install (installs all necessary packages)

\$ npm start

* the api will start writing data from .xml to <http://localhost:9000/data-api>

Tab 2

\$ cd <project>/app/client (frontend folder)

\$ npm install (installs all necessary packages)

\$ ng serve

- 4) The app will now be running at <http://localhost:4200/> in your browser.

Included Files

- app/
 - folder containing source code
 - api/ *backend Node.js src*
 - client/ *frontend Angular src*
- screenshots/
 - final app running in browser
 - screenshot of app using test file
- Design Diagram and initial handwritten notes
- Project Report .pdf

Thank you for your consideration!