Software Developer - Data Acquisition and Integrations @ Romeo Power

Take Home Challenge

📦**Project file download**

👋 Hello {{name}}!

Thank you for taking the time to interview with Romeo Power! We are looking forward to seeing what you can produce in the next 2 hours. The goal is to demonstrate your skills at understanding a new problem and show off your creativity. Please complete the challenge as much as you can.

You are free to use the programming language of your choice however Python & Javascript preferred. We highly recommend you use a language that you are the most comfortable with, even if not Python or JS. You are free to use any 3rd party packages. The packages listed below will allow you to interface with a WebSocket server.

Recommended Websocket Packages

- Javascript: ws: a Node.js WebSocket library
- Python: Websocket Client
- Python: WebSockets
- Online Websocket Tool: WebsocketKing
- Others: Awesome Websockets

We encourage you to ask questions in your code, make comments on your thought process, and express what you would have done given more time.

Every candidate will receive feedback, within 2 - 3 business days.

In this challenge you will be interfacing with an electric torque driver which has the ability to communicate with a client via websockets. Accessible @ wss://romeo-power-code-challenge.herokuapp.com

An electric torque driver is a device that is used to fasten screws in a manufacturing environment. Here is a quick video on how these work: https://www.youtube.com/watch?v=dF4buzcQEGE.

The tool in this challenge is 100% fictional, and does not represent how a real torque driver communicates.

You will create a console application that will allow a user to input a serial number and begin to fasten several screws.

A given serial number is related to one part number + revision, and a given part number + revision may have one or more steps, representing multiple screws. Each screw has a different torque value expectation and an acceptable range (min, max).

> Each step represents 1 screw, all of which must be done in order.

Your tool should be able to:

1. Accept a valid Serial Number
2. Set the expected torque value
3. Properly handle the value in which the screw was torqued, sent to the client as an "operation" event. The events below are 2 operation events:

```
{
    "type": "operation",

    "data": {
        "direction": "forward",
        "currentValue": 19.513,
        "configuredValue": 19
    }
}
{
    "type": "operation",
    "data": {
        "direction": "reverse",
        "configuredValue": 0,
        "currentValue": 0
    }
}
```

4. A record should be created storing the serial number, step number, actual torque value, and timestamp. You can store this onto a JSON file to make it simple, or you can store it to an SQLite DB, etc (Please attach your storage file).

5. On a reverse operation your program should delete the previously stored value and re-send the step to the driver.

- If the fastening falls within the accepted range your tool should inform the user and continue to the next step.
- If the fastening falls outside the accepted range your tool should inform the user and require them to retry the fastening by reversing and rescrewing.

6. On complete you should inform the user and return a table with the recorded values.

Sample Interaction

```
* RUN PROGRAM *
Please Input a valid serial number:
...

Sending step #1 to torque driver.
Received 10.098 - In Range

Sending step #2 to torque driver.
Recieved 14.091 - Out of range.
Please Re-Do

Sending reverse to torque driver.
Received reverse confirmation.

Sending step #2 to torque driver.
...

Job Complete:
Step 1: 10.098
Step 2: 13.15
Step 3: 14.01
* GRACEFULLY EXIT *
...
```

Expectations

1. Use the JSON data files **HERE** to consume data. You should not modify the data in these files.
2. Accept a serial number from the user then verify that such serial is valid and exists.
3. Gracefully handle errors.
4. Include all instructions required to run your code, including requirements file, package.json , etc.
5. Try to stay within the 2 hour time limit, at or around the 2 hour limit if you have not finished write notes on what you would have done next if you had more time.

   > If you really need an extra 30 minutes please take them.

6. ZIP your project (excluding package dependencies) and upload via the unique Green House link in the email sent to you.

Accepted Events

The "device" accepts the following messages: (Hint: JSON must be sent as string)

```
// Sample Events
// Clears current set torque value.
{
  "type": "reset"
}
//Sets torque value.
{
  "type": "setTorque",
  "data": {
    "direction": "forward",
    "value": 12.10
  }
}
 //Sets torque value.
{
  "type": "setTorque",
  "data": {
    "direction": "reverse"
  }
}
// Request the current status of the device
{
  "type": "status"
}
```

Finished Early?

1. Create a menu that allows the user to look up a serial number's actual torque values.
2. Store all data (including the provided JSON files) in a SQLite database file.
3. Feel free to be creative.

Notice

🤫**As we plan to use this project for other candidates we kindly ask that you reframe from publicly sharing your code.**

Find a bug? 🐛

Submit it to alejandro@romeopower.com