Candidate,

You have 2 weeks to complete and submit the challenge. It should take you a several hours or days depending on your current skill level. Every member of our team has completed this challenge and universally agrees this is a far better representation of engineering skill than white board tests. Good luck! The team is looking forward to seeing your skills.

Task Description:

We do most of our frontend work in Qt/QML, so the frontend coding challenge is in QML. In addition, you will be implementing a basic WebSocket server using nodejs.

Create a simple QML application which communicates to a WebSocket server. Each QML window instance can send and receive messages from the server. See screenshot and video for an example layout (you are not required to precisely replicate this visual design). To make the WebSocket simple you need to support at least two clients talking to each other as shown on the video.

Extra Credit: support multiple clients connecting to the server and one to one conversation with any connected client.

Other Requirements:

- You have two weeks to complete the challenge.
- All code shall be submitted to a GitHub account.
- We will review and discuss your code in follow-up interviews; be prepared to explain your code and justify decisions made.

Notes:

- This challenge is not meant to "trick" or "confuse" you in any way.
  It is exactly how I would pose work to one of my engineers.
  Do not make it more complex than it really is.
- Code clarity, consistency, organization and correctness are all evaluated.
- Use all resources at your disposal. You are allowed to ask appropriate questions.
- For reference, this challenge took me 2.5 hours to complete at a base level, expect it to take longer without previous Qt/QML experience, and don't forget all phases below.

What we are looking for:

- Coding ability (aka: what you learned in school)
- Desire to demonstrate your skills (motivation and drive)
- Ability to take loosely defined requirements and fill in the gaps required to accomplish the task. This is not school, this is professional software engineering. Tasks are often vague and exploratory in nature. In school the answer is already known by the professor and you are solving their "puzzle". Professional Engineers rarely get to solve puzzles already completed. Most tasks are unknown.
- Clarity and organization - Engineering software projects are large and complex and you must be able to communicate highly technical ideas and code in a clear and concise manner so the rest of the team can help you and add to your efforts.

Recommended Workflow ----------

Environment Setup:

Create a Qt account and download the Qt installer from https://www.qt.io/download (open source version), use it to install a 5.12.X version of Qt (i.e. 5.12.10, NOT 5.14.X, 6.X) which will include the QtCreator IDE. Use the attached ChatChallenge project to get started in QtCreator after install. Note: select needed packages when installing Qt. For windows users ie. MSVC 2017 64-bit if you have VS installed or MinGW 7.3.0 64-bit. These are pre-built Qt libraries and you still need to have a compiler. For more info see: https://doc.qt.io/qt-5/windows-requirements.html#compilers

Multiple phases are recommended because they prevent you from becoming overwhelmed and distracted during Phase 1 with details that don't matter initially. In later phases, you KNOW it works and you can make modifications and clean up one step at a time and re-verify working state each iteration.

Phase 1: Basic Functionality and Technology Exploration (just get it working)

1. Lay out your base visual components in QML. I.e. use container Rectangle elements with borders to help visualize your initial application control layout:
   ```
   border {
      color: "red";
      width: 1
   }
   ```
   Think of this as your "floor plan" for your new house. Where are you going to put your kitchen, bedrooms?
2. Start with one element and get it capturing and displaying text into your console as well as your text output areas. Skip going through your nodejs server at this point.
3. Implement your nodejs server. Make sure you can send and receive messages.
4. Test and assure you have it all working

Phase 2: Refine and Improve (professionalize)

1. Make names (variables, files et al) and file organization clear
2. Clean up clunky code and document
3. Improve performance and code clarity
4. Break out the QML into separate components/files to make the application easier to implement and read (this has been started in the project with main.qml and ChatView.qml).

Phase 3: Complete the WebSocket implementation

1. Design a simple messaging architecture. Think about how are you going to send out the information
2. Implement send and receive based on the messaging architecture
3. Make sure to connect everything by now and have a complete working project

Hint: use JSON between QML and the server for easier data access. JSON.parse() and JSON.stringify() work on both sides use them when needed.

Resources (you may not need all of these) ----------

Qt/QML Introduction Book:
https://qmlbook.github.io/

Integrating QML and C++
http://doc.qt.io/qt-5/qtqml-cppintegration-topic.html

Qt Signals and Slots
http://doc.qt.io/qt-5/signalsandslots.html

QML Signal Handling
http://doc.qt.io/qt-5/qtqml-syntax-signals.html

// INPUTS
QML Button
https://doc.qt.io/qt-5.10/qml-qtquick-controls2-button.html

QML TextField
https://doc.qt.io/qt-5/qml-qtquick-controls2-textfield.html

// OUTPUTS
https://doc.qt.io/qt-5/qml-qtquick-textedit.html

Visually customizing QML Controls
https://doc.qt.io/qt-5/qtquickcontrols2-customize.html

WebSocket:
Lots of examples available in the source code
https://github.com/websockets/ws

Hints ----------

To help with the chat transcript and formatting- This will make a TextEdit area that scrolls and allows HTML formatted text:

```
ScrollView {
   id: chatTranscriptScroll
   anchors {
      // your anchors here
   }

   TextEdit {
      id: chatTranscriptText
      width: chatTranscriptScroll.width
      readOnly: true
      textFormat: Text.RichText // enables HTML formatting
      wrapMode: TextEdit.Wrap
   }
}
```

Another VERY useful hint/technique- Add the following semi-transparent Rectangle to any other QML component and it will shade it “tomato” color so you can see its exact position and size, as well as log what it believes is its height and width, to make debugging layout and element positioning easy:

```
// insert anywhere inside another QML element to visualize its placement.
Rectangle {
   color: "tomato"
   opacity: .15
   anchors {
      fill: parent
   }
   z:20
   Component.onCompleted: {
      console.log("height: " + height + "\\n     width:  " + width)
   }
}

// Example usage:
Rectangle {
   id: chatWindow1
   …
   …
```

```
    // will highlight entire chatWindow1 with “color” to show its boundaries and
    // location on screen.
    Rectangle {
        color: "tomato"
        opacity: .15
        anchors {
            fill: parent
        }
        z:20
        Component.onCompleted: {
            console.log("height: " + height + "\\n     width:  " + width)
        }
    }
}
```

Use QML Controls 2 control elements - Controls 1 elements are deprecated, you'll know you're using one if you import Qt.Quick Controls 1.x. Note that not all Controls 1 controls exist in Controls 2 and vice versa (but most are).