Coding Take-home Project - Full Stack

This project is designed to provide you the opportunity to demonstrate your approach to solving a full-stack development task. While the technical problem may not be overly complex, we are looking to see how you interpret the problem and expect you to write code which validates the correct behavior.

The problem statements below make recommendations based on our technology stack. The purpose is not make you learn new languages or tools; if you are more comfortable using another language and/or toolset please do so. Keep in mind that you should document your assumptions and steps thoroughly. You can assume a Linux-based platform when documenting any build steps or run instructions.

Successful candidates must complete levels 1 and 2; levels 3 and 4 are for included to provide candidates further opportunity to distinguish themselves. Feel free to attempt part of these levels and submit what is working. All completed code and documentation should be submitted through your GitHub account.

Level 1

Create a web service that returns a randomized 9x9 grid of integers in the range [1,9] that represents a solved Sudoku board.[1]

Web Service API

Request:
```
GET /sudoku/board
```

Response:
```
Content Type: application/json
Body:
[1,2,3,….,9] //Array of 81 integers
```

Technology Recommendations

We recommend using the following technologies:

- Programming Language: Javascript
- Build Tools: npm
- Libraries/Tools:
    - Node.js

[1] Refer to Sudoku if you are unfamiliar with the rules of the game.

- Mocha
- Sinon

Acceptance Criteria

1. Entire solution can be built using a single command (eg. ./npm build)
2. Solution contains appropriate level of unit testing, which is run as part of the build.
3. When the service is running locally, it should be accessible with a request similar to:

```
> curl http://localhost:8080/sudoku/board
```

4. The entire HTTP request must return in less that 500ms (when issues from local host)
5. The resultant Sudoku boards must always be valid, solved solutions according to the rules of the game.
6. All steps to build and run are documented.

Level 2

Create a single-page application that displays the Sudoku board web server created in Level 1.

Technology Recommendations

We recommend using the following technical stack:

- Programming Language: Javascript
- Build Tools: npm
- Libraries/Tools:
  - HTML5
  - CSS
  - React
  - Nginx

Acceptance Criteria

1. SPA web application must be built using a single command (eg. ./npm build)
2. Solution contains appropriate level of unit testing, which is run as part of the build.
3. Nginx should be used to act as a reverse proxy so that the SPA and web service backend appear as one web site on the local machine.
4. When the web page first loads, it should load the first board.
5. Display a progress spinner when the web page is loading a board.
6. The UI will contain a button called “Reload” that will refresh the board (reload a new board from the server).
7. The resultant Sudoku boards must always be valid, solved solutions according to the rules of the game.
8. All steps to build and run are documented.

Level 3

Extend the SPA to support minor interactions with the Sudoku board and updated results.

Technology Recommendations

Same as Level 2.

Acceptance Criteria

1. SPA web application must be built using a single command (eg. ./npm build)
2. Nginx should be used to act as a reverse proxy so that the SPA and web service backend appear as one web site on the local machine.
3. The user interface allows the user to select a cell. When cell is selected it will be highlighted.
4. Cell selection will act like a toggle (on/off) when it is clicked.
5. If a cell is "toggled on" and the *Reload* button is pressed, the requested board will be new and randomized BUT the selected cell value will be the same (*Note: the backend will have to be adjusted accordingly)*
6. The resultant Sudoku boards must always be valid, solved solutions according to the rules of the game.
7. All steps to build and run are documented.

Level 4

This is an optional section for candidates that have some familiarity with Docker. We would like you to "dockerize" the web service and front-end components you have built.

Technology Recommendations

- Docker

Acceptance Criteria

1. Web service component, including docker container, must be built with one command and should generate a docker image called "sudoku-ws" with the tag "level-4".
2. The web service docker container must be run via the command:

```
> docker run –it –p 8080:8080 sudoku-ws:level-4
```

3. When the docker container is running, the following command should return the sudoku board:

```
> curl http://localhost:8080/sudoku/board
```

4. SPA web application, including docker container, must be built with one command and should generate a docker image called “sudoku-spa” with the tag “level-4”.
5. Nginx should be used to act as a reverse proxy so that the SPA and web service backend appear as one web site on the local machine (Nginx can be integrated into the sudoku-spa docker image).