Take home coding challenge

The purpose of this challenge is to assess your coding ability in a task that is representative of the type of work you'll be doing at Endla.

This task should be unfamiliar and require you to use your problem-solving skills to come up with a solution that delivers the desired result for the end user. We look for problem solving skills that show you can tackle real-world technical challenges like those you'll encounter in daily work.

Submission instructions

Please use git and GitHub during this task and submit your code.

Create a **private** GitHub repository with the supplied code on the main branch and add your changes to a new branch.

Once completed **open a PR to the main branch** and submit by inviting GitHub users *rileyodon* and *michael1997* to your repository.

Confidentiality

Don't worry, we haven't put anything secret in here but please don't share or publish this challenge or your solution to protect the integrity of the challenge for other candidates.

Time

You should be able to complete the task in <2 hours and please spend no more than 4 hours.

Evaluation criteria

Your code will be assessed against the following criteria:

- Correctness
- Readability
- Code style
- Maintainability

Feedback

Following our review of your submission we will conduct a brief feedback session. During this session we will:

- Discuss your approach to solving the problem (please keep any notes to aid this)
- Dive deeper into an aspect of the problem
- Share our feedback
- Hear feedback from you.

Task description

A customer has provided us with some drilling data contained in *src/data/wirelineData.csv* that they would like to visualise. Utilising the boiler plate included develop the code required to complete the following features. The boiler plate is a simplified version of our frontend stack with no backend connectors. There are no restrictions on what libraries you can use to assist.

Your submitted code should result in an interactive plot being shown at http://localhost:3000/.

1. **Load the data from a csv**

There is some sample data in *src/data/wirelineData.csv*. Load this data into json so that it can be used and displayed in the application.

Feel free to add and dependencies to the project to help with loading the data.

You can find some extra information about the data at the end of this document under the title 'Data Description.'

2. **Return the coordinates for a given measured depth.**

We often need to know what the coordinates are at a given measured depth. Write a function that will return X,Y,Z coordinates for the input measured depth, azimuth and inclination.

3. **Plot the well path**

We want to get a 3D visualisation of the well path given the wireline data.

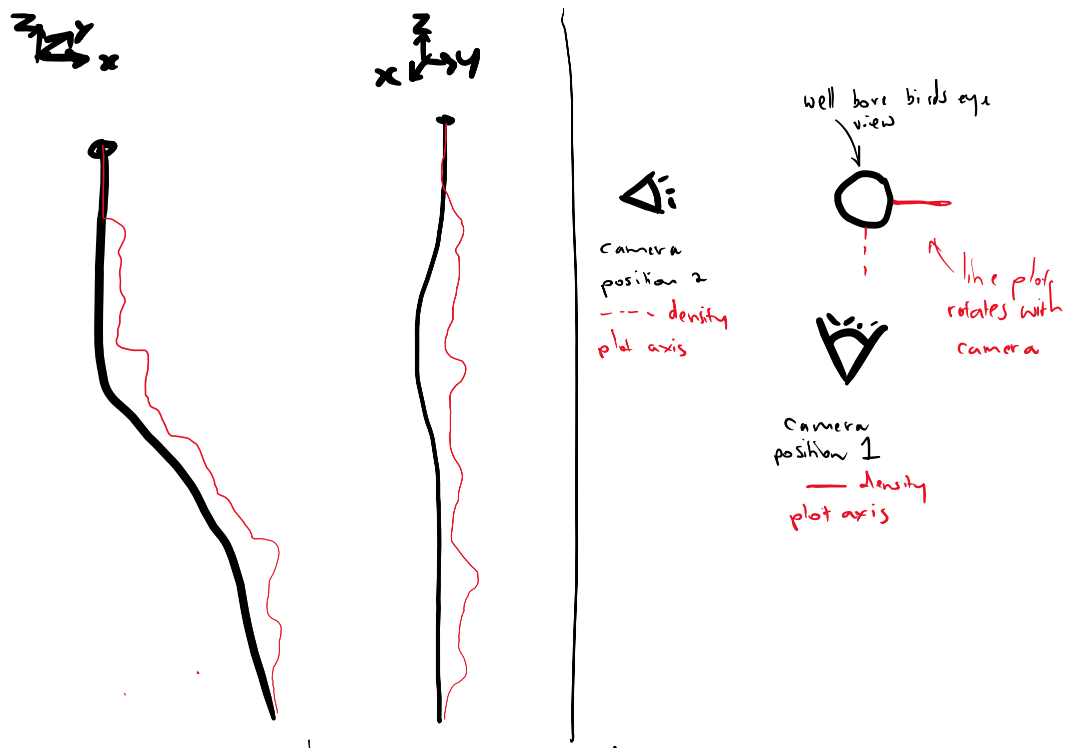
We have written some boilerplate with a dummy well to visualise a the well in the frontend. When you startup the app you'll see the dummy example at http://localhost:3000/.

Edit the code in *src/components/WellProfile.tsx* to display the data loaded from the csv instead.

4. Challenge

We'd like to also see the density log as a line graph adjacent to the well path in the 3D visualisation. The line should always be visible to the camera as we orbit and pan around the well, and the zero for the line graph should be the edge of the well path. The boilerplate code includes the frontend handling needed to make this work.

The sketch below illustrates the end goal – excuse the roughness but this is often how ideas are communicated between engineers:



— Well path. Here shown from 2 different viewing angles.
— density log

Because this needs to be responsive this code will need to be written in the frontend with Typescript. We've written all the boilerplate you should need and your job is mainly to figure out mathematically how to make this work.

The file of interest is *frontend\src\containers\WellProfile.tsx* Within this file the lineplot is created with *function LineGraph (props).* Within this function we've shown how you can modify the position of the line plot. You'll need to change *function exampleLineCalc(dataPoint)* to actually calculate the correct position to point each data point to form the line graph.

Data description

The data included (*src/data/wirelineData.csv*) is dummy wireline data. Wireline data is collected after a well is drilled and provides valuable information for the operator to then install the completion and forecast the well's production.

Some points to consider with the data:

- sampling is relatively high frequency and relatively consistent (e.g. samples are taken every ~1m down to the bottom of the hole)
- the top and bottom of the well will have missing data (e.g. wireline data could be missing for depths <50m and the last 15m of the well)

For the purpose of this task assume the total depth is always 15m deeper than the last wireline data point and the top of the well is:

Measured depth (m)	Inclination (deg)	Azimuth (deg)
0	0	0