Background
The Xtract platform makes heavy use of JSON-structured configurability to allow us to customise our journeys to match the data our customers want us to collect. In the full platform, the frontend will load both the company configuration information and the specific incident data from the backend. The configuration information is then used to dynamically render the required form fields, while the incident data is used to pre-populate form fields, and can be added to or overwritten when the form is submitted.

Task Overview
For this exercise, we will be asking you to build a configurable form with some different field types, and dynamically manipulate incident data based on the form input values. To get you started, we will be providing:

- A statically rendered React & Material-UI form
- A specification for a set of fields to be dynamically rendered
- A sample configuration for you to work with while building the renderer
- Some pretend incident data to be added to, used as default values, and overwritten, by the form

This starting point can be found here:
https://codesandbox.io/p/sandbox/staticformrenderer-864sn4

The form you produce should be able to handle reasonable changes to the provided configuration or input data based on the spec provided, including:

- Adding or removing fields
- Reordering fields
- Changing where a field should read default values from or write inputted data to
- Renaming fields

On submission, the form will display the updated data on screen, reflecting any changes made within the form.

Where appropriate, we would like you to improve the design of the form using additional components from Material UI.

Staged Objectives
We have defined various stages in the development below, for you to work through, each adding new features to the dynamic renderer. You are welcome to tackle the steps out of order, if you prefer.

1. Create a dynamic renderer that can render text fields (`textInput`) like those provided in the order specified by the configuration, and updating the mock incident data based on form input values
2. Using values from the mock incident data to render default values for fields using a lookup path specified by the config, and updating existing values on submission
3. Add the `integerInput` field type that allows integer values between a maximum and a minimum to be input

4. Add the `enumInput` field type, that renders a dropdown component based on the list of possible values provided
5. Add the `currencyInput` field type, that displays both a currency selector dropdown, and an integer input. The fields should appear together on a single line, and should store their data together as a structured object (see input sample)
6. Add the ability for some fields to be required to allow submission. This should be reflected visually, and be enforced upon submission
7. Add the ability for some fields to be displayed conditionally based on values in the input data, or provided elsewhere in the form (prior to submission!)

What we are looking for
We are first and foremost looking for code that is cleanly written and easy to understand. We will also be looking at the functionality of the code, and seeing whether it can handle the configuration changes described in the task overview. The renderer should also gracefully discard any aspects of the specification that it does not support.

We will judge the code against only the levels which have been completed, and ask that you spend no more than three hours on the exercise. We do not expect you to have completed every level in that time, and progress through the steps will not be a major consideration in the decision making. They are provided such that you should always be able to fill three hours.

Next Steps
Once the candidate feels they have reached the state they wish to submit either a forked Code Sandbox or a git repository. Please send the link to Joe either via Cord or by e-mail to joe.hitchen@xtract360.com

In the interview, we will ask you to describe the code you have written and how it works. We will also be looking to see it working with both the sample configuration provided with the exercise, and a similar configuration we will share with you. Off the back of this, we will ask you various questions about the code, and how you would modify it in future to support extra features.

Configuration Specification

The configuration for a single page consists of a list of field specifications. For this exercise, there will be four field types:

A standard text input

```
  {
    "type": "textInput",
    "label": "This is a text input", // The text to be displayed
    "path": "store.the.data.here" // The path within the data structure to
read data from and write data to
```

```
    },
```

An numerical integer input, with upper and lower bounds

```
    {
      "type": "integerInput",
      "label": "This is a integer input", // The text to be displayed
      "path": "store.the.data.here", // The path within the data structure to read data from and write data to
      "min": 0, // Optional, defaults to 0 - Sets the min number that can be assigned
      "max": 100 // Optional, defaults to infinite - Sets the max number that can be assigned,
    },
```

A dropdown enum selector field

```
    {
      "type": "enumInput",
      "label": "This is a enum input", // The text to be displayed
      "path": "store.the.data.here", // The path within the data structure to read data from and write data to
      "values": ["yes", "no", "maybe"] // Acceptable values
    },
```

A currency amount input, which has both a currency selector and an integer input as a combined field

```
    {
      "type": "currencyInput",
      "label": "This is a currency input", // The text to be displayed
      "path": "store.the.data.here", // The path within the data structure to read data from and write data to.
      "min": 0, // Optional, defaults to 0 - Sets the min number that can be assigned
      "max": 100, // Optional, defaults to infinite - Sets the max number that can be assigned,
      "currencies": ["EUR", "USD", "GBP"] // List of the allowed currencies that can be selected
    }
```

Currency data itself should be stored as a combined object in the data structure

```
{ "currency": "EUR", "value": 3554.5 }
```