Tech Test Documentation / Overview of completed work

The project is built using Create React App with Typescript. React version 18.1, Typescript 4.6.4 Node 14.17.3, and npm 6.14.16

For styling, the choice was sass (version 1.52.1). Generally, all of the code in the project is related to the given task - creating a reusable Form component. There are two parts of the code:

1. Typescript code related to the reusable Form component (**Form.tsx** and **InputComponent.tsx**) and the sass files for its styles (forms have feelings too, they love to dress up and look decent, and hate being ugly and naked)

2. Supporting code such as **utils.ts**, general styles in **App.scss**, assets/images folder that stores background image, **custom.d.ts** file for typing jpg image (had a type not found error when importing the jpg image into the component, so I needed to make this file)

App.tsx
Generally, it's good practice to keep App.tsx as clean as possible since it's the root of our application.

However, in this particular test, due to the simplicity of the app, I decided to use the **Form** component directly in the **App.tsx** along with the code related to adding particular form contents we want (input fields defined in **InputComponent.tsx**) and desired behavior that happens when we submit the form (adding the desired method to the onSubmit prop, to define what happens when user clicks on submit button).

Later on, during code refactoring, I decided to move that code to **utils.ts** after all, since it isn't closely related to the form core logic and functionality. It presents the changeable, custom data that we provide to the Form to tailor the reusable, generic Form component into the one suited for our needs. That "demo code" was clogging the App.tsx file, so I decided that it was better to move it to utils.ts.

Form.tsx and **InputComponent.tsx**

Reusable form component, the spotlight of this task. As per the requirements, users will only call this form component, providing it with the data about the input fields and with the function they wish to trigger on submitting the form.

They don't have contact with what happens under the hood with the input fields, the onChange handlers of input fields, and storing the data in the component state.

The form component is in its base a regular <form> component that accepts the following props:

- **inputFields** - array of objects containing data about input fields (name, label, placeholder, etc.)
- **onSubmit** - a function that's going to be triggered when user submits the form. In this example we use a function that persists our form data to local storage, but, of course, we could also submit the data to the backend using, for example, Fetch API with Post method.
- **formTitle -** a name of the form, to be displayed above
- **defaultValues** - default values of the input fields;

The component stores the values of its fields as an object in state, using useState hook. It also defines the onChange method, in common with all the InputComponents, that handles what happens when input is changed. In our case, when we change the input, we call the set method from useState, to set the value from the field into the state.

Except for the "submit" text in the button (which can also be changed to be editable and more generic), the Form component doesn't have any hardcoded values. Everything is used from prop variables, making this component completely reusable. We just pass our values as a prop and the magic happens!

We map through the array of objects containing data about input fields, assigning an **InputComponent** for each of the array values. That custom data, given by the user describing the input fields, is passed to the InputComponent through the following props (input field object properties):

- **placeholder**;
- **label**;
- **type**;
- **name**;

There are also a couple of optional properties, which are there for validation purposes. So, if the users want to make some of the input fields fulfill some kind of condition, like the proposed wooga.name, they can do it here:

- **pattern**? - regex that defines needed condition;
- **errorMessage**?;
- **required**?

I concluded that it would be a pretty straightforward approach for users to only provide the data about form inputs in an object, specifying the data about input fields they need, and then **Form.tsx** and **InputComponent.tsx** do the rest.

Encountered problem: When mapping through inputFields, of course, react asked for a unique key for each item. For that purposes, inside the utils.ts I added Universal Unique Identifier generator function (uuidv4). However, I experienced a bug - losing focus of input field after each keystroke. I realized it's because calling the generator function on each re-render, so react sees a new key each time. So instead of seeing the same key and reusing the object with that key in the DOM, the new key causes React to regenerate the view, making the input field lose focus. The **solution** is to generate keys on the first render using useEffect hook, and assign them to the object. That way we'll always have the same keys.

Validation
Validation was implemented in a way that was suggested in the requirements.

InputComponent.tsx has an optional pattern prop where the user can specify the regex needed for the validation.