Popular external libraries

In this reading you will learn about some well-designed UI libraries, such as ChakraUI, that can speed up your application delivery.

You will also explore how to simplify your form design with tools like Formik, and write declarative validation rules with chain operators using Yup.

Chakra UI

UI libraries are a great way to speed up the development process. They provide a set of robust, well-tested and highly configurable pre-built components that you can use to create your applications. Those components act as atoms or building blocks, laying the foundation to create more complex components.

One of the most popular UI solutions is Chakra UI. Chakra UI is a simple, modular and accessible component library that provides you with the building blocks you need for your React applications.

Chakra groups its different components by categories, like layout, forms, data display, feedback, typography or overlay.

Layout components are in charge of setting virtual delimiters or boundaries for your content. They also manage how their children are laid (row or column) and the spacing between them among other properties. Some layout components to highlight are:

- HStack and VStack: they display children using flex properties and stack elements horizontally or vertically respectively. Both take a spacing prop that allows you to set the spacing between the elements.
- Box: it allows you to create a box with a background color, border, shadow, etc. It takes a bg prop that allows you to set the background color.

Typography is also an important category that is worth mentioning. There are two main components from this group:

- Heading: renders one of the different DOM header tags (`h1`, `h2`, `h3`...). It takes a size prop that allows you to set the size of the heading and an as prop to specify the particular semantic HTML tag.

```
<Heading as='h2' size='2xl'>
    Little Lemon
 </Heading>
```

- Text: is used to render text and paragraph within an interface. It offers a `fontSize` prop to increase the font size of the text.

```
<Text fontSize='lg'>Best restaurant in town</Text>
```

In order to see all the different component categories and the different props each component accepts, you can check the official documentation page.

Style props

Chakra uses style props to provide css directives directly as props to the different components. You can find a reference of all the available style props in the Chakra UI documentation.

As a general rule, you can consider *camelCase* versions of css styles to be valid style props. But you can also leverage the shorthand version. For example, instead of using **`backgroundColor`**, you can use **`bg`**.

```
<Box backgroundColor='tomato' /> is equivalent to <Box bg='tomato' />
```

Putting all together, the below example represents three boxes stacked in a row, with a vertical space of 16px between boxes, where each box has a height of 40px and a different background color, as well as a particular number as its children:

```
<HStack spacing="16px">
  <Box h='40px' bg='yellow.200'>
    1
  </Box>
  <Box h='40px' bg='tomato'>
    2
  </Box>
  <Box h='40px' bg='pink.100'>
    3
  </Box>
</HStack>
```

Formik and Yup

Formik is another popular open-source library that helps you to create forms in React. The library takes care of the repetitive tasks of managing the state of the form, validation and submission, so you can focus on the business logic of your application. It does so by providing a set of components and hooks that you can plug into your forms.

Yup is a JavaScript open-source library used to validate the form data before submitting it to the server. It provides a set of chainable operators that you can apply to your form fields to declaratively specify the validation rules.

Formik comes with built-in support for schema based form-level validation through Yup, so they work together seamlessly.

The most important component from Formik is the **`useFormik`** hook. This hook handles all the different states of your form. It only needs a configuration object as an argument.

Let's break down the hook usage with some code example:

```
import * as Yup from 'yup';
import { useFormik } from 'formik';

// The below code would go inside a React component
const {
  values,
  errors,
  touched,
getFieldProps,
handleSubmit,
} = useFormik({
```

```
  intialValues: {
      comment: "",
    },
  onSubmit: (values) => {
      // Handle form submission
    },
  validationSchema: Yup.object({
      comment: Yup.string().required("Required"),
    }),
  });
```

The `useFormik` hook takes an object as an argument with the following properties:

- `initialValues`: An object with the initial values of the form fields
- `onSubmit`: A function that will be called when the form is submitted, with the form values as an argument. In that example you could access the message via `values.comment`.
- `validationSchema`: A Yup schema that will be used to validate the form fields. In that example, the message is a field with a string value that is required. As you can see the rules are human-readable and easy to understand.

The hook returns an object with the following properties:

- `values`: An object with the current values of the form fields. In that example you could access the message via `values.comment`.
- `errors`: An object with the current errors of the form fields. If validation fails for the "comment" field, which would be the case if the input has been touched and its value is empty, according to the validation schema, you could access the message error via errors.comment. In that particular case, the message error would be "Required". If the field is valid though, the value will be undefined.
- `touched`: An object with the current touched state of the form fields. You can use this to determine if a field has been touched (focused at least once) or not. In that example, you could access the comment state via `touched.comment`. If the field has been touched, the value will be true, otherwise it will be false.
- `getFieldProps`: A function that takes a field name as an argument and returns an object with the following properties:
  - `name`: The field name.
  - `value`: The current value of the field.
  - `onChange`: The `handleChange` function.
  - `onBlur`: A function that will be called when the field is blurred. It updates the corresponding field in the touched object.

The way you would use this function is by spreading the returned object into the input element. For example, if you had an input element with the name "comment", you would do something like this:

```
<input {...getFieldProps("comment")} />
```

- `handleSubmit`: A function that will be called when the form is submitted. It takes an event as an argument and calls the `onSubmit` function with the values object as an argument. You should hook this function to the form `onSubmit` event.

Conclusion

In this reading, you have learned about three of the most popular libraries that can save you precious time during your app development. Their main goal is to take care of mundane and repetitive tasks and let you focus on the stuff that matters.

You were introduced to Chakra UI as a way to leverage well designed components that you can put together to build more complex interfaces.

Finally, you gained knowledge about an open-source library called Formik and its perfect companion, Yup, to create complex React forms with ease.