**Solution code**

Here is the completed solution code for the App.js file:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

import { ChakraProvider } from "@chakra-ui/react";

import Header from "./components/Header";

import LandingSection from "./components/LandingSection";

import ProjectsSection from "./components/ProjectsSection";

import ContactMeSection from "./components/ContactMeSection";

import Footer from "./components/Footer";

import { AlertProvider } from "./context/alertContext";

import Alert from "./components/Alert";

function App() {

 return (

   <ChakraProvider>

     <AlertProvider>

       <main>

         <Header />

         <LandingSection />

         <ProjectsSection />

         <ContactMeSection />

         <Footer />

         <Alert />

       </main>

     </AlertProvider>

   </ChakraProvider>

 );

}

export default App;





Here is the completed solution code for the context/alertContext.js file:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

import {createContext, useContext, useState} from "react";

const AlertContext = createContext(undefined);

export const AlertProvider = ({ children }) => {

  const [state, setState] = useState({

    isOpen: false,

    type: 'success',

    message: '',

  });

  return (

    <AlertContext.Provider

      value={{

        ...state,

        onOpen: (type, message) => setState({ isOpen: true, type, message }),

        onClose: () => setState({ isOpen: false, type: '', message: '' }),

      }}

    >

      {children}

    </AlertContext.Provider>

  );

};

export const useAlertContext = () => useContext(AlertContext);





Here is the completed solution code for the components/Header.js file:

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

         py={4}

         justifyContent="space-between"

         alignItems="center"

       >

         <nav>

           <HStack spacing={8}>

             {socials.map(({ icon, url }) => (

               <a

                 key={url}

                 href={url}

                 target="\_blank"

                 rel="noopener noreferrer"

               >

                 <FontAwesomeIcon icon={icon} size="2x" key={url} />

               </a>

             ))}

           </HStack>

         </nav>

         <nav>

           <HStack spacing={8}>

             <a href="#projects" onClick={handleClick("projects")}>

               Projects

             </a>

             <a href="#contactme" onClick={handleClick("contactme")}>

               Contact Me

             </a>

           </HStack>

         </nav>

       </HStack>

     </Box>

   </Box>

 );

};

export default Header;





Here is the completed solution code for the components/Card.js file:

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

6

7

3

4

5

1

2

     <VStack

       color="black"

       backgroundColor="white"

       cursor="pointer"

       borderRadius="xl"

     >

       <Image borderRadius="xl" src={imageSrc} alt={title} />

       <VStack spacing={4} p={4} alignItems="flex-start">

         <HStack justifyContent="space-between" alignItems="center">

           <Heading as="h3" size="md">

             {title}

           </Heading>

         </HStack>

         <Text color="#64748b" fontSize="lg">

           {description}

         </Text>

         <HStack spacing={2} alignItems="center">

           <p>See more</p>

           <FontAwesomeIcon icon={faArrowRight} size="1x" />

         </HStack>

       </VStack>

     </VStack>

   );

};

export default Card;

const Card = ({ title, description, imageSrc }) => {

   return (

import { faArrowRight } from "@fortawesome/free-solid-svg-icons";

import React from "react";

import { Heading, HStack, Image, Text, VStack } from "@chakra-ui/react";

import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";





Here is the completed solution code for the components/Alert.js  file:

32

33

34

35

36

37

     </AlertDialogOverlay>

   </AlertDialog>

 );

}

export default Alert;





Here is the completed solution code for the components/Footer.js file:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

import React from "react";

import {Box, Flex} from "@chakra-ui/react";

const Footer = () => {

 return (

   <Box backgroundColor="#18181b">

     <footer>

       <Flex

         margin="0 auto"

         px={12}

         color="white"

         justifyContent="center"

         alignItems="center"

         maxWidth="1024px"

         height={16}

       >

         <p>Pete • © 2022</p>

       </Flex>

     </footer>

   </Box>

 );

};

export default Footer;





Here is the completed solution code for the components/FullScreenSection.js file:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

import \* as React from "react";

import { VStack } from "@chakra-ui/react";

/\*\*

\* Illustrates the use of children prop and spread operator

\*/

const FullScreenSection = ({ children, isDarkBackground, ...boxProps }) => {

 return (

   <VStack

     backgroundColor={boxProps.backgroundColor}

     color={isDarkBackground ? "white" : "black"}

   >

     <VStack maxWidth="1280px" minHeight="100vh" {...boxProps}>

       {children}

     </VStack>

   </VStack>

 );

};

export default FullScreenSection;





Here is the completed solution code for the components/LandingSection.js file:

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

19

20

21

16

17

18

14

15

11

12

13

8

9

10

6

7

3

4

5

1

2

       />

       <Heading as="h4" size="md" noOfLines={1}>

         {greeting}

       </Heading>

     </VStack>

     <VStack spacing={6}>

       <Heading as="h1" size="3xl" noOfLines={1}>

         {bio1}

       </Heading>

       <Heading as="h1" size="3xl" noOfLines={1}>

         {bio2}

       </Heading>

     </VStack>

   </VStack>

 </FullScreenSection>

);

export default LandingSection;

         src="https://i.pravatar.cc/150?img=7"

         size="2xl"

         name="Your Name"

   <VStack spacing={16}>

     <VStack spacing={4} alignItems="center">

       <Avatar

   backgroundColor="#2A4365"

 >

   justifyContent="center"

   alignItems="center"

   isDarkBackground

const LandingSection = () => (

 <FullScreenSection

const bio1 = "A frontend developer";

const bio2 = "specialized in React";

import FullScreenSection from "./FullScreenSection";

const greeting = "Hello, I am Pete!";

import React from "react";

import { Avatar, Heading, VStack } from "@chakra-ui/react";





Here is the completed solution code for the components/ProjectsSection.js file:

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

     spacing={8}

   >

     <Heading as="h1" id="projects-section">

       Featured Projects

     </Heading>

     <Box

       display="grid"

       gridTemplateColumns="repeat(2,minmax(0,1fr))"

       gridGap={8}

     >

       {projects.map((project) => (

         <Card

           key={project.title}

           title={project.title}

           description={project.description}

           url="https://github.com/rgommezz/react-native-offline"

           imageSrc={project.getImageSrc()}

         />

       ))}

     </Box>

   </FullScreenSection>

 );

};

export default ProjectsSection;





Here is the completed solution code for the components/ContactMeSection.js file:

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

             <FormControl isInvalid={!!formik.errors.comment && formik.touched.comment}>

               <FormLabel htmlFor="comment">Your message</FormLabel>

               <Textarea

                 id="comment"

                 name="comment"

                 height={250}

                 {...formik.getFieldProps("comment")}

               />

               <FormErrorMessage>{formik.errors.comment}</FormErrorMessage>

             </FormControl>

             <Button type="submit" colorScheme="purple" width="full" isLoading={isLoading}>

               Submit

             </Button>

           </VStack>

         </form>

       </Box>

     </VStack>

   </FullScreenSection>

 );

};

export default ContactMeSection;





Here is the completed solution code for the hooks/useSubmit.js file:

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

   try {

     await wait(2000);

     if (random < 0.5) {

       throw new Error("Something went wrong");

     }

     setResponse({

       type: 'success',

       message: `Thanks for your submission ${data.firstName}, we will get back to you shortly!`,

     })

   } catch (error) {

     setResponse({

       type: 'error',

       message: 'Something went wrong, please try again later!',

     })

   } finally {

     setLoading(false);

   }

 };

 return { isLoading, response, submit };

}

export default useSubmit;





In a previous video, you were introduced to a possible solution for the portfolio page, where most of the concepts you learned over the duration of this course were applied in one way or another. However, there are still some interesting extras about the solution that will be illustrated in this reading.

**Header animation**

In the Header.js component, there are two React core hooks being used: **useRef** and **useEffect**.

Those two are used in conjunction to achieve the smooth animation of the header. If you run the application, you can see that the header hides when I am scrolling down, and shows up when I am scrolling back up.

To implement this behavior, I have to use a side effect and subscribe to the scroll event on the window object using **window.addEventListener**.

It’s important to remove all subscriptions before the unmounting phase. For that, I have to return a function inside useEffect that performs that task. That’s the **window.removeEventListener** call you see executed inside that function.

1

2

3

4

5

6

7

8

9

10

11

useEffect(() => {

  const handleScroll = () => {

    // Business logic

  };

  window.addEventListener('scroll', handleScroll);

  return () => {

    window.removeEventListener('scroll', handleScroll);

  }

}, []);





To animate the header, you need to deal with its underlying DOM node and apply some style transition. Do you recall the React way to do that? If you said **useRef**, you guessed right! That’s what I am doing on the container **Box** and **headerRef** holds a reference to the underlying **<div>** node.

8

9

10

11

12

13

6

7

3

4

5

1

2

      {...}

    >

      …

    </Box>

  );

};

    <Box

      ref={headerRef}

  …

  return (

const Header = () => {

  const headerRef = useRef(null);





Finally, **handleScroll** is the handler function that will be called every time there is a change in the vertical scroll position.

The meat of this function resides in the comparison between the previous value and the new value. That determines the direction of the scroll and which style I should apply in order to either show or hide the header. Since I am using transition properties in the container **Box** component, the change is animated.

40

37

38

39

35

36

32

33

34

41

    </Box>

      ref={headerRef}

    >

     …

      transitionTimingFunction="ease-in-out"

      backgroundColor="#18181b"

      translateY={0}

      transitionProperty="transform"

      transitionDuration=".3s"

  );





**Header navigation**

There is another neat trick I would like to show you, which also happens in the Header component.

Let’s see what happens when I click on one of the header sections. Do you see how it nicely animates and scrolls into its position on the page? Let me show you how simple it is to implement something like that. Coming back to the code, I have this **handleClick** function that is invoked when I click on one of the header navigation items, either Projects or Contact Me.

1

2

3

4

5

6

7

8

9

10

const handleClick = (anchor) => () => {

  const id = `${anchor}-section`;

  const element = document.getElementById(id);

  if (element) {

    element.scrollIntoView({

      behavior: "smooth",

      block: "start",

    });

  }

};





I have defined some **id**s in other sections of the page. For instance, the header of the projects section has an **id** called project-section. The **handleClick** function is called with the anchor name depending on where the navigation should happen, as per the code below:

1

2

3

4

5

6

7

8

<HStack spacing={8}>

  <a href="#projects" onClick={handleClick("projects")}>

    Projects

  </a>

  <a href="#contactme" onClick={handleClick("contactme")}>

    Contact Me

  </a>

</HStack>





To access that DOM element, you can then use **document.getElementById** and pass the corresponding ID. Once you have it, you can call **element.scrollIntoView** with an object as parameter, setting behavior as smooth and block start. Nice and simple, isn’t it?

**Formik and Yup validation**

[Formik](https://formik.org/docs/overview) works very nicely with [Yup](https://github.com/jquense/yup), an open source library that allows you to define validation rules in a declarative way. Let’s break down in detail the rules set for the Contact Me form, as part of the **useFormik** hook. **useFormik** hook comes with a **validationSchema** option as part of its configuration object.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

const formik = useFormik({

  initialValues: {

    firstName: "",

    email: "",

    type: "hireMe",

    comment: "",

  },

  onSubmit: (values) => {

    submit('https://john.com/contactme', values);

  },

  validationSchema: Yup.object({

    firstName: Yup.string().required("Required"),

    email: Yup.string().email("Invalid email address").required("Required"),

    comment: Yup.string()

      .min(25, "Must be at least 25 characters")

      .required("Required"),

  }),

});





For the **firstName** field, the rule states that it has to be a string and it can’t be empty. If empty, Formik will register an error message with the label “Required”.

1

firstName: Yup.string().required("Required"),





The email input is also required. Observe how Yup already provides us with common validators out of the box, like one to verify that what users type is a valid email. If incorrect, Formik will register an error on that input with the error message “Invalid email address”. Quite straightforward right?

1

email: Yup.string().email("Invalid email address").required("Required"),





Finally, I am making the comment field mandatory, with a minimum length of 25 characters.

1

2

3

comment: Yup.string()

 .min(25, "Must be at least 25 characters")

 .required("Required"),





Go to next item

**Completed**