File Edit Selection View Go Run ··· dummy-folder

App.js | index.js | index.html | BookPT.js | ShowBooking.js | package.json | Exercises.js

src > App.js > ...

```
function App() {
  return (
    <>
      <BrowserRouter>
        <Navbar />
        <Routes>
          {/*
            Add the routes as mentioned below:
              1. It should redirect to Exercises component for default URL i.e. http://localhost:3000/.
              2. It should load BookPT component for /book_pt/:exerciseType.
              3. It should load ShowBooking component for /show_booking/:exerciseType.
              4. It should redirect to Exercises component for any wrong URL e.g. http://localhost:3000/wrongpage.
          */}
          <Route path="/" element={<Exercises/>} />
          <Route path="/book_pt/:exerciseType" element={<BookPT/>} />
          <Route path="/show_booking/:exerciseType" element={<ShowBooking/>} />
          <Route path="*" element={<Exercises/>} />


        </Routes>
      </BrowserRouter>
    </>
  );
}
export default App;
```

0 0 0 Ln 6, Col 1 Spaces: 2 UTF-8 LF {} JavaScript

BookPT.js Starts now

```jsx
import React, { useState } from "react";
import { useParams } from "react-router-dom";
import axios from "axios";

export default function BookPT() {
  const params = useParams();

  const [formData, setFormData] = useState({
    exerciseType: params.exerciseType,
    name: "",
    contactNo: "",
    startDate: "",
  });

  const [formErrorMessage, setFormErrorMessage] = useState({
    name: "",
    contactNo: "",
    startDate: "",
  });

  const [formDataValid, setFormDataValid] = useState({
    name: false, //value given to 'name' field is invalid, if false
    contactNo: false, //value given to 'contactNo field is invalid, if false
    startDate: false, //value given to 'startDate field is invalid, if false
    buttonActive: false, //the form button is disabled if set to false
  });

  const [successMessage, setSuccessMessage] = useState("");
  const [errorMessage, setErrorMessage] = useState("");
```

```
5   export default function BookPT() {
26  });
27
28  const [successMessage, setSuccessMessage] = useState("");
29  const [errorMessage, setErrorMessage] = useState("");
30
31  /* Use the messages from the below messages object
32  to display all kinds of success, error and validation messages instead of hardcoding. */
33  const messages = {
34    REQUIRED_VALIDATION: "Field is required",
35    NAME_VALIDATION: "Must be at least 3 characters long",
36    DATE_VALIDATION: "You cannot select today's date or a date in the past. Please choose a future date to begin your package.",
37    CONTACT_VALIDATION: "Cannot start with a number below 6 and must be 10 digits",
38    TRAINER_BOOKED_SUCCESS: "Personal trainer booked successfully",
39    TRAINER_BOOKED_ERROR: "Something went wrong",
40  };
41
42  const url = "http://localhost:4000/bookings";
43
44  /* To Be Implemented. */
45  /* This function will be invoked whenever the value provided to any of the input fields changes */
46  const handleChange = (event) => {
47    /*
48      1. use the name and value property from the event object.
49      2. set the corresponding formData state property to the received value.
50      3. invoke the validateField by passing event object's name and value as parameter.
51    */
```

```
42  const url = "http://localhost:4000/bookings";
43
44  /* To Be Implemented. */
45  /* This function will be invoked whenever the value provided to any of the input fields changes */
46  const handleChange = (event) => {
47    /*
48      1. use the name and value property from the event object.
49      2. set the corresponding formData state property to the received value.
50      3. invoke the validateField by passing event object's name and value as parameter.
51    */
52
53    const { name, value } = event.target;
54    setFormData((prevFormData) => ({ ...prevFormData, [name]: value }));
55    validateField(name, value);
56  };
57
58  /* To Be Modified. */
59  /* This method validates the input fields based on the mentioned conditions. */
60  const validateField = (fieldName, fieldValue) => {
61    const newFormDataErrorMessages = { ...formErrorMessage };
62    const newFormDataValid = { ...formDataValid };
63
```

```
    Common requirement for all the inputs as follows:
    1. If input is valid then
      a. set the associated error message as empty string.
      b. set the associated formValid value as true,
    2. Otherwise set associated formValid value as false.
  */
  switch (fieldName) {
    case "name":
      const nameRegex = /^[A-Za-z][A-Za-z ]{3,29}$/;
      /*
        1. If the value of name is empty,
            set the associated error message as REQUIRED_VALIDATION.
        2. If the value contains less than 3 alphabets or greater than 29 alphabets or
           if the value contains other than alphabet characters,
            set the associated error message as NAME_VALIDATION.
      */
      if (fieldValue === "") {
        newFormDataErrorMessages.name = messages.REQUIRED_VALIDATION;
        newFormDataValid.name = false;
      } else if (!nameRegex.test(fieldValue)) {
        newFormDataErrorMessages.name = messages.NAME_VALIDATION;
        newFormDataValid.name = false;
      } else {
        newFormDataErrorMessages.name = "";
        newFormDataValid.name = true;
      }
      break;
```

```
92
93      case "startDate":
94        let today = new Date().setUTCHours(0, 0, 0, 0);
95        let checkIn = new Date(fieldValue).setUTCHours(0, 0, 0, 0);
96        /*
97          1. If startDate field is empty it should set the error message as REQUIRED_VALIDATION.
98          2. If startDate field has past date or today's date,
99             it should set the error message as DATE_VALIDATION.
100       */
101       if (fieldValue === "") {
102         newFormDataErrorMessages.startDate = messages.REQUIRED_VALIDATION;
103         newFormDataValid.startDate = false;
104       } else if (checkIn <= today) {
105         newFormDataErrorMessages.startDate = messages.DATE_VALIDATION;
106         newFormDataValid.startDate = false;
107       } else {
108         newFormDataErrorMessages.startDate = "";
109         newFormDataValid.startDate = true;
110       }
111       break;
112
```

```
112
113     case "contactNo":
114       /*
115         1. If contactNo field is empty it should set the error message as REQUIRED_VALIDATION.
116         2. If contactNo field has fieldValues starting from a number less than 6 and
117             the number of digits is less than ten,
118             it should set the error message as CONTACT_VALIDATION.
119             Hint: Use Regex pattern.
120       */
121       const contactNoRegex = /^[6-9][0-9]{9}$/;
122       if (fieldValue === "") {
123         newFormDataErrorMessages.contactNo = messages.REQUIRED_VALIDATION;
124         newFormDataValid.contactNo = false;
125       } else if (!contactNoRegex.test(fieldValue)) {
126         newFormDataErrorMessages.contactNo = messages.CONTACT_VALIDATION;
127         newFormDataValid.contactNo = false;
128       } else {
129         newFormDataErrorMessages.contactNo = "";
130         newFormDataValid.contactNo = true;
131       }
132       break;
133
134     default:
135       break;
136   }
```

```
137
138     /*
139       Set isButtonActive to true when the associated formValid property of all three input fields are true.
140     */
141
142     if (newFormDataValid.name && newFormDataValid.contactNo && newFormDataValid.startDate) {
143       newFormDataValid.buttonActive = true;
144     } else {
145       newFormDataValid.buttonActive = false;
146     }
147     setFormErrorMessage(newFormDataErrorMessages);
148     setFormDataValid(newFormDataValid);
149   };
150
```

```
/* To Be Implemented. */
/* This function will save the booking form data in the backend. */
const bookPT = async (event) => {
  /*
    1. Should prevent default behaviour on form submission.
    2. Should make a POST request to http://localhost:4000/bookings with formData
     as the request body.
    3. In case of success, it should set successMessage as TRAINER_BOOKED_SUCCESS.
    4. In case of error, it should set errorMessage as TRAINER_BOOKED_ERROR.
  */
  event.preventDefault();
  try {
    const response = await axios.post(url, formData);
    setSuccessMessage(messages.TRAINER_BOOKED_SUCCESS);
    setErrorMessage("");
  } catch (error) {
    setErrorMessage(messages.TRAINER_BOOKED_ERROR);
    setSuccessMessage("");
  }


};
```

```
/* To be modified. */
/* The below code designs registration form. */
return (
  <>
    <div className="container-fluid">
      <div className="row">
        <div className="col-md-6 offset-md-3">
          <div
            className="card mt-5"
            style={{ backgroundColor: "rgba(255, 255, 255, 0.75)" }}
          >
            <div className="card-header text text-center">
              <h3 id="card-title" data-testid="form-title">
                Book Your Trainer for: {params.exerciseType}
                {/* Display selected Exercise Type */}
              </h3>
            </div>
            <div className="card-body">
              {/*
```

```
The common requirements for each input field for the following form are mentioned below:
1. On form submission, bookPT method should be invoked.
2. Bind all form fields to their corresponding properties in the formData state.
3. Give all form fields a name attribute that matches their formData state property.
4. Set proper type attributes and bootstrap classes for all form fields.
5. Invoke the handleChange() method on every input field value change.
6. contactNo field should be of type number.
7. startDate field should be of type date.
8. The "Book PT" button should have bootstrap classes "btn btn-block btn-dark mt-3".
9. Disable the button until the buttonActive state property is true.
10. Show error message and success message on form submission as appropriate, but not both at the same time.
*/}
                <form data-testid="bookingForm" onSubmit={bookPT}>
                  <div className="form-group" id="exerciseType">
                    <label
                      htmlFor="exerciseType"
                      data-testid="labelExerciseType"
                    >
                      Personal Trainer For
                    </label>
                    {/* input1 - Do not change.*/}
                    <input
                      id="exerciseType"
                      name="exerciseType"
                      className="form-control"
                      value={formData.exerciseType}
                      type="text"
                      disabled
```

```
211             {/* input1 - Do not change.*/}
212             <input
213               id="exerciseType"
214               name="exerciseType"
215               className="form-control"
216               value={formData.exerciseType}
217               type="text"
218               disabled
219               data-testid="inputExerciseType"
220             />
221           </div>
```

```
222             <div className="form-group" id="name">
223               <label htmlFor="name" data-testid="labelName">
224                 Name
225               </label>
226               {/* input2 */}
227               <input
228                 id="name"
229                 type="text"
230                 data-testid="inputName"
231                 name="name"
232                 value={formData.name}
233                 onChange={handleChange}
234                 className="form-control"
235
236               />
237               <div id="nameError" data-testid="errorName" className="badge text-bg-danger">
238                 {/* Show name error message  and style it with class "badge text-bg-danger". */}
239                 {formErrorMessage.name}
240               </div>
241             </div>
```

```jsx
<div className="form-group" id="contactNo">
  <label htmlFor="contactNo" data-testid="labelContactNo">
    Contact No
  </label>
  {/* input3 */}
  <input
    id="contactNo"
    data-testid="inputContactNo"
    type="number"
    name="contactNo"
    value={formData.contactNo}
    onChange={handleChange}
    className="form-control"
  />
  <div id="contactNoError" data-testid="errorContactNo" className="badge text-bg-danger">
    {/* Show contact number error message and style it with class "badge text-bg-danger". */}
    {formErrorMessage.contactNo}
  </div>
</div>
```

```
261          <div className="form-group" id="startDate">
262            <label htmlFor="startDate" data-testid="labelDate">
263              Package Start Date
264            </label>
265            {/* input4 */}
266            <input id="startDate"
267            data-testid="inputDate"
268              name="startDate"
269              type="date"
270              value={formData.startDate}
271              onChange={handleChange}
272              className="form-control"/>
273            <div id="startDateError" data-testid="errorDate" className="badge text-bg-danger">
274              {/* Show date error message and style it with class "badge text-bg-danger". */}
275              {formErrorMessage.startDate}
276            </div>
277          </div>
```

```
                    </div>
                    <div className="form-group" id="btnForm" >
                      {/* button */}
                      <button id="bookPTBtn" data-testid="bookPTButton"
                        className="btn btn-block btn-dark mt-3"
                        disabled={!formDataValid.buttonActive}>
                        Book PT
                      </button>
                    </div>
                  </form>
                  <div id="successMessage" data-testid="successMessageForm" className="badge text-bg-success">
                    {/* Display success message here and style it with class "badge text-bg-success". */}
                    {successMessage}
                  </div>
                  <div id="errorMessage" data-testid="errorMessageForm" className="badge text-bg-danger">
                    {/* Display error message here and style it with class "badge text-bg-danger". */}
                    {errorMessage}
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </>
    );
}
```

Exercises.js starts now

```jsx
import React, { useEffect, useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";

export default function Exercises() {
  const [exercisesArray, setExercisesArray] = useState([]);
  const [errorMessage, setErrorMessage] = useState("");
  const navigate = useNavigate();

  const messages = {
    NETWORK_ERROR: "Network Error",
  };
  const url = "http://localhost:4000/exercises";
  /* To Be Implemented. */
  /* This method fetches all the exercises from url i.e. http://localhost:4000/exercises. */
  const getAllExercises = async () => {
    /*
      1. Send an AXIOS GET request to the url and handle the success and error cases appropriately.
      2. In case of success, assign received response's data to exercisesArray state.
      3. In case of error, set the errorMessage state as NETWORK_ERROR from the messages object.
    */
    try {
      const response = await axios.get(url);
      setExercisesArray(response.data);
    } catch (error) {
      setErrorMessage(messages.NETWORK_ERROR);
    }
  };
```

```
export default function Exercises() {
  /* Implemented - No Changes Required. */
  useEffect(() => {
    getAllExercises();
  }, []);

  /* To Be Implemented */
  /* This method navigates to BookPT component. */
  const handleClickBooking = (exercise) => {
    /*
      1. Navigate to BookPT component with route params as "exerciseType" of the
      selected exercise.
      Note : Use the navigate method.
    */
    navigate(`/book-pt/${exercise.exerciseType}`);

  };

  /* To Be Implemented */
  /* This method navigates to ShowBooking component. */
  const handleClickShowBooking = (exercise) => {
    /*
      1. Navigate to ShowBooking component with route params as "exerciseType" of
      the selected exercise.
    */
    navigate(`/show-booking/${exercise.exerciseType}`);
```

```
/* To Be Modified. */
/* This method iterates over the exercisesArray to display all the exercises cards. */
const displayExercises = () => {
  let counter = 1000;
  /*
    Start iteration over the exercisesArray state below the end of this comment block.
    return a map function to iterate over the exercisesArray and move the below counter incrementation
    and return block inside the map function.
    You are not required to change any attributes where counter is mentioned
  */
  return( exercisesArray.map((exercise) => {
    counter++;
    return (
      <div className="col-md-3" key={counter}>
        <div className="card shadow">
          {/*
              1. Set image source as imagePath.
              2. Set alternate text as id of exercise.
            */}
          <img
            data-testid={`imageT${counter}`}
            style={{ height: "200px" }}
            className="card-img-top"
            src={exercise.imagePath}
            alt={exercise.id}
          />
```

```
83          <div className="card-body gradient-buttons text-center">
84            <h4
85              className="text"
86              name="exerciseType"
87              data-testid={`exerciseTypeT${counter}`}
88            >
89              {/* Display the exerciseType of the exercise here. */}
90              {exercise.exerciseType}
91            </h4>
92            <h5
93              name="cost"
94              className="text-success"
95              data-testid={`costT${counter}`}
96            >
97              {/* Display the cost of the exercise here. */}
98              Price:{exercise.cost}$/month
99            </h5>
```

```
100             {/* Invoke handleClickBooking method by passing proper exercise Object when "Book Trainer" button is clicked. */}
101             <button
102               name="bookTrainer"
103               className="btn btn-primary mt-2"
104               style={{ marginRight: "5px" }}
105               data-testid={`bookButtonT${counter}`}
106               onClick={() => handleClickBooking(exercise)}>
107               Book Trainer
108             </button>
109             {/* Invoke handleClickShowBooking method by passing proper exercise Object when "Show Booking" button is clicked. */}
110             <button
111               name="bookTrainer"
112               className="btn btn-primary mt-2"
113               data-testid={`showButtonT${counter}`}
114               onClick={() => handleClickShowBooking(exercise)}>
115               Show Booking
116             </button>
117           </div>
118         </div>
119       </div>
120     );//End of return
121     /*
122       Iteration should be ended here.
123     */
124   }));
125 };
```

```
126   return (
127     <>
128       <div className="container">
129         <h1 style={{ fontWeight: "bold", color: "black" }}>
130           Make yourself stronger than your excuses...
131         </h1>
132         <p className="lead" style={{ fontWeight: "700", color: "black" }}>
133           Book a Personal Coach on your journey to SHAPIFY yourself
134         </p>
135       </div>
136       <div className="container-fluid mt-5">
137         {errorMessage && (
138           <h2
139             className="text-dark text-center"
140             data-testid="errorMessageExercises"
141           >
142             {/* Render error message here. */}
143             {errorMessage}
144           </h2>
145         )}
146         {exercisesArray.length > 0 &&
147           (
148             <div className="row">
149               {/* Invoke the displayExercises method. */}
150               {displayExercises()}
151             </div>
152           )}
153       </div>
```

```
          Make yourself stronger than your excuses...
        </h1>
        <p className="lead" style={{ fontWeight: "700", color: "black" }}>
          Book a Personal Coach on your journey to SHAPIFY yourself
        </p>
      </div>
      <div className="container-fluid mt-5">
        {errorMessage && (
          <h2
            className="text-dark text-center"
            data-testid="errorMessageExercises"
          >
            {/* Render error message here. */}
            {errorMessage}
          </h2>
        )}
        {exercisesArray.length > 0 &&
          (
            <div className="row">
              {/* Invoke the displayExercises method. */}
              {displayExercises()}
            </div>
          )}
      </div>
    </>
  );
}
```

END