Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2025-26

Class:	TE	**Semester:**	V
Course Code:	CSC502	**Course Name:**	WC

Name of Student:	Yash Kasare
Roll No. :	27
Assignment No.:	06
Title of Assignment:	Functional components of react.
Date of Submission:	06/10/25
Date of Correction:	10/10/25

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Completeness	5	5
Demonstrated Knowledge	3	3
Legibility	2	2
Total	10	10

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Completeness	5	3-4	1-2
Demonstrated Knowledge Legibility	3	2	1
Legibility	2	1	0

Checked by

Name of Faculty : Ms. Kshitija Gharat

Signature : Gharat

Date : 10/10/25

WC.

Assignment No. 06.

1. React Component Code:

```
import React, { useEffect, useState } from "react";
function useList ()
{ const [users, setUsers] = useState([]);
  useEffect(() => {
  fetch ("https://jsonplaceholder.typeicode.com/users")
   .then ((res) => res.json ())
   . then ((data)=> setUsers (data))
   . then catch ((error) => console.error (" Error Fetching
         users: " error));
 }, []);
 return (
  <div>
  <h2> UserList </h2>
  <ul>
  {users. map ((user) => {
    <li key = {user.id}> {user. name} </li>
 ))}
</ul>
</div>
);
}
export default UserList;
```

IMPLEMENTATION :

- useEffect hook is used to perform a side effect, i.e. fetching data from an external API.

- the side effect here is the network request, http://jsonplaceholder.typicode.com/users .
- we pass an empty dependency array ([]) as the second arguement to useEffect. This ensures that the effect runs only once when the component is mounted.
- The fetch () functⁿ retrieves the user data, & when the response arrives, we update the state using setUsers (data).
- Updating state triggers a re-render, & the component displays the list of usernames inside the <ul>.
- Without useEffect, if we put fetch () directly in the component body, it would run on every render & cause an infinite loop.

OUTPUT.

UserList

- Leanne Gauden
- Ervin Howell
- Clementine Bauch.

Q.2. models / Book.js :

```
let books = [];
export function addBook (title, author) {
    const book = {id : books.lenght+1, title, author);
    books. push ( book);
    return book ;
}
export function getAllBooks () {
    return books ;
}
```

controller/bookController.js :

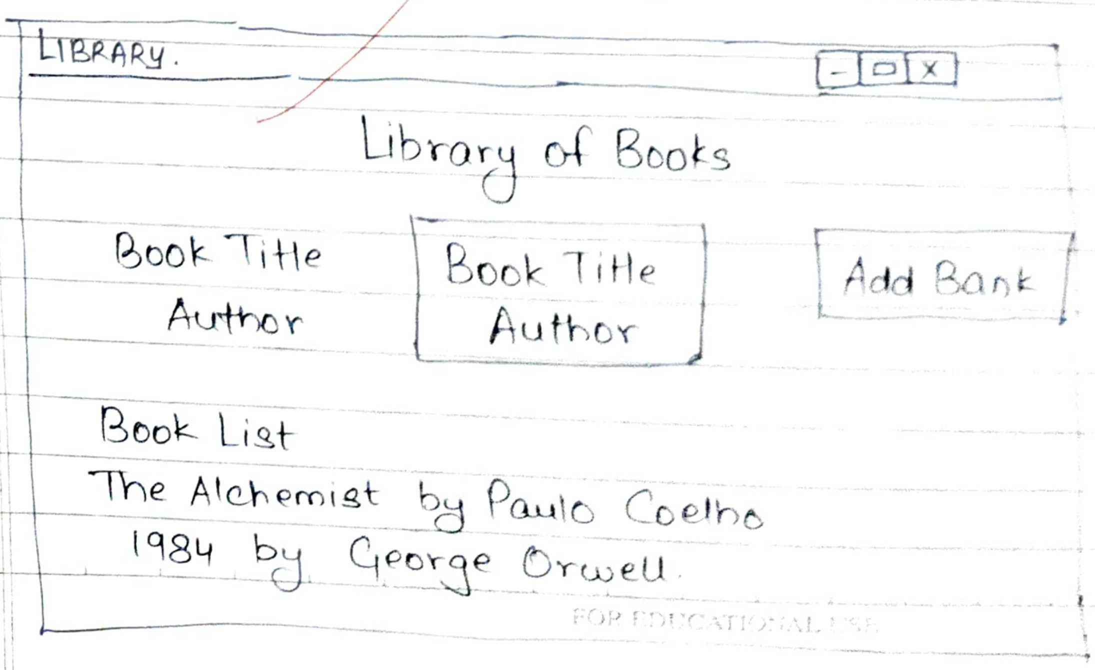
```
import { addBook, getAllBooks } from "../models/Book.js";
export function showBooks (req, res);
  const books = getAllBooks();
  res.render("books", {books});
}
export function createBook (req, res) {
   const {title, author} = req.body;
   addBook(title, author);
   res.redirect ("/books");
}
```

views/books.js :

```
<!DOCTYPE html>
<html>
<head>
  <title> Library </title>
  </head>
  <body>
    <h1> Library of Books </h1>
    <form action = "/books" method = "POST" >
      <input type = "text" name = "title" placeholder = "Book
         title" required/>
     <input type = "text" name = "author" placeholder =
        "Author" required />
     <button type = "Submit"> Add Book </button>
  </form>
  <h2> Book list </h2>
  <ul>
   <% Books.forEach (book => { % >
<li><%= book.title %> by <%= book.author %></l
<% }) %>
</ul>
</body>
</html>
```

* Server.js :

```
import express from "express";
import bodyParser from "body-Parser";
import{ showBooks , createBooks } from "./controllers/
bookController.js";
const app = express();
app.set ("View Engine", "efs");
app.use(bodyParser.urlencoderl ({extended: true}));
app.get ("/books", showBooks);
app.post ("/books", createBooks);
app.listen ( 5000 , ()=> {
    console.log ("Server running on http://localhost:5000/
                    books");
 });
```


LIBRARY.

Library of Books

Book Title
Author

Book Title
Author

Add Bank

Book List
The Alchemist by Paulo Coelho
1984 by George Orwell

	Features	MVC	FLUX	Redux
1	Data Flow	Bidirectional	Unidirectional	Uniderectional
2	State Managge-ment.	Scattered accross models.	centralized in stores.	stoingle global store.
3	Complexity handling.	Simplex Apps.	Better for complex apps.	but for large apps.
4	Predictability	Less predictable	More predict-able.	Highly predictable
5	Middleware Support.	Limited	Optional	Built-in support.

Q.4 actions/cartActions.js :

```
export const ADD_TO_CART = "ADD_TO_CART";
export const REMOVE_FROM_CART = "REMOVE_FROM_CART";
export const UPDATE_CART_ITEM = "UPDATE_CART_ITEM";
export const addToCart = (product) => ({ type: ADD_TO_CART,
    payload: product });
export const removeFromCart = (productId) => ({ type:
  REMOVE_FROM_CART, payload: productID });
const updateCartItem = (productId, quantity) => ({ type:
  UPDATE_CART_ITEM,
  payload: {productId, quantity},
});
```

```
store.js
import {createStore} from "redux";
import cartReducer from "./reducers/cartReducer";
const store = createStore(cartReducer, window.__REDUX
DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_
EXTENSION__());
export default store;

index.js
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import {Provider} from "react-redux";
import store from "./store";
const root = ReactDOM.createRoot(document.getElementById
                                    ("root"));

root.render(
      <Provider store = {store}>
      <App/>
   </Provider>
);
```

Q.5.
```
import React, {createContext, useContext, useState, useRef,
useImperativeHandle, forwardRef} from "react";
const FormContext = createContext();
export conet useForm = () => useContext (FormContext);
const FormProvider = ({children}) => {
  const [formData, setFormData] = useState({name : "",
   email : ""});
  const updateField = (field, value) => {
```

```
setFormData (prev => ({...prev, [field]: value }));
};
return (
<FormContext. Provider value = {{ formData , updateField }} >
   {children}
</FormContext. Provider>
 );
};
const NameInput = forwardRef ((props , ref) => {
  const {formData , updateField} = useform();
  const inputRef = useRef ();
 UseImperativeHandle (ref , () => ({
     focusInput : () => InputRef.current. focus ()
  }));
  return(
  <input
   ref = {inputRef}
   value = { formData. name}
  onChange = {(e) => updateField ("name", e.target. value));
   placeholder = "Enter Name" />
 );
});
const form = () => {
   const nameInputRef = useRef () ;
   const {formData} = useForm ();
   return (
   <div>
   <h2> Form </h2>
   <NameInput ref = {nameInputRef} />
   <p> Email {formData.email} </p>
```

```
<button onClick = {() => nameInputRef.current.fous.
  Input() }>
      Fous Name Input
  </button>
  </div>
 );
};
export default function App() {
   return (
    <FormProvider>
       <Form/>
    </FormProvider>
  );
}
```

BENEFITS :

- Cleaner State Management.
- Better Reusability
- Direct Component Control.
- Improved Performance.