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:	Dhanashree Gawai
Roll No. :	21
Assignment No.:	06
Title of Assignment:	Using functional components of react develop back-end application
Date of Submission:	06/10/2025
Date of Correction:	06/10/2025

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Completeness	5	5
Demonstrated Knowledge	3	3
Legibility	2	1
Total	10	9

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 :

Signature : Bharat

Date : 10/10/25

Q1t -

* React Component code :

```
import React, {useEffect, usestate} from "react";
   function uselist ()
{
   Const [users, setusers] = useState ([]);
    useEffect (() => {
    fetch ("https://jsonplaceholder.typicode.com/users")
     • then ((res) => res.json ())
     • then ((data) => setusers (data))
     • catch ((errors) => consol.error(" error fetching
              users :" ; error ));
    } [])
    return (
     <div>
      <h2 User list </h2>
       <ul>
{users.map (user) => {
     <li key = {user.id}> {user.names} </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 to http://jsonplaceholder.typicode.com/users.
- we pass an empty dependancy array ([]) as the Second argument to useEffect. This ensures that the effect runs only once when the component is mounted.
- the fetch() function retrieves the user data, & when the response arives, we update the state using setUsers(data)
- updating state triggers a re-render, & the component displays the list of user names inside the
- without useEffect, if we put fetch() directory in the component body, it would run on every render & cause an infinite loop.

output :

User list

- Leanne Graham
- Ervin Howell
- Clementine Beach.

Q 2) -

* models/Book.js:

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

* controllers/ books Controller. js :

```
import {addBook, getAllBooks} from "../models/Books.js";
export function showbooks (req, res){
    count 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 :

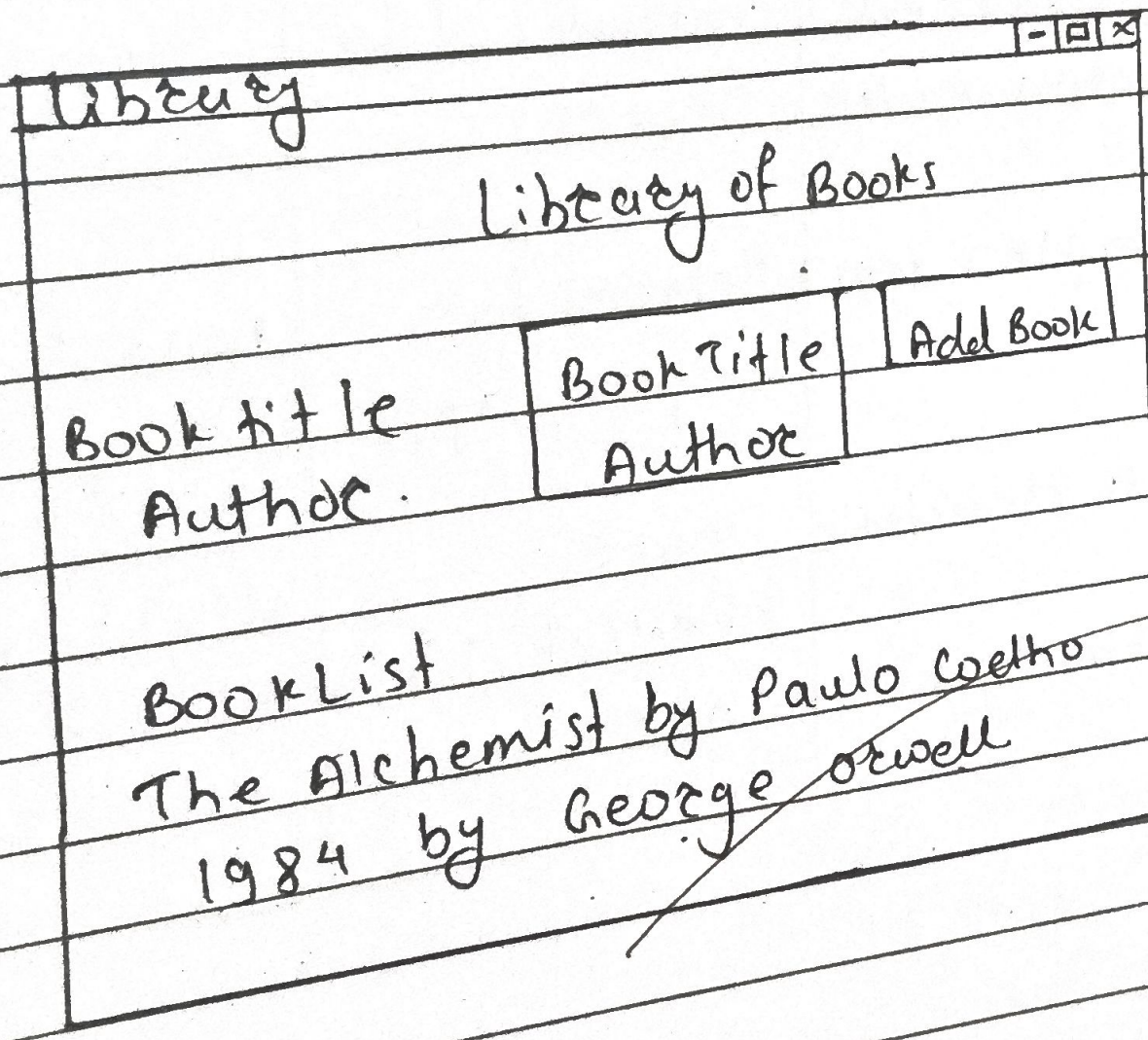
```
<!DOCTYPE html>
<html>
<head>
```

```
<title> library </title>
</head>
<body>
<h1> Library of Books </h1>
<form action = "/books" method = "title" placeho
                    = "Books 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 %>
<% }) %>
</ul>
</body>
</html>
```

Server.js:

```
import express from "express";
import body parser from "body-parser";
import {showBooks, createBooks} from "./controllers/
  bookController.js";
const app = express ();
app.set ("view engine", "ejs");
app.use (bodyparser.urlencoded ({extended: true}));
app.get ("/books", showBooks);
```

```
app.post("/books", createBook);
app.listen(5000, () => {
console.log("server running on http://localhost:5000
    /books");
});
```



Legibility Total

Q3> -

>	Features	mvc	FLUX	Redu
i)	Data flow	Bidirectional	Unidirectional	Unidirec
ii)	State management	Scattered access models	Centralized in stores	Single glo Store.
iii)	Complexity handeling	Simpler apps	Better for complex apps	Best for larg apps.
iv)	Predictability	less Predictable	more Predictable	Highly predicta
v)	Middleware Support	Limited	optional	Built in Suppor

4> -

> actions/cartActions.js:

```
Support 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, quality) => ({
  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>
);
```

Q5) -

```
import React, { createContext, useContext, useState,
    useRef, useImperativeHandle, forwardRef }
    from "react";
const FormContext = createContext();
export const 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>
<Name Input ref = {nameInputRef} />
 <p> Email: {formData.Email} </p>
<button onClick = {() => nameInputRef.
   current.focus Input ()}>
     focus NameInput.
   </button>
  </div>
  );
};

export default function App (){
return (
  <Form Provider>
    <form />
    </ Form Provider>
  );
}
```

* Benifits :
  - Cleaner State Management
  - Better Reusability
  - Direct Component Control
  - Improved performance