REACT 1st program:

AddProperty.js:

```
import React, { useState } from 'react';

import PropertyService from './PropertyService';

import './App.css';

const AddProperty = () => {

 const [property, setProperty] = useState({

 _id: '',

 type: '',

 location: '',

 price: '',

 rooms: '',

 size: ''
 };

 const handleSubmit = async (e) => {

 e.preventDefault();

 setError(null);

 try {

  await PropertyService.addProperty(property);
  setMessage('Property added successfully!');
 } catch (err) {
  setError(err.message);
 }
 };
 return (
<div className="add-property-container">
<h2>Add New Property</h2>
  {error && <p className="error">{error}</p>}
  {message && <p className="success">{message}</p>}
<form onSubmit={handleSubmit}>
  {['type', 'location', 'price', 'rooms', 'size', '_id'].map((field) => (
<div key={field}>
<label>{field}:</label>
<input
    name={field}
    value={property[field]}
    onChange={handleChange}
    required
```

```
   />
</div>
  ))}
<button type="submit">Add Property</button>
</form>
</div>
 );
};
export default AddProperty;
```

PropertyList.js :

```
import React, { useState, useEffect } from 'react';
import PropertyService from './PropertyService';
import { Link } from 'react-router-dom';
import './App.css';
const PropertyList = () => {
 const [properties, setProperties] = useState([]);
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState(null);

 useEffect(() => {

 const fetchProperties = async () => {

  try {

  const data = await PropertyService.getAllProperties();

  setProperties(data);

  setLoading(false);

  } catch (err) {

  setError(err.message);

  setLoading(false);

  }

 };

 fetchProperties();

 }, []);

 if (loading) return <p>Loading...</p>;

 if (error) return <p>Error: {error}</p>;

 return (

<div className="property-list-container">

<h2 className="property-list-header">Properties List</h2>
<ul className="property-list">

  {properties.map((property) => (

<li key={property._id}>

<Link to={`/properties/${property._id}`}>

    {property.location} - {property.type}

</Link>

</li>

  ))}

</ul>

</div>

 );

};

export default PropertyList;
```

PropertyDetail.js :

```
import React, { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import PropertyService from './PropertyService';
import './App.css';

const PropertyDetail = () => {

 const { propertyID } = useParams();

 const [property, setProperty] = useState(null);

 const [loading, setLoading] = useState(true);

 const [error, setError] = useState(null);

 useEffect(() => {

 const fetchProperty = async () => {

  try {

  const data = await PropertyService.getPropertyByID(propertyID);

  setProperty(data[0]); // use first element

  setLoading(false);

  } catch (err) {

  setError(err.message);

  setLoading(false);

  }

 };

 fetchProperty();

 }, [propertyID]);
 if (loading) return <p>Loading...</p>;

 if (error) return <p>Error: {error}</p>;

 return (

<div className="property-detail-container">

<h2>Property Details</h2>

<p>Type: {property.type}</p>

<p>Location: {property.location}</p>

<p>Price: {property.price}</p>

<p>Rooms: {property.rooms}</p>

<p>Size: {property.size}</p>

</div>

 );

};

export default PropertyDetail;
```

PropertyService.js :

```
const API_URL = `http://localhost:3000/properties`;

const PropertyService = {
```

```
getAllProperties: async () => {
const response = await fetch(API_URL);
if (!response.ok) {
 throw new Error('Failed to fetch properties');
}
return response.json();
},
getPropertyByID: async (propertyID) => {
const response = await fetch(`${API_URL}?_id=${propertyID}`);
if (!response.ok) {
 throw new Error('Failed to fetch property details');
}
return response.json(); // returns an array
},
addProperty: async (newProperty) => {
const response = await fetch(API_URL, {
 method: 'POST',
 headers: { 'Content-Type': 'application/json' },
 body: JSON.stringify(newProperty),
 });

 if (!response.ok) {

  throw new Error('Failed to add property');

 }

 return response.json();

 },

};

export default PropertyService;
```

REACT 2nd program:

PatientInformation.js :

```
import React, { useState, useEffect } from 'react';

import { getPatients } from './PatientService';

import './App.css';
```

```
export const PatientInformation = ({ patientID }) => {
 const [patient, setPatient] = useState(null);
 useEffect(() => {
 const fetchPatient = async () => {
  const patients = await getPatients();
  const found = patients.find(p => p.patientID === patientID);
  setPatient(found || null);
 };
 if (patientID) {
  fetchPatient();
 }
 }, [patientID]);
 return (
 <div className="patient-info-container">
  {patient ? (
  <div className="patient-card">
   <h3>Patient Details</h3>
   <p>Patient ID: {patient.patientID}</p>
   <p>Name: {patient.name}</p>
   <p>Age: {patient.age}</p>

   <p>Gender: {patient.gender}</p>

   <p>Condition: {patient.condition}</p>

   <p>Last Visit: {patient.lastVisit}</p>

  </div>

  ) : (

  <p>No patient found for ID: {patientID}</p>

  )}

 </div>

 );

};
```

PatientRegistrationForm.js :

```
import React, { useState } from 'react';

import { addPatient } from './PatientService';

import './App.css';

const PatientRegistrationForm = ({ onRegister }) => {

 const [errors, setErrors] = useState({});
 const [formData, setFormData] = useState({

 name: '',

 age: '',

 gender: '',

 condition: '',

 lastVisit: '',

 });

 const handleChange = (e) => {

 const { name, value } = e.target;

 setFormData({ ...formData, [name]: value });

 };

 const isValidDate = (dateString) => {

 const regex = /^\d{4}-\d{2}-\d{2}$/;

 return regex.test(dateString);

 };

 const validateForm = () => {

 const errs = {};

 if (!formData.name.trim()) errs.name = 'Name is required';

 if (!formData.age) errs.age = 'Age is required';
```

```
 else if (isNaN(formData.age) || formData.age <= 0) errs.age = 'Age must be a
positive number';

 if (!formData.gender) errs.gender = 'Gender is required';

 if (!formData.condition.trim()) errs.condition = 'Condition is required';

 if (!formData.lastVisit.trim()) errs.lastVisit = 'Last Visit is required';

 else if (!isValidDate(formData.lastVisit)) errs.lastVisit = 'Invalid date format
(YYYY-MM-DD)';

 setErrors(errs);

 return Object.keys(errs).length === 0;

 };

 const handleSubmit = async (e) => {

 e.preventDefault();

 if (!validateForm()) return;

 const newPatient = {

  ...formData,

  patientID: `P${Date.now().toString().slice(-4)}`

 };

 await addPatient(newPatient);

 if (onRegister) {

  onRegister(formData); // matches test expectation
 }

 setFormData({ name: '', age: '', gender: '', condition: '', lastVisit: '' });

 setErrors({});

 };

 return (

 <form className="patient-form" onSubmit={handleSubmit}>

  <h3>Register New Patient</h3>

  <input name="name" placeholder="Name" value={formData.name}
onChange={handleChange} />

  {errors.name && <div className="error">{errors.name}</div>}

  <input name="age" placeholder="Age" value={formData.age}
onChange={handleChange} />

  {errors.age && <div className="error">{errors.age}</div>}

  <select name="gender" value={formData.gender} onChange={handleChange}>

  <option value="">Select Gender</option>

  <option>Male</option>

  <option>Female</option>

  <option>Other</option>

  </select>

  {errors.gender && <div className="error">{errors.gender}</div>}
  <input name="condition" placeholder="Condition" value={formData.condition}
onChange={handleChange} />

  {errors.condition && <div className="error">{errors.condition}</div>}

  <input name="lastVisit" placeholder="Last Visit (YYYY-MM-DD)"
value={formData.lastVisit} onChange={handleChange} />

  {errors.lastVisit && <div className="error">{errors.lastVisit}</div>}

  <button type="submit">Register Patient</button>

 </form>

 );

};

export default PatientRegistrationForm;
```

PatientService.js :

```
import environment from "./environments/environment.ts"

const API_URL = environment.apiUrl;

export const getPatients = async () => {

 const response = await fetch(`${API_URL}/patients`);

 if (!response.ok) throw new Error("Failed to fetch patients");

 return await response.json();
};

export const addPatient = async (newPatient) => {
 const response = await fetch(`${API_URL}/patients`, {
  method: 'POST',
  headers: {
   'Content-Type': 'application/json'
  },
  body: JSON.stringify(newPatient)
 });
 if (!response.ok) throw new Error("Failed to add patient");
 return await response.json();
};
```

HTML

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

```
<title>Online Banking: Account Transactions Viewer</title>

<style>

 body {

 background-color: #f0f0f0;

 }

 form {

 display: flex;

 flex-direction: column;

 width: 50%;

 justify-content: center;

 align-items: center;

 border: 1px solid #fff;

 margin: 0 auto;

 padding: 10px;

 }

 div {

 width: 50%;

 display: flex;

 justify-content: center;
```

```
margin: 4rem auto;

}

label {

width: 20%;

font-size: 1.2rem;

}

select {

width: 20%;

}

table {

font-family: arial, sans-serif;

border-collapse: collapse;

width: 100%;

}

td,

th {

border: 1px solid #dddddd;

text-align: left;

padding: 8px;
```

```
 }

 tr.deposit {

 background-color: #d4edda;

 color: #155724;

 }

 tr.withdrawl {

 background-color: #f8d7da;

 color: #721c24;

 }

 a:hover {

 color: orange;

 }

</style>

</head>

<body>

<h2>Online Banking: Account Transactions Viewer</h2>

<div>

 <label for="type">Transaction Type</label>

 <select id="type">
 <option value="">All</option>

 <option value="DEPOSIT">DEPOSIT</option>

 <option value="WITHDRAWL">WITHDRAWL</option>

 </select>

 <button id="search-btn">Search</button>

</div>

<div>

 <table>

 <thead>

  <tr>

  <th>Description</th>

  <th>Amount</th>

  <th>Type</th>

  </tr>

 </thead>

 <tbody id="transactionTableBody"></tbody>

 </table>

</div>

<script type="text/javascript">
```

```
// Do not change these hardcoded transactions

const transactions = [

{

 description: "Transfer to Mr A",

 amount: 1000,

 type: "WITHDRAWL",

},

{

 description: "Salary March 2022",

 amount: 50000,

 type: "DEPOSIT",

},

{

 description: "House Rent",

 amount: 4000,

 type: "WITHDRAWL",

},

{

 description: "Receive from Mr B",
   amount: 2000,

   type: "DEPOSIT",

  },

  ];

  const transactionTableBody =
document.getElementById("transactionTableBody");

  const searchBtn = document.getElementById("search-btn");

  const dropdown = document.getElementById("type");

  // Populate transactions based on selected type

  searchBtn.addEventListener("click", (e) => {

  e.preventDefault();

  const selectedType = dropdown.value;

  populateTransactions(selectedType);

  });

  function populateTransactions(selectedType = "") {

  transactionTableBody.innerHTML = "";

  const filteredTransactions = getTransactions(selectedType);

  filteredTransactions.forEach((transaction) => {

   const row = document.createElement("tr");
  row.className = transaction.type.toLowerCase();

  row.innerHTML = `

  <td>${transaction.description}</td>

  <td>${transaction.amount}</td>

  <td>${transaction.type}</td>

  `;

  transactionTableBody.appendChild(row);

 });

 }

 function getTransactions(selectedType) {

 if (selectedType === "") {

  return transactions;

 }

 return transactions.filter((transaction) => transaction.type === selectedType);

 }

 // Populate all transactions initially

 populateTransactions();

</script>

</body>
```

```
</html>
```

----

react location mapper

```
import React, { useState, useEffect, useCallback } from "react";
import { Map, GoogleApiWrapper } from "google-maps-react";
import LocationMarker from "./LocationMarker";
export const App = ({ google }) => {
  const [properties, setProperties] = useState([]);
  const [searchQuery, setSearchQuery] = useState("");
  const [searchResults, setSearchResults] = useState([]);
  const [mapCenter, setMapCenter] = useState({ lat: 31.5497, lng: 74.3436 });
  const [isMounted, setIsMounted] = useState(true);
  const [map, setMap] = useState(null);
  const [markers, setMarkers] = useState([]);



  const handleMapReady = (mapProps, map) => {
    setMap(map);
    setMapCenter(map.center.toJSON());

      };

  const handleSearch = () => {
    if(!google || google.maps) return;
    const service=new google.maps.places.PlaceService(map);
    service.textSearch({query:searchQuery},(results,status)=>{
      if(status==="OK"){
        setSearchResults(results);
      }
    });
```

```
};

const handleAddLocation = (result) => {
  const location=result.geometry.location;
  const position={
    lat:location.let(),
    lng:location.lng(),
  };

  const marker=new google.maps.Marker({
    position,map,title:result.name,
  });
  setMarkers((prev)=>[...prev,marker]);
  setProperties((prev)=>[...prev,{name:result.name,position}]);
  setSearchResults([]);
  setSearchQuery("");


};

const handleRemoveLocation = useCallback(
 (index)=>{
  const newProperties=[...properties];
  newProperties.splice(index,1);
  setProperties(newProperties);
  removeMarker(index);
},
  [properties]
);

const removeMarker = useCallback(

      (index)=>{
        if(index<0 || index>=markers[index]) return;
      const marker=markers[index];
      markers.setMap(null);
      const newMarkers=[...markers];
      newMarkers.splice(index,1);
      setMarkers(newMarkers);
      },
        [markers]
      );


   const handleMapClick = (mapProps, map, clickEvent) => {
     const geocoder = new google.maps.Geocoder();
     const latLng={
       lat: clickEvent.latLng.lat(),
           lng: clickEvent.latLng.lng()
     }
     geocoder.geocode(
       {
         location: {
           lat: clickEvent.latLng.lat(),
           lng: clickEvent.latLng.lng(),
         },
       },
       (results, status) => {
         if (status === "OK") {
           if (results[0]) {
             const marker=new
google.maps.Marker({position:latLng,map,title:results[0].formatted_address,});
             setMarkers((prev)=>[...prev,marker]);

setProperties((prev)=>[...prev,{name:results[0].formatted_address,position:latLng
},]);

         } else {
           console.log("Geocoder failed due to: " + status);
         }
       }
   });
   };

   useEffect(() => {
     if (properties.length > 0 && isMounted) {
```

```
     if(properties.length>0){
       setMapCenter(properties[properties.length-1].position);
      }

    }
  }, [properties]);

  return (
    <div style={{ display: "flex" }}>

      <div style={{ flex: "1 1 50%", position: "relative", height: "500px" }}>
      <label htmlFor="search">Enter location</label>
        <input type="text" id="search" value={searchQuery}
onChange={(e)=>setSearchQuery(e.target.value)} placeholder="Search
Location"/>
        <button onClick={handleSearch} >Search</button>

        <ul>
          {searchResults.map((result,index)=>{
            <li key={index}>{result.name}<button onClick={
              ()=>{
                handleAddLocation(result)
              }
            }>Add</button></li>
          })}
        </ul>
        <h3>Saved Location</h3>
        <ul>
          {properties.map((prop,index)=>(

            <li key={index}>{prop.name}<button onClick={
              ()=>{
                handleRemoveLocation(index)
              }}>Remove</button></li>
            ))}
        </ul>
      </div>
```

```
        <div>
          <Map
          google={google}
          zoom={5}
          initialCenter={mapCenter}
          onReady={handleMapReady}
          onClick={handleMapClick}
          >
        {properties.map((prop,index)=>(
          <LocationMarker
          key={index}
          position={prop.position}
          map={map}
          marker={markers[index]}
          onRemove={()=>handleRemoveLocation(index)}

          />
        ))}


          </Map>
        </div>
      </div>
    );
  };

  export default GoogleApiWrapper({
    apiKey: "AIzaSyDh0LyUchQyqlcsHgYRO5w7iUV4ttlNdDI",
  })(App);
  LocationMarker.js

  import { useEffect } from 'react';

  const LocationMarker = ({ position, map, marker, onRemove }) => {
    useEffect(() => {
        if(!marker)return;
        const handleClick=()=>{
          if(onRemove) onRemove();
      marker.setMap(null);
     };
     marker.addListener("click",handleClick);




   return () => {
     window.google.maps.event.clearListeners(marker,"click");


   };
  }, [map, position, marker, onRemove]);

  return null;
};

export default LocationMarker;
```

----

mock test react

```
import React, { useEffect, useState } from "react";

import {

  getSalesData,

  calculateTotalSales,

  calculateTotalCashSale,

  calculateTotalCreditSale,

  calculateBuyerWithMostSale,
} from "./Reports";

import "./Dashboard.css";

function Dashboard() {

  const [totalSales, setTotalSales] = useState(0);

  const [totalCashSales, setTotalCashSales] = useState(0);

  const [totalCreditSales, setTotalCreditSales] = useState(0);

  const [mostSalesBuyer, setMostSalesBuyer] = useState({

    buyerName: "",

    saleTotal: 0,

  });

  useEffect(() => {

    async function loadData() {

      try {

        const sales = await getSalesData();

        console.log("Fetched sales data:", sales);

        setTotalSales(calculateTotalSales(sales));

        setTotalCashSales(calculateTotalCashSale(sales));

        setTotalCreditSales(calculateTotalCreditSale(sales));

        setMostSalesBuyer(calculateBuyerWithMostSale(sales));
      } catch (error) {

        console.error("Error loading sales data:", error);

      }

    }

    loadData();

  }, []);

  return (
<div className="dashboard">
<div className="card">
<h2>Total Sales</h2>
<p>{totalSales}</p>
</div>
<div className="card">
<h2>Total Cash Sales</h2>
<p>{totalCashSales}</p>
</div>
<div className="card">
<h2>Total Credit Sales</h2>
<p>{totalCreditSales}</p>
</div>
<div className="card">
<h2>Buyer with Most Sales</h2>
<p>{mostSalesBuyer.buyerName}</p>
<p>{mostSalesBuyer.saleTotal}</p>
</div>
</div>

  );

}
```

```
export default Dashboard;

import axios from "axios";

export const getSalesData = async () => {

  let { data } = await axios.get(`/sales.json`);

  return data;

};

export const calculateTotalSales = (sales) => {

  if (!Array.isArray(sales)) return 0;

  return sales.reduce((sum, sale) => sum + sale.saleTotal, 0);

};

export const calculateTotalCashSale = (sales) => {

  if (!Array.isArray(sales)) return 0;

  return sales

    .filter((sale) => sale.creditCard === false)

    .reduce((sum, sale) => sum + sale.saleTotal, 0);

};

export const calculateTotalCreditSale = (sales) => {

  if (!Array.isArray(sales)) return 0;

  return sales
```

```
    .filter((sale) => sale.creditCard === true)
    .reduce((sum, sale) => sum + sale.saleTotal, 0);
};
export const calculateBuyerWithMostSale = (sales) => {
  if (!Array.isArray(sales)) return { buyerName: "", saleTotal: 0 };
  const buyerTotals = {};
  sales.forEach((sale) => {
    if (!buyerTotals[sale.buyerName]) {
      buyerTotals[sale.buyerName] = 0;
    }
    buyerTotals[sale.buyerName] += sale.saleTotal;
  });
  let maxBuyer = "";
  let maxTotal = 0;
  for (const [buyerName, total] of Object.entries(buyerTotals)) {
    if (total > maxTotal) {
      maxBuyer = buyerName;
      maxTotal = total;
    }
  }

  return { buyerName: maxBuyer, saleTotal: maxTotal };

};
```