Assignment - II

Name	:	Rajkumar. R
Roll No	:	727822TUCS147
Class	:	II - CSE - B
Course Code	:	22SB303
Subject	:	Front End Development Using React.

1). you are tasked with crafting a dynamic web appliation tailored for an Edutech platform. focusing on seamless user profile management. The appliations are feautures include utilizing redux for adapt state management. orchestrating HTTP requests to failitate data retrieval and updates and employing react router to optimize navigation your responsibilities extend to implementing robust form validation technique and establish a comprehensive error-handling system, ensuring a user-centric, reliable and error - free experience within the appliation. Discuss your approach to address the following aspects:

1). State management with redux:

Redux plays a crucial role is managing the state of appliation. especially is scenarious where state multiple components. For user profiles, the redux store is slice dedicated to user information. The structure would include

Actions: Actions are payloads of information that send data from the application to the redux store actions related to user profiles might include Fetch_user, update_user, Delete_user etc

Reducers: Reducers specify how the applications state changes in response to actions.

Store: The Redux store would be configured with middleware for async actions and combined reducers to manage various slices of state.

Example redux action for user profiles.

```
const fetchuser = ( userId ) => async (dispatch)
=> { dispatch ( { type , 'Fetch_user_Request'});
try {
 const response = await axios .get ('/api/
                                   users/${ userId});
dispatch ({ type: 'Fetch_user_success', payload
                                response.data });
}
catch (error) {
dispatch ({ type: Fetch_user_success', payload
                        error.message });
}
}
```

2). HTTP requests and data management

For making HTTP requests, Axios is a popular choice. In the redux action creators, asynchronous operations would be handled.

For user profiles

Fetching Data

The Fetch_user action would trigger an axios request upon success. fetch_user_success action would be dispatched with retrieved data.

Error Handling

Axios interceptors are used for global error handling. Fetch_user_failure action would be dispatched capturing the error information.

```
axios. interceptors. response. use (
( response ) => response,
(error ) => {
store. dispatch ( { type: 'global. error',
payload : error . message });
return Promise . reject ( error );
}
}
```

3). React Router and Navigation

React Router is crucial for managing navigation in single - page application.

Linking views

Links would be created using link components ensuring that navigation is handled without full - page reloads for a seamless experience

Dynamic routes:-

If the user profile has a unique URL, dynamic routes can be set up to handle different profiles defining individual routes for each

```
<Router>
<Switch>
<Route path="/profile/:userId"
    component={userprofile}/>
<Route path="/dashboard"
    component={Dashboard}/>
<Redirect from="/" to="/dashboard"/>
</Switch>
</Routers>
```

4). Form validation in React

It is vital for smooth user experience
A custom dynamic input Component would be created to handle different types of inputs.

Valildation Rules :
Define validation rules for each form field specifying whether a field is required . type of data it should contain.
Dynamic Rendering
The form component would dynamically render input fields based on the configuration . making it easier to manage

```
// Dynamic input component with validation
const DynamicInput = ({ type . value .
onchange . validation }) => {
return < input type = { type }
              value = { value }
              onchange = { onchange } / > ;
};
```

5). Error Handling and Debugging
For Debugging react applications
React DevTools:
These browser extensions allow

inspecting react component hierarchies.

Browser Development Tools:

Standard browser tools for debugging profiling and monitoring network requests.

Error Boundaries

Implementing error boundaries to gracefully handle unexpected errors and provide a user-friendly error message without crashing the application.

```
// Error boundary example
class ErrorBoundary extends React.
                         Component {
  constructor ( props) {
      super ( props);
      This . state = { has error : false };
}

component Did Catch (error , error Info) {
```

```
this . setState ({ hasError : false });
logErrorToService (error, errorInfo);

}
render () {
return this. state . hasError ?
< ErrorFallback /> ;
this . props. children ;
}
}
```