Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

All questions carry equal weight. Total points: 100. The test is open book, open notes and open Internet.

(1) Consider the following html code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My first HTML5 page</title>
    <meta name="description" content="My HTML5 page">
    <link rel="stylesheet" href="css/styles.css">
   </head>

  <body>
    <div id="root">
    </div>
    <script src="js/scripts.js"></script>
  </body>
</html>
```

Note that there is a div with an id of “root”. Write/type html code below to insert a form to allow user login. The form should have two inputs one of type text and the other of type password with corresponding label elements. Each input element should have a change event handler passed to it. The form should have a submit button and should handle the submit event. Just pass the handlers by name. You will write the handler definitions in JavaScript in Question #2. For each html element either pass a class or id attribute and you will use that in Question #3 to write the CSS.

```
<div id="root">
     <form>
      <label>Username:</label>
      <input type = "text" id="username" name="username" onchange="validateUsername()">
      <label>Password:</label>
      <input type = "password" id="password" name="password" onchange="validatePassword()">
      <button onclick="Submit()">Submit</button>
     </form>
</div>
```

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(2) Write JavaScript functions for the change handlers for the input elements and the submit handler in Question #1. Assume JavaScript functions validateUsername and validatePassword are available and they return either true or an error message which is a string. The change listeners should call the appropriate validator function and output any error message to the console. The submit handler should prevent the default behavior of the browser and output the username and password to the console. (Don't do these things in a real world application!)

```
let user = document.querySelector('username');
let pass = document.querySelector('password');

function validateUsername(e){
    if (e != "")
        return true;
    else{
        var error = new Error('Invalid Username');
        console.log(error);
    }
}
function validatePassword(e){
    if (e != "")
        return true;
    else{
        var error = new Error('Invalid Password');
        console.log(error);
    }
}
function Submit(){
    console.log(user);
    console.log(pass);
}
```

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(3) Write CSS styles for the form, label, input and button elements. The form should have a two column grid display.The labels should be placed right justified in the left column and the inputs should be left justified on the right column. Place the button center justified on the right column. Style the elements either using a class name or id. Do not style them using html element names. Do not use styled-components.

```
.form {
    display: grid;
    grid-template-columns: auto auto;
    background-color: gray;
}

.label {
    display: grid;
    color: yellow;
    justify-items: right;
}

.input {
    display: grid;
    max-width: 200px;
    justify-items: left;
}

.button {
    display: grid;
    grid-column-start: 2;
    justify-self: center;
    max-width: 80px;
}
```

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(4) Rewrite your CSS styles in Question #3 using styled-components. Specifically, define components UNLabel, PWLabel, UserName, Password, LoginForm and SubmitBtn and add styles to them based on the styles from Question #3.

```
const LoginForm = styled.form`
    display: grid;
    grid-template-columns: auto auto;
    background-color: gray;
`;

const UNLabel = styled.label`
    display: grid;
    color: yellow;
    justify-items: right;
`;

const PWLabel = styled.label`
    display: grid;
    color: yellow;
    justify-items: right;
`;

const Username = styled.input`
    display: grid;
    max-width: 200px;
    justify-items: left;
`;

const Password = styled.input`
    display: grid;
    max-width: 200px;
    justify-items: left;
`;

const SubmitBtn = styled.button`
    display: grid;
    grid-column-start: 2;
    justify-self: center;
    max-width: 80px;
`;
```

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(5) Rewrite the html form in Question #1 using a React function component called Login. Your Login component should accept a props parameter. It should use the useState function to create two state variables username and password respectively and the corresponding setters, setUserName and setPassword. The Login component should define two functions handleUserNameChange and handlePasswordChange both of which accept an event as a parameter. Inside these functions, you should call the appropriate setters. The Login component should also define a handleSubmitClick that should behave similarly to what you wrote in Question #2. The Login component should return a form created using the styled-components components you created in Question #4. Use the handlers you created above as needed in your form.

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

function Login(props){
    const [username, setUsername] = useState("");
    const [password, setPassword] = useState("");

    function handleSubmit(){
        console.log("Username: " + username + "\nPassword: " + password);
    }

    function handleUserNameChange(e){
        e => setUsername(e.target.value)
    }

    function handlePasswordChange(e){
        e => setPassword(e.target.value)
    }

    return(
        <Fragment>
            <LoginForm>
            <Username
                type="text"
                placeholder="Username"
                value={username}
                onChange={e => {handleUserNameChange(e)}}
            />
            <Password
                type="password"
                placeholder="Password"
                value={password}
                onChange={e => {handlePasswordChange(e)}}
            />
        </LoginForm>
        <SubmitBtn onClick={e => {handleSubmit()}} type="submit">Submit</SubmitBtn>
        </Fragment>
    )
}
```

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(6) Create a Higher Order Component withErrorInput that uses a component called AddErrorToInput. In AddErrorToInput, first render props.children and then if props.error is true then render props.errorMsg in a paragraph below where you rendered props.children.The function withErrorInput should take a Component as input and return an AddErrorToInput and passes a function that renders Component as the child.

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(7) Let reducer be a function that takes two parameters called state and action and returns an object that is an updated version of state based on action. Let PRODUCTS be an array of objects. Use the useReducer hook and the context API to demonstrate how to pass data between components at different levels of hierarchy in a React-based application. (Reverse data flow is in Question #8).

```
function reducer(state, action) {
        switch(action.type){
            case "INCREMENT_QUANTITY":
                const {quantity: qty} = state[action.index];
                state[action.index] = {...state[action.index], quantity: qty+1}
                break;
            case "DECREMENT_QUANTITY":
                const {quantity: qty} = state[action.index];
                state[action.index] = {...state[action.index], quantity: qty-1}
                break;
        }
        return state;
    }

    const ProductsContext = React.createContext(null);
    const DispatchContext = React.createContext(null);

    function changeQuantity(index, quantity){
        const [state,dispatch] = useReducer(reducer, initialState);
        return{
            <>
            Count: {state.count}
            <button onClick={() => dispatch({type: 'DECREMENT_QUANTITY'})}>-</button>
            <button onClick={() => dispatch({type: 'INCREMENT_QUANTITY'})}>+</button>
            </>
        }
    }
```

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(8) Demonstrate reverse data flow for the same application as Question #7 using the context API and the useReducer hook.

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(9) In a react-boilerplate based application, you are supposed to implement the login functionality. Write constants.js, actions.js, reducer.js and selectors.js suitable for modifying the Login component from Question #6.

Instructor: Beddhu Murali CSC 440 Client-side Web Programming Name: William Burkes

Fall 2020 Final

(10)Write the complete index.js file re-implementing the Login component from Question #6 suitable for a react-boilerplate application. You should (1) import suitable actions and selectors, (3) define mapStateToProps, mapDispatchToProps, (3) use compose and connect and React.memo as appropriate, (4) define additional handlers in the Login component, (5) and return a style-components based form. Pass appropriate values and functions to the styled-components.