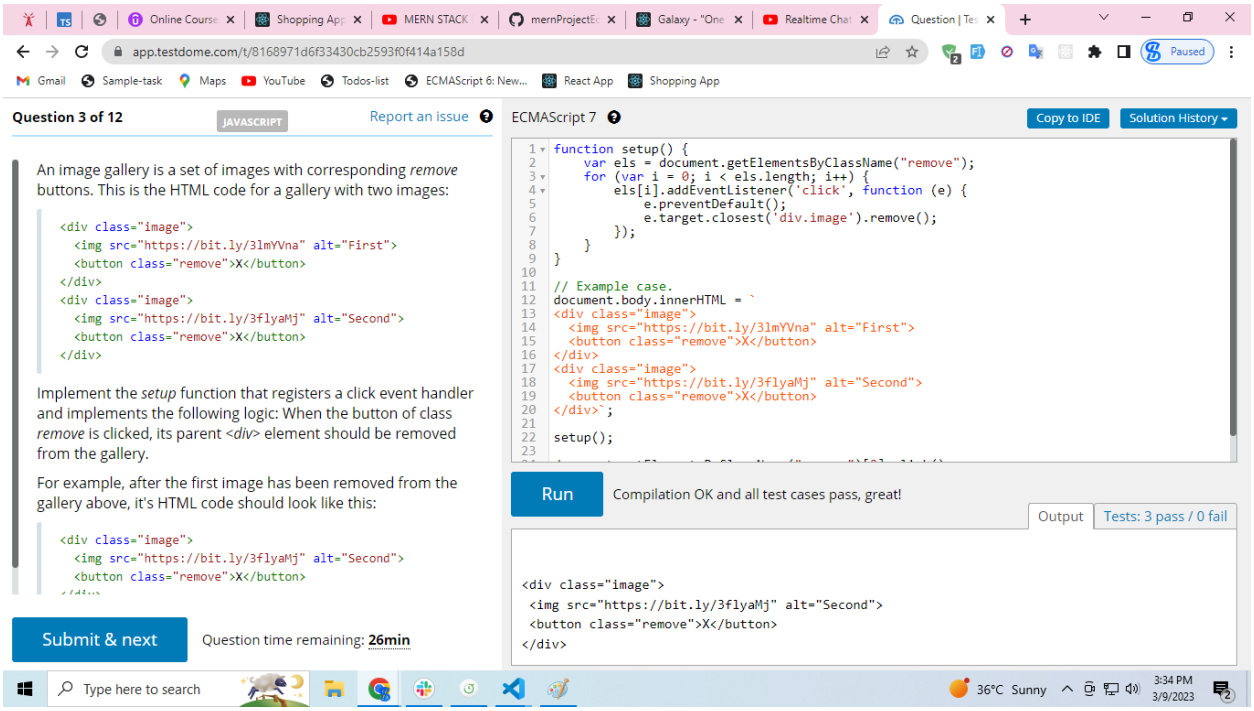

Question 3 of 12
JAVASCRIPT
Report an issue
An image gallery is a set of images with corresponding remove buttons. This is the HTML code for a gallery with two images:

<img src="https://bit.ly/3lmYVna" alt="First">
<button class="remove">X</button>

<img src="https://bit.ly/3flyaMj" alt="Second">
<button class="remove">X</button>

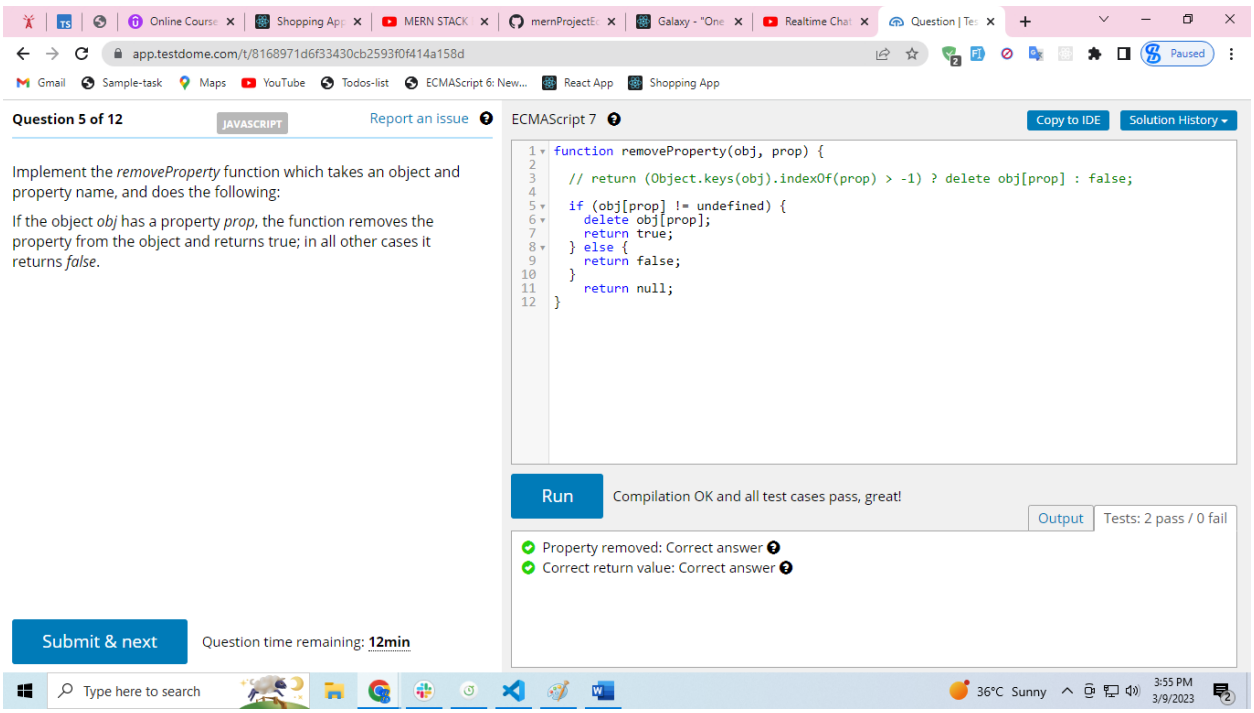
Implement the setup function that registers a click event handler and implements the following logic: When the button of class remove is clicked, its parent element should be removed from the gallery.
For example, after the first image has been removed from the gallery above, it's HTML code should look like this:

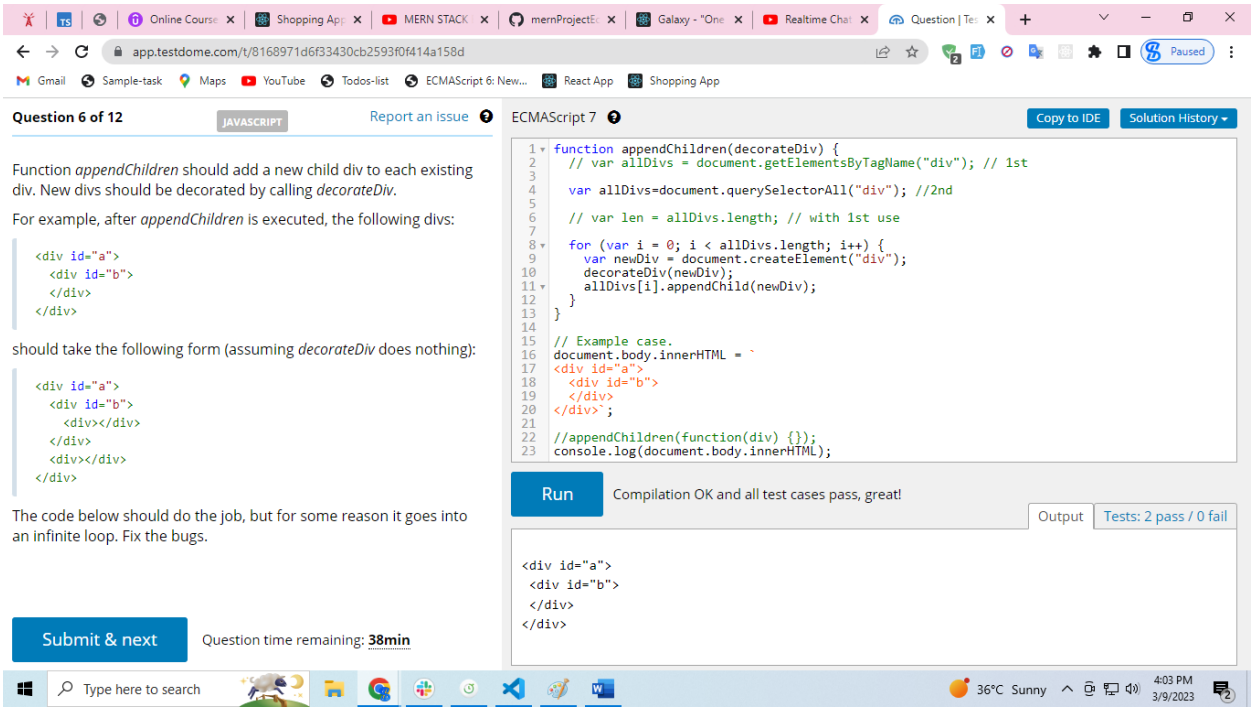
<img src="https://bit.ly/3flyaMj" alt="Second">
<button class="remove">X</button>
Submit & next
Question time remaining: 26min
ECMAScript 7
Copy to IDE
Solution History
function setup() {
var els = document.getElementsByClassName("remove");
for (var i = 0; i < els.length; i++) {
els[i].addEventListener('click', function (e) {
e.preventDefault();
e.target.closest('div.image').remove();
});
}
}
// Example case.
document.body.innerHTML = `

<img src="https://bit.ly/3lmYVna" alt="First">
<button class="remove">X</button>

<img src="https://bit.ly/3flyaMj" alt="Second">
<button class="remove">X</button>
`;
setup();
Run
Compilation OK and all test cases pass, great!
Output
Tests: 3 pass / 0 fail

<img src="https://bit.ly/3flyaMj" alt="Second">
<button class="remove">X</button>


Question 5 of 12
JAVASCRIPT
Report an issue
Implement the removeProperty function which takes an object and property name, and does the following:
If the object obj has a property prop, the function removes the property from the object and returns true; in all other cases it returns false.
Submit & next
Question time remaining: 12min
ECMAScript 7
Copy to IDE
Solution History
function removeProperty(obj, prop) {
// return (Object.keys(obj).indexOf(prop) > -1) ? delete obj[prop] : false;
if (obj[prop] != undefined) {
delete obj[prop];
return true;
} else {
return false;
}
return null;
}
Run
Compilation OK and all test cases pass, great!
Output
Tests: 2 pass / 0 fail
Property removed: Correct answer
Correct return value: Correct answer



Question 6 of 12 JAVASCRIPT Report an issue

Function *appendChildren* should add a new child div to each existing div. New divs should be decorated by calling *decorateDiv*.

For example, after *appendChildren* is executed, the following divs:

```
<div id="a">
  <div id="b">
  </div>
</div>
```

should take the following form (assuming *decorateDiv* does nothing):

```
<div id="a">
  <div id="b">
    <div></div>
  </div>
  <div></div>
</div>
```

The code below should do the job, but for some reason it goes into an infinite loop. Fix the bugs.

Submit & next Question time remaining: **38min**

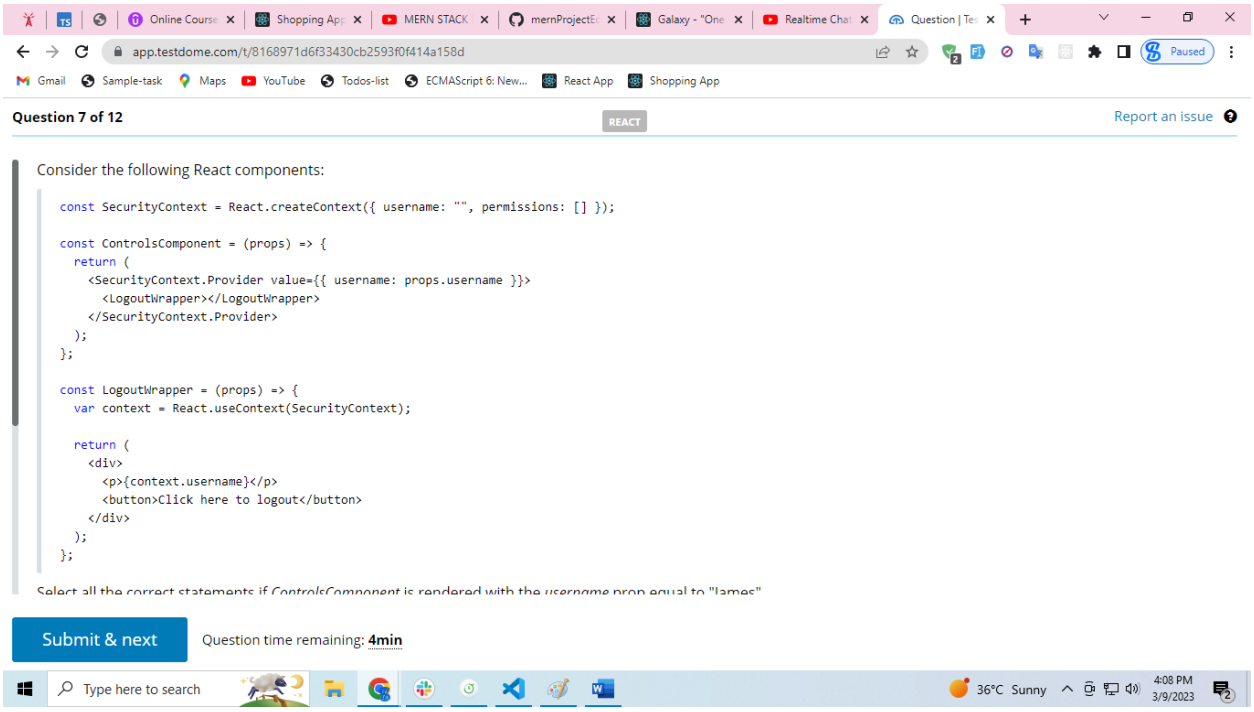
ECMAScript 7 Copy to IDE Solution History

```
function appendChildren(decorateDiv) {
  // var allDivs = document.getElementsByTagName("div"); // 1st

  var allDivs=document.querySelectorAll("div"); //2nd

  // var len = allDivs.length; // with 1st use

  for (var i = 0; i < allDivs.length; i++) {
    var newDiv = document.createElement("div");
    decorateDiv(newDiv);
    allDivs[i].appendChild(newDiv);
  }
}

// Example case.
document.body.innerHTML = `
<div id="a">
  <div id="b">
  </div>
</div>`;

//appendChildren(function(div) {});
console.log(document.body.innerHTML);
```

Run Compilation OK and all test cases pass, great!

Output Tests: 2 pass / 0 fail

```
<div id="a">
 <div id="b">
 </div>
</div>
```



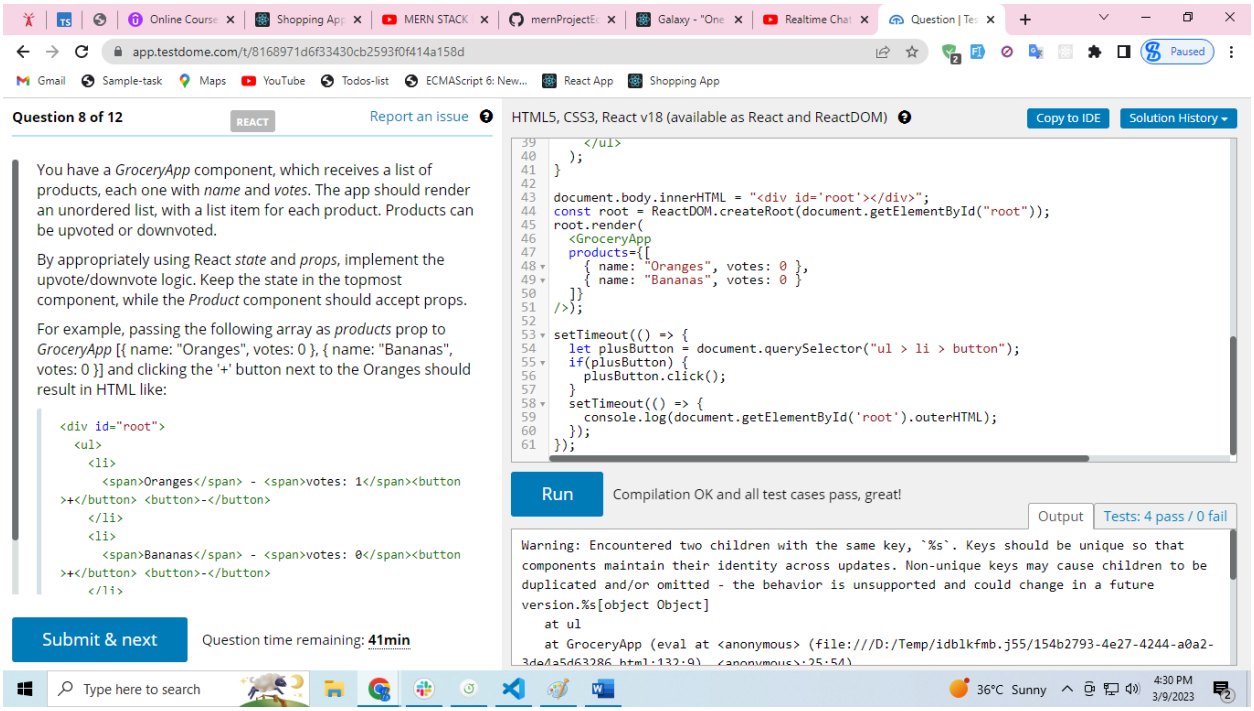
Question 7 of 12 REACT Report an issue

Consider the following React components:

```
const SecurityContext = React.createContext({ username: "", permissions: [] });

const ControlsComponent = (props) => {
  return (
    <SecurityContext.Provider value={{ username: props.username }}>
      <LogoutWrapper></LogoutWrapper>
    </SecurityContext.Provider>
  );
};

const LogoutWrapper = (props) => {
  var context = React.useContext(SecurityContext);

  return (
    <div>
      <p>{context.username}</p>
      <button>Click here to logout</button>
    </div>
  );
};
```

Select all the correct statements if *ControlsComponent* is rendered with the *username* prop equal to "James"

Submit & next Question time remaining: **4min**

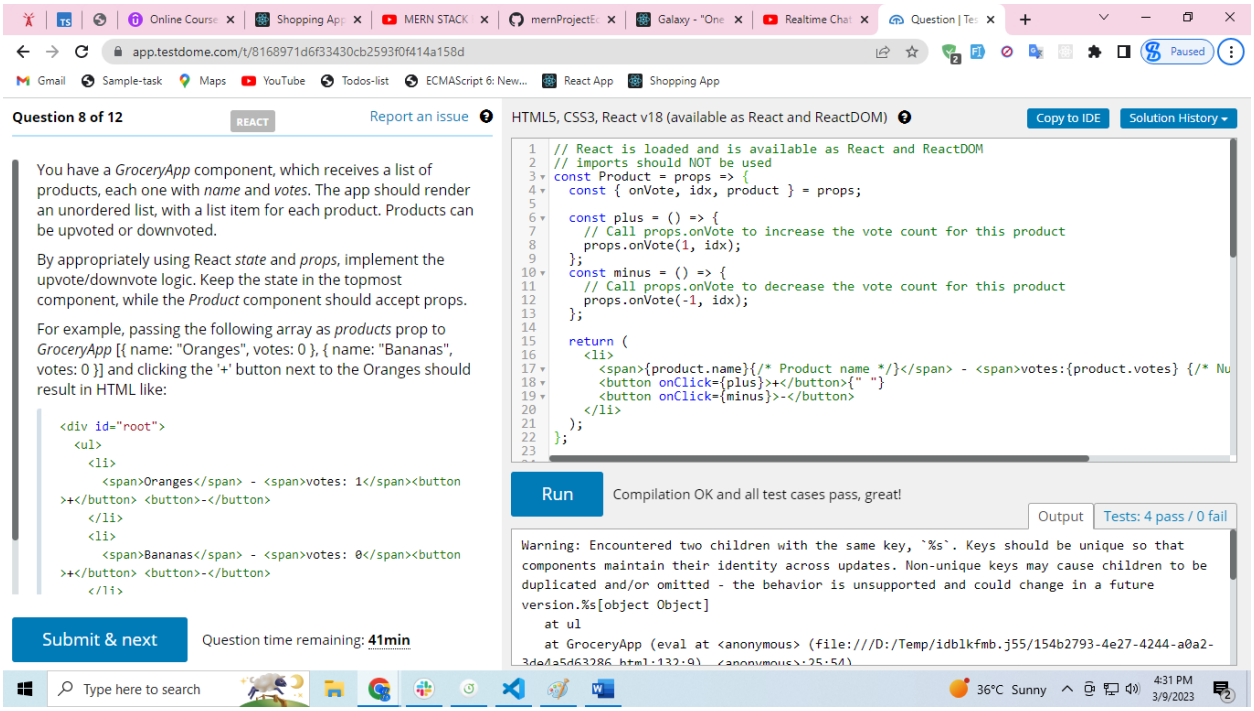

app.testdome.com/t/8168971d6f33430cb2593f0f414a158d
Question 8 of 12
REACT
Report an issue
You have a GroceryApp component, which receives a list of products, each one with name and votes. The app should render an unordered list, with a list item for each product. Products can be upvoted or downvoted.
By appropriately using React state and props, implement the upvote/downvote logic. Keep the state in the topmost component, while the Product component should accept props.
For example, passing the following array as products prop to GroceryApp [{ name: "Oranges", votes: 0 }, { name: "Bananas", votes: 0 }] and clicking the '+' button next to the Oranges should result in HTML like:

Oranges - votes: 1<button
>+</button> <button>-</button>

Bananas - votes: 0<button
>+</button> <button>-</button>

Submit & next
Question time remaining: 41min
HTML5, CSS3, React v18 (available as React and ReactDOM)
Copy to IDE
Solution History

);
}
document.body.innerHTML = " ";
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
<GroceryApp
products={[
{ name: "Oranges", votes: 0 },
{ name: "Bananas", votes: 0 }
]}
/>);
setTimeout(() => {
let plusButton = document.querySelector("ul > li > button");
if(plusButton) {
plusButton.click();
}
setTimeout(() => {
console.log(document.getElementById('root').outerHTML);
});
});
Run
Compilation OK and all test cases pass, great!
Output
Tests: 4 pass / 0 fail
Warning: Encountered two children with the same key, `%s`. Keys should be unique so that components maintain their identity across updates. Non-unique keys may cause children to be duplicated and/or omitted - the behavior is unsupported and could change in a future version.%s[object Object]
at ul
at GroceryApp (eval at <anonymous> (file:///D:/Temp/idblkfmb.j55/154b2793-4e27-4244-a0a2-
Type here to search
36°C Sunny
4:30 PM
3/9/2023


app.testdome.com/t/8168971d6f33430cb2593f0f414a158d
Question 8 of 12
REACT
Report an issue
You have a GroceryApp component, which receives a list of products, each one with name and votes. The app should render an unordered list, with a list item for each product. Products can be upvoted or downvoted.
By appropriately using React state and props, implement the upvote/downvote logic. Keep the state in the topmost component, while the Product component should accept props.
For example, passing the following array as products prop to GroceryApp [{ name: "Oranges", votes: 0 }, { name: "Bananas", votes: 0 }] and clicking the '+' button next to the Oranges should result in HTML like:

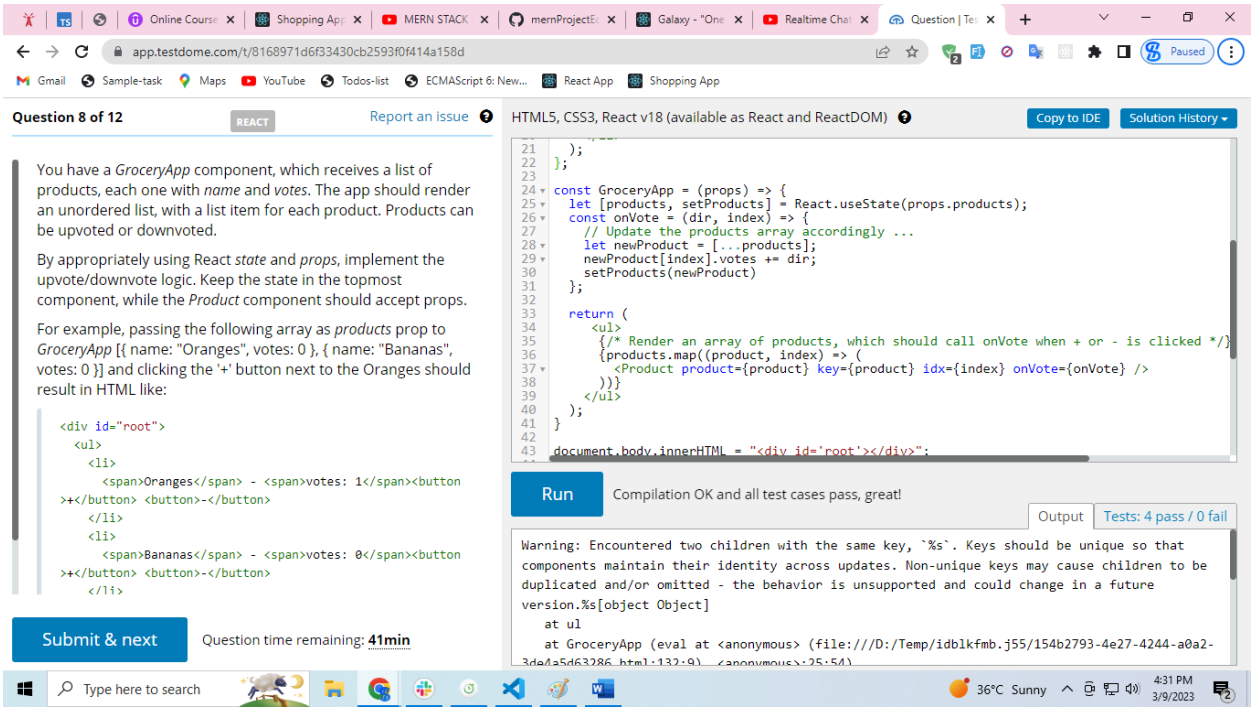
Oranges - votes: 1<button
>+</button> <button>-</button>

Bananas - votes: 0<button
>+</button> <button>-</button>

Submit & next
Question time remaining: 41min
HTML5, CSS3, React v18 (available as React and ReactDOM)
Copy to IDE
Solution History
// React is loaded and is available as React and ReactDOM
// imports should NOT be used
const Product = props => {
const { onVote, idx, product } = props;
const plus = () => {
// Call props.onVote to increase the vote count for this product
props.onVote(1, idx);
};
const minus = () => {
// Call props.onVote to decrease the vote count for this product
props.onVote(-1, idx);
};
return (

{product.name}{/* Product name */} - votes:{product.votes} {/* Nu
<button onClick={plus}>+</button>{" "}
<button onClick={minus}>-</button>

);
};
Run
Compilation OK and all test cases pass, great!
Output
Tests: 4 pass / 0 fail
Warning: Encountered two children with the same key, `%s`. Keys should be unique so that components maintain their identity across updates. Non-unique keys may cause children to be duplicated and/or omitted - the behavior is unsupported and could change in a future version.%s[object Object]
at ul
at GroceryApp (eval at <anonymous> (file:///D:/Temp/idblkfmb.j55/154b2793-4e27-4244-a0a2-
Type here to search
36°C Sunny
4:31 PM
3/9/2023



Question 8 of 12 REACT Report an issue

You have a *GroceryApp* component, which receives a list of products, each one with *name* and *votes*. The app should render an unordered list, with a list item for each product. Products can be upvoted or downvoted.

By appropriately using React *state* and *props*, implement the upvote/downvote logic. Keep the state in the topmost component, while the *Product* component should accept props.

For example, passing the following array as *products* prop to *GroceryApp* [{ name: "Oranges", votes: 0 }, { name: "Bananas", votes: 0 }] and clicking the '+' button next to the Oranges should result in HTML like:

```
<div id="root">
  <ul>
    <li>
      <span>Oranges</span> - <span>votes: 1</span><button
>+</button> <button>-</button>
    </li>
    <li>
      <span>Bananas</span> - <span>votes: 0</span><button
>+</button> <button>-</button>
    </li>
```

Submit & next Question time remaining: **41min**

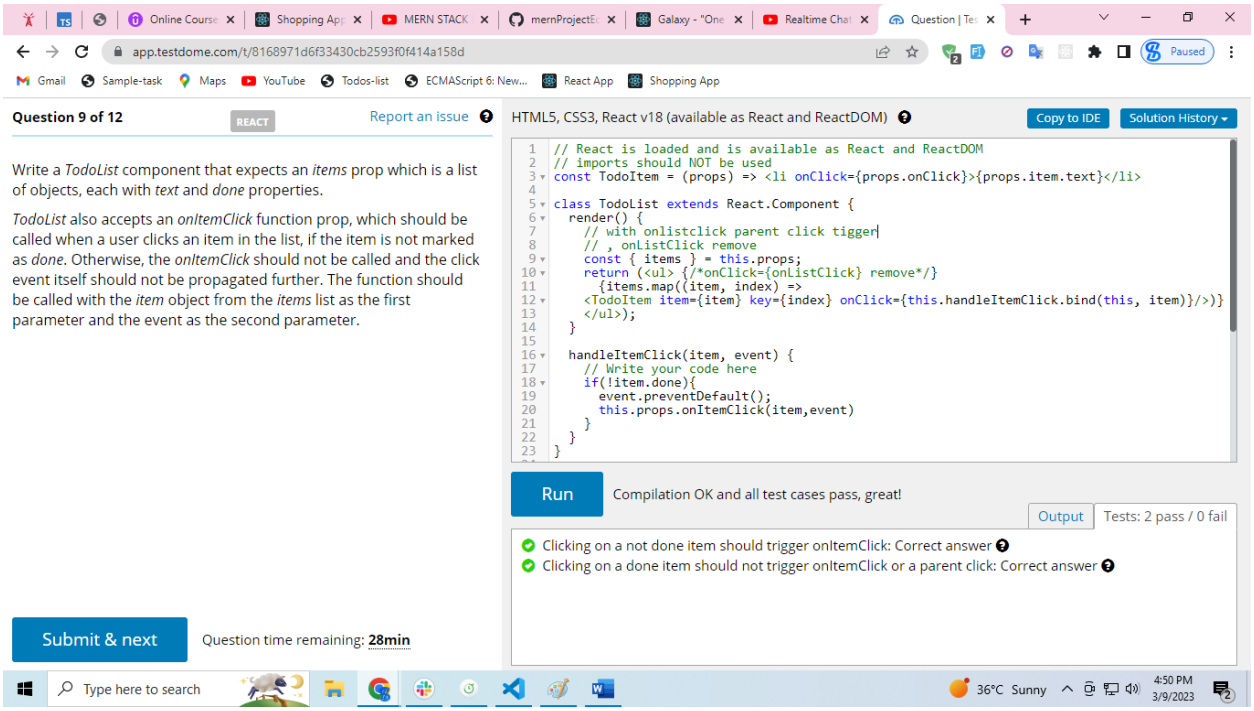
HTML5, CSS3, React v18 (available as React and ReactDOM) Copy to IDE Solution History

```
  );
};

const GroceryApp = (props) => {
  let [products, setProducts] = React.useState(props.products);
  const onVote = (dir, index) => {
    // Update the products array accordingly ...
    let newProduct = [...products];
    newProduct[index].votes += dir;
    setProducts(newProduct)
  };

  return (
    <ul>
      {/* Render an array of products, which should call onVote when + or - is clicked */}
      {products.map((product, index) => (
        <Product product={product} key={product} idx={index} onVote={onVote} />
      ))}
    </ul>
  );
}

document.body.innerHTML = "<div id='root'></div>";
```

Run Compilation OK and all test cases pass, great!

Output Tests: 4 pass / 0 fail

```
Warning: Encountered two children with the same key, `%s`. Keys should be unique so that
components maintain their identity across updates. Non-unique keys may cause children to be
duplicated and/or omitted - the behavior is unsupported and could change in a future
version.%s[object Object]
    at ul
    at GroceryApp (eval at <anonymous> (file:///D:/Temp/idblkfmb.j55/154b2793-4e27-4244-a0a2-
```



Question 9 of 12 REACT Report an issue

Write a *TodoList* component that expects an *items* prop which is a list of objects, each with *text* and *done* properties.

TodoList also accepts an *onItemClick* function prop, which should be called when a user clicks an item in the list, if the item is not marked as *done*. Otherwise, the *onItemClick* should not be called and the click event itself should not be propagated further. The function should be called with the *item* object from the *items* list as the first parameter and the event as the second parameter.

Submit & next Question time remaining: **28min**

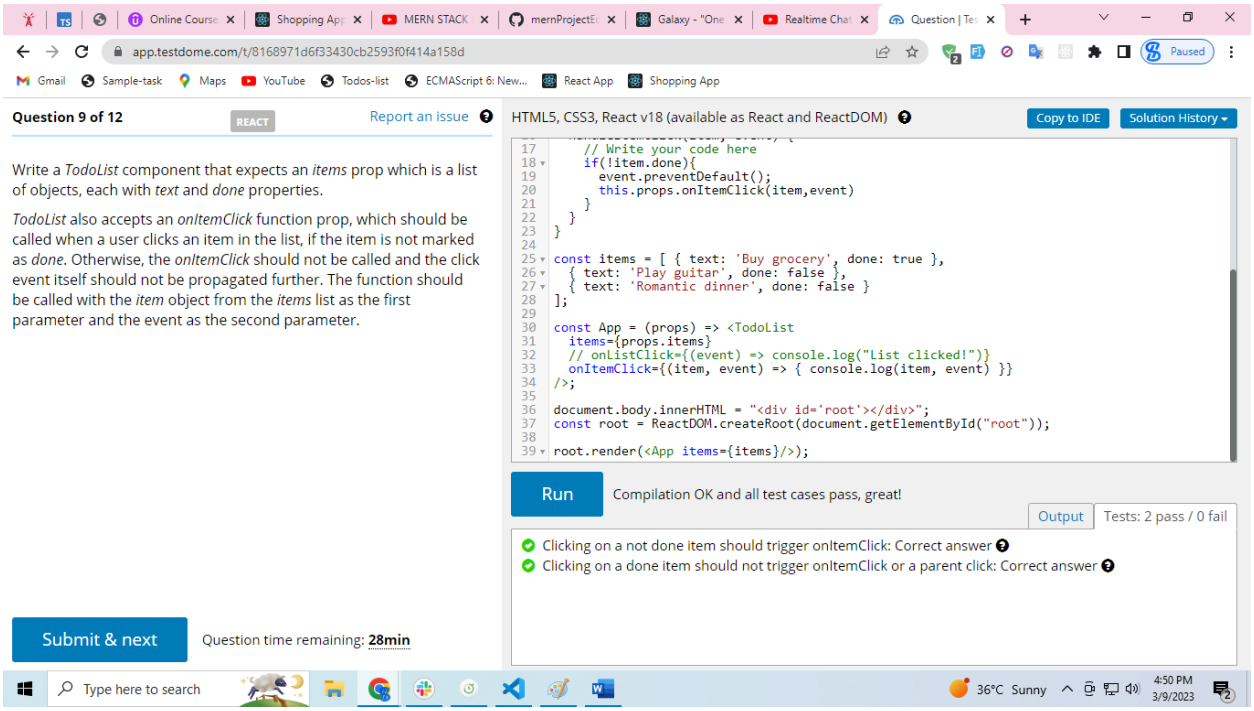
HTML5, CSS3, React v18 (available as React and ReactDOM) Copy to IDE Solution History

```
// React is loaded and is available as React and ReactDOM
// imports should NOT be used
const TodoItem = (props) => <li onClick={props.onClick}>{props.item.text}</li>

class TodoList extends React.Component {
  render() {
    // with onlistclick parent click tigger
    // , onListClick remove
    const { items } = this.props;
    return (<ul> {/*onClick={onListClick} remove*/}
      {items.map((item, index) =>
    <TodoItem item={item} key={index} onClick={this.handleItemClick.bind(this, item)}/>)}
    </ul>);
  }

  handleItemClick(item, event) {
    // Write your code here
    if(!item.done){
      event.preventDefault();
      this.props.onItemClick(item,event)
    }
  }
}
```

Run Compilation OK and all test cases pass, great!

Output Tests: 2 pass / 0 fail

- Clicking on a not done item should trigger onItemClick: Correct answer
- Clicking on a done item should not trigger onItemClick or a parent click: Correct answer



Question 9 of 12 REACT Report an issue

Write a *TodoList* component that expects an *items* prop which is a list of objects, each with *text* and *done* properties.

TodoList also accepts an *onItemClick* function prop, which should be called when a user clicks an item in the list, if the item is not marked as *done*. Otherwise, the *onItemClick* should not be called and the click event itself should not be propagated further. The function should be called with the *item* object from the *items* list as the first parameter and the event as the second parameter.

HTML5, CSS3, React v18 (available as React and ReactDOM)

Copy to IDE | Solution History

```
    // Write your code here
    if(!item.done){
      event.preventDefault();
      this.props.onItemClick(item,event)
    }
  }
}

const items = [ { text: 'Buy grocery', done: true },
  { text: 'Play guitar', done: false },
  { text: 'Romantic dinner', done: false }
];

const App = (props) => <TodoList
  items={props.items}
  // onListClick={(event) => console.log("List clicked!")}
  onItemClick={(item, event) => { console.log(item, event) }}
/>;

document.body.innerHTML = "<div id='root'></div>";
const root = ReactDOM.createRoot(document.getElementById("root"));

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

Run — Compilation OK and all test cases pass, great!

Output | Tests: 2 pass / 0 fail

- Clicking on a not done item should trigger onItemClick: Correct answer
- Clicking on a done item should not trigger onItemClick or a parent click: Correct answer

Submit & next — Question time remaining: **28min**

Question 10 of 12 REACT Report an issue

This application should allow the user to update their username by inputting a custom value and clicking the button.

The *Username* component is finished and should not be changed, but the *App* component is missing parts. Finish the *App* component so that the *Username* component displays the inputted text when the button is clicked.

The *App* component should use the *React.useRef* Hook to pass the input to the *Username* component for the *input* element and for the *Username* component.

For example, if the user inputs a new username of "John Doe" and clicks the button, *App*'s div element should look like this:

```
<div><button>Change Username</button><input type="text"><h1>J
ohn Doe</h1></div>
```

There might be multiple App elements on the page.

HTML5, CSS3, React v18 (available as React and ReactDOM)

Copy to IDE | Solution History

```
// React is loaded and is available as React and ReactDOM
// imports should NOT be used
class Username extends React.Component {
  state = { value: "" };

  changeValue(value) {
    this.setState({ value });
  }

  render() {
    const { value } = this.state;
    return <h1>{value}</h1>;
  }
}

function App() {

  const inputRef = React.useRef();
  const userNameRef = React.useRef();

  function clickHandler() {
    userNameRef.current.changeValue(inputRef.current.value);
  }

  return (
    <div>
      <button onClick={clickHandler}>Change Username</button>
      <input type="text" ref={inputRef} />
      <Username ref={userNameRef} />
    </div>
  );
}
```

Image review — Ctrl+Win+R to Reject — 1/1 — Accept 1 s — Reject

Run — Compilation OK and all test cases pass, great!

Output | Tests: 3 pass / 0 fail

- Example case: Correct answer
- Clicking on the button updates the username from the input: Correct answer
- Reference Hook was used to update username: Correct answer

Submit & next — Question time remaining: **38min**

Question 10 of 12 REACT Report an issue

This application should allow the user to update their username by inputting a custom value and clicking the button.

The *Username* component is finished and should not be changed, but the *App* component is missing parts. Finish the *App* component so that the *Username* component displays the inputted text when the button is clicked.

The *App* component should use the *React.useRef* Hook to pass the input to the *Username* component for the *input* element and for the *Username* component.

For example, if the user inputs a new username of "John Doe" and clicks the button, *App*'s div element should look like this:

```
<div><button>Change Username</button><input type="text"><h1>J
ohn Doe</h1></div>
```

There might be multiple App elements on the page.

Submit & next Question time remaining: **38min**

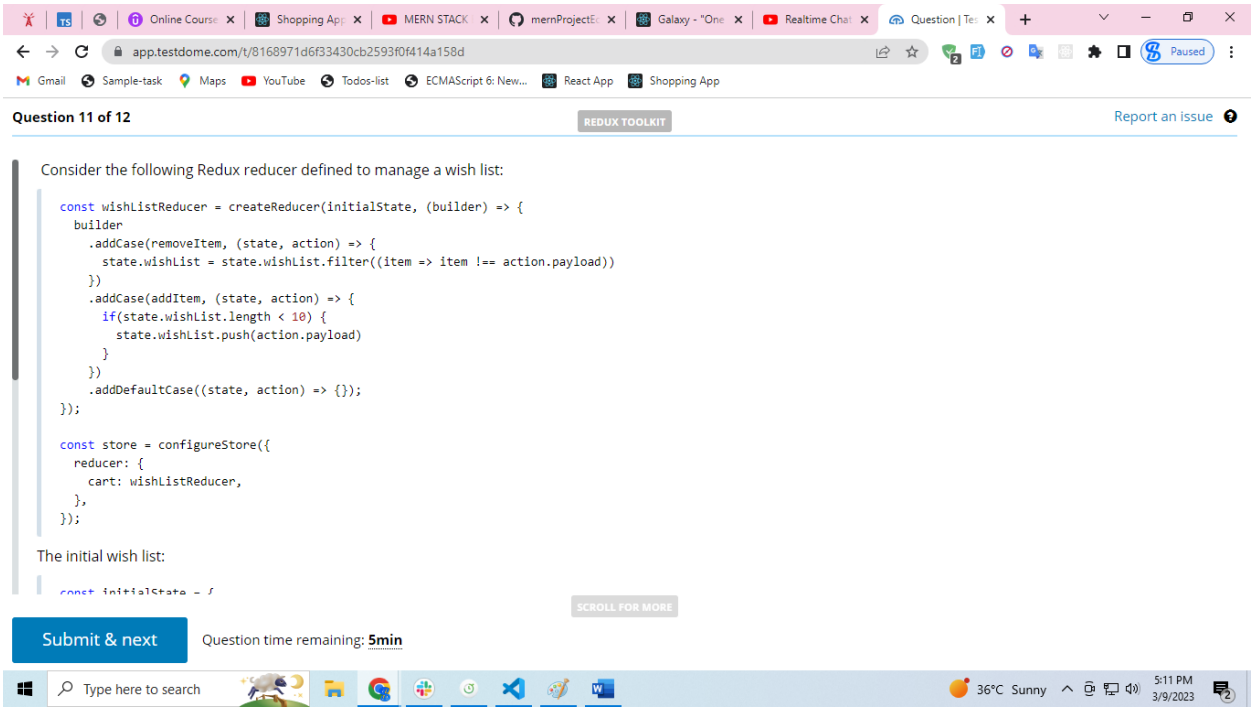
HTML5, CSS3, React v18 (available as React and ReactDOM) Copy to IDE Solution History

```
  }
}

function App() {

  const inputRef = React.useRef();
  const userNameRef = React.useRef();

  function clickHandler() {
    userNameRef.current.changeValue(inputRef.current.value);
  }

  return (
    <div>
      <button onClick={clickHandler}>Change Username</button>
      <input type="text" ref={inputRef} />
      <Username ref={userNameRef} />
    </div>
  );
}

document.body.innerHTML = "<div id='root'></div>";
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);

setTimeout(() => {
  document.querySelector("input").value = "John Doe";
  document.querySelector("button").click();

  setTimeout(() => {
    console.log(document.getElementById("root").innerHTML);
  }, 300);
}, 300);
```

Run Compilation OK and all test cases pass, great!

Output Tests: 3 pass / 0 fail

- Example case: Correct answer
- Clicking on the button updates the username from the input: Correct answer
- Reference Hook was used to update username: Correct answer



Question 11 of 12

REDUX TOOLKIT

Report an issue

```
  reducer: {
    cart: wishListReducer,
  },
});
```

The initial wish list:

```
const initialState = {
  wishList: [
    "Toy Car",
    "Football"
  ],
};
```

Considering the *initalState* and *wishListReducer* defined above, what is true about *wishListReducer?*

(Select all acceptable answers.)

- [x] The *addItem* action will have no effect on the state if there are more than 11 items in *wishList* in the initialState.
- [x] The *addItem* action will add an item to the *state* when the *payload* has a value that already exists in the *state*.
- [ ] The *addItem* action will be applied multiple times to the *state* when the *payload* is an array.
- [x] *removeItem* reducer will be invoked when the *removeItem* action is dispatched and the *payload* is *undefined*.
- [x] The default case will be invoked when the *state* is empty and *removeItem* action is dispatched.
- [ ] The default case will be invoked after every invocation of *addItem* or *removeItem* reducer, regardless of any changes to the state.

SCROLL FOR MORE

Submit & next

Question time remaining: **3min**

Question 12 of 12

REDUX TOOLKIT

Report an issue

A web application for a meteorologist's dashboard uses a Redux store to manage temperature data. The Redux store is configured with the Redux toolkit where the latest temperature data is fetched using the Redux *thunk* function.

Below is the code for the store's configuration:

```
const weatherURL = "http://example.com/temperaturejson";

const fetchTemperature = createAsyncThunk(
  "weather/fetchTemperature",
  async (arg, thunkAPI) => {
    const response = await fetch(weatherURL);
    return response.json();
  }
);

const temperatureSlice = createSlice({
  name: "temperature",
  initialState: { temperature: 0 },
  reducers: {},
  extraReducers: (builder) => {
    builder.addCase(fetchTemperature.fulfilled, (state, action) => {
      state.temperature = action.payload.temperature;
    });
    builder.addCase(fetchTemperature.pending, (state, action) => {
      state.temperature = "loading";
    });
    builder.addCase(fetchTemperature.rejected, (state, action) => {
      console.log(action);
      state.temperature = "unavailable";
    });
  },
```

SCROLL FOR MORE

Submit & next

Question time remaining: **10min**

```
  initialState: { temperature: 0 },
  reducers: {},
  extraReducers: (builder) => {
    builder.addCase(fetchTemperature.fulfilled, (state, action) => {
      state.temperature = action.payload.temperature;
    });
    builder.addCase(fetchTemperature.pending, (state, action) => {
      state.temperature = "loading";
    });
    builder.addCase(fetchTemperature.rejected, (state, action) => {
      console.log(action);
      state.temperature = "unavailable";
    });
  },
});

const weatherStore = configureStore({ reducer: temperatureSlice.reducer });
```

What is true about the behavior of *weatherStore?*

(Select all acceptable answers.)

- [x] A request will be made to *weatherURL* every time *dispatch(fetchTemperature())* is called.
- [x] *state.temperature* will be set as "unavailable" when *dispatch(fetchTemperature())* is called and *fetch(weatherURL)* throws an error.
- [ ] The *fetchTemperature.fulfilled* case will be invoked only when the response from *weatherURL* has a different temperature than the current state.
- [ ] The JSON response from *weatherURL* must have the same schema as the state object for the *fetchTemperature.fulfilled* case to be invoked.

SCROLL FOR MORE

Submit & next

Question time remaining: **10min**