Multiple Choice Questions

1. Which of the following is a library used in Redux?

[✓ A. React]

B. Library

C. All of the above

D. None of the above

2. Which of the following is the correct syntax to install redux?

`A. npm install redux;`

`B. npm install react-redux`

[✓ `C. npm install redux`]

`D. npm install react-redux;`

3. Which of the terms below is defined by a plain JavaScript object that describes what has happened?

A. Class

[✓ B. Action]

C. Function

D. None of the above

4. What does the code below indicate?

```
{
  todos: [{
    text: 'Eat food',
    completed: true
  }, {
    text: 'Exercise',
    completed: false
  }],
  visibilityFilter: 'SHOW_COMPLETED'
}
```

[✓ A. State of a todo app]

B. An action on a todo app

C. Todo is used as an object

D. A todo app reducer

5. Which of the following is NOT the function of a reducer?

A. To tie states and actions together

B. Take state and actions as functions

C. Return the next state of the app

[✓ D. None of the above]

6. What are the three main concepts of redux?

A. Function, State, Action

B. Function, Object, Reducer

C. Object, State, Reducer

[✓ D. State, Action, Reducer]

7. Which principle of redux does the following code indicate?

```
store.dispatch({
  type: 'COMPLETE_TODO',
  index: 1
})

store.dispatch({
  type: 'SET_VISIBILITY_FILTER',
  filter: 'SHOW_COMPLETED'
})
```

[✓ A. State is read-only]

B. A reducer is a function

C. Single source of truth

D. Changes are made with pure functions

8. In which principle of redux are pure reducers used?

A. State is read-only

B. A reducer is a function

C. Single source of truth

[✓ D. Changes are made with pure functions]

9. Which of the following is NOT a benefit of redux?

A. Availability of Developer tools

B. Excellent organization

C. Low cost

[✓ D. Ease of testing]

10. Which of the following entities CANNOT be combined with Redux?

A. Angular.Js

B. Vue.Js

[✓ C. Meteor]

D. None of the above

11. Everything in react is a _______.

A. Module

B. Package

C. Class

D. Component

12. What is ReactJS?

A. User interface framework

B. Interaction interfaces building library

C. Server side framework

D. All of the above

13.____________is the directory where react components are saved.

A. Inside js/components/

B. Inside vendor/components/

C. Inside external/components/

D. Inside vendor/

14. __________is the number of elements that a React component returns

A. 1

B. 2

C. 0

D. Multiple

15. What is the use of a webpack command?

A. Bundling modules

B. Transpiling all JavaScripts down into one

C. Compiling Redux commands

D. Running after a local development server

Practical Activity

Build a simple application for rating Questions

[This exercise will be split into two. The first part will be done on the first day of learning Redux, while the second part will be done on the second day of Redux]

The app in question should be able to rate questions.

Step 1

Modularize the base

Structure the components so that they fit perfectly into the Redux application. The code base can be found here.

Note

- The stopwatch component has its own local state - it does not depend on other components.
- The stats and counter components depend on other components
- The AddQuestionForm depends on other components - it contains logical information
- Structure the header and question components.

Step 2

Adding Redux

2.1 - Action Types

- Decide which components should take part in the Redux store. Only the questions have to made available to all components.
- Define the events that take place in the application for this specific state. These inlclude:
    - Changing the score
    - Adding questions
    - Removing questions

2.2 Reducers

Create the reducer functions, as shown in the code below:

```
export default function Player(state = initialState, action) {
  switch (action.type) {
    case QuestionActionTypes.ADD_PLAYER:
      return [
        ...state,
        {
          name: action.name,
          score: 0,
        },
      ];
    case QuestionActionTypes.REMOVE_QUESTION:
      return [...state.slice(0, action.index), ...state.sclice(action.index + 1)];
    case QuestionActionTypes.UPDATE_QUESTION_SCORE:
      return state.map((question, index) => {
        if (index === action.index) {
          return {
            ...question,
            score: question.score + question.score,
          };
        }
        return question;
      });
    default:
      return state;
  }
}
```

2.2 - Actions and Action Creators

To add a question, use the following action:

```
export const addQuestion = name => ({
  type: QuestionActionTypes.ADD_QUESTION,
  name,
});
```

2.3 - Create the Redux Store

Create a store in your index.js. Pass it as the main reducer and wrap it around the scoreboard component to avail the it to the entire application.

2.4 - Connect the container to the store

- Use mapStateToProps to assign the state to a prop value. You can assign the state of the questions as props.
- Use the following code to automatically dispatch actions that are created.

```
const {dispatch, questions} = this.props;
const addQuestion = bindActionCreators(QuestionActionCreators.addQuestion,
dispatch);
const removeQuestion =
bindActionCreators(QuestionActionCreators.removeQuestion, dispatch);
const updateQuestionScore =
bindActionCreators(QuestionActionCreators.updateQuestionScore, dispatch);
```

- Update the event handlers on the components accordingly - counter, question and scoreboard components)
- You don’t have to change the header and stopwatch components, since they don’t take part in the redux cycle.

Step 3

Add another component in the Redux App

We will need to display details to each question. Do the following:

- Add a new action type (select a question)
- Extend the reducer with a new switch case and additional state.
- Add a new action creator for selecting a question
- Create a new bindActionCreator in the scoreboard component
- Update mapStateToProps with the selected question index
- Create a QuestionDetail component to display details
- Update the event handler on the question component

Step 4

Implement ducks

The ducks concept helps develop a Redux application faster. Rather than keep everything modular, keep them in one file to be able to view everything at once.

Here’s how your file should look:

```
// Actions
const ADD_QUESTION = 'question/ADD_QUESTION';
const REMOVE_QUESTION = 'question/REMOVE_QUESTION';
const UPDATE_QUESTION_SCORE = 'question/UPDATE_QUESTION_SCORE';
const SELECT_QUESTION = 'question/SELECT_QUESTION';
// Reducers
const initialState = {
  questions: [
    {
      name: 'Do you like AI?',
      score: 31,
      created: '00:00',
      updated: '00:00',
    },
    {
      name: 'Do you like Engineering?',
      score: 20,
      created: '00:00',
      updated: '00:00',
    },
    {
      name: 'How many Redux Apps?',
      score: 50,
      created: '00:00',
      updated: '00:00',
    },
  ],
  selectedQuestionIndex: -1,
};
export default function Question(state = initialState, action) {
  const date = `${new Date().getHours()}:00`;
  switch (action.type) {
    case ADD_QUESTION:
      const addQuestionList = [
        ...state.questions,
        {
          name: action.name,
          score: 0,
          created: date,
        },
      ];
      return {
        ...state,
        questions: addQuestionList,
      };
    case REMOVE_QUESTION:
      const removeQuestionList = [
        ...state.questions.slice(0, action.index),
        ...state.questions.slice(action.index + 1),
      ];
      return {
        ...state,
        questions: removeQuestionList,
      };
    case UPDATE_QUESTION_SCORE:
      const updateQuestionList = state.questions.map((question, index) => {
        if (index === action.index) {
          return {
            ...question,
            score: question.score + action.score,
            updated: date,
          };
        }
        return question;
      });
      return {
        ...state,
        questions: updateQuestionList,
      };
    case SELECT_QUESTION:
      return {
        ...state,
        selectedQuestionIndex: action.index,
      };
    default:
      return state;
  }
}
// ActionCreators
export const addQuestion = name => ({
  type: ADD_QUESTION,
  name,
});
export const removeQuestion = index => ({
  type: REMOVE_QUESTION,
  index,
});
export const updateQuestionScore = (index, score) => ({
  type: UPDATE_QUESTION_SCORE,
  index,
  score,
});
export const selectQuestion = index => ({
  type: SELECT_QUESTION,
  index,
});
```

Step 5

Chrome Redux DevTools

Download the Redux DevTools Extension from here. Add the required line of code to your store.

```
const store = createStore(
        QuestionReducer,
        window.__REDUX_DEVTOOLS_EXTENSION__ &&
window.__REDUX_DEVTOOLS_EXTENSION__(),
);
```

The dev tools will help you develop and debug your Redux app.