WebDev Midterm Review

1. HTML
    a. [ok] best practice questions
        i. Proper use of TABLE -- use for layout is not appropriate. instead use for tabular data
        Proper

Heading 1	Heading 2		
Value 1,1	Value 2,1		
Value 1,2	Value 2,2		

        ii. Improper, use CSS instead

Column 1	Column 2	Column 3
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat

        iii. Improve user experience with bigger clickable areas, e.g, labels for
            1. especially small clickable UI elements: checkboxes, radio buttons … embedding in labels
    b. [ok] radio buttons mutially exclusive using same name
        i. <input type="radio" name="**tenured**"/>YES
        ii. <input type="radio" name="**tenured**"/> NO
        iii. <input type="radio" name="married"/>YES
        iv. <input type="radio" name="married"/> NO
    c. checkboxes can be groupped with same name
        i. <input type="checkbox" name="**genre**" value="HORROR"/>HORROR
        ii. <input type="checkbox" name="**genre**" value="SCIFI"/>SCIFI
        iii. <input type="checkbox" name="**genre**" value="DRAMA"/>DRAMA
        iv. <input type="checkbox" name="color" value="YELLOW"/>YELLOW
        v. <input type="checkbox" name="color" value="BLUE"/>BLUE
        vi. <input type="checkbox" name="color" value="RED"/>RED
    d. input id, value, placeholder, title (tooltip), name
    e. input date YYYY-MM-DD
        i. <input type="date" value="2020-02-24"/>
    f. select
        i. <select value="FEB" name="month">
            1. <option value="JAN">January (01)</option>
            2. <option value="FEB" selected>February (02)</option>
            3. <option value="MAR">March (03)</option>
        ii. </select>
    g. Focus

        i. <label for="uname">Username</label><input id="uname" type="checkbox"/>
        ii. <label>Username☐</label>
2. CSS
    a. float to create vertical columns
    b. box model margin, padding, border, width, height, top, left, bottom, right, position=absolute|static|relative
    c. style attribute is a bad practice <element **style**="color: blue"/>
        i. vs style tag <**style**> element { color: blue} </style>
        ii. vs link and best practices <**link** href="style.css" rel="stylesheet"/>
            1. many pages can refer to same style sheet
            2. single point of maintainance
    d. [NO] bootstrap -- no need to memorize the various classes
    e. basic CSS properties: color, backrgound-color, margin, padding,
    f. no border radius, border styles, fonts
    g. order of priority of style rules when applied to same element
        i. if multiple CSS rules apply to same element
            1. <style> h1 { color: yellow } h1#blue { color: blue } </style>
            2. <h1>I am yellow</h1> <h1 id="blue">I am blue</h1> <h1>I am yellow</h1>
            3. more specific rules have more priority
            4. document order
3. [NO] RESTful
    a. @GetMapping, @PostMapping, @DeleteMapping, @PutMapping
    b. @PathVariable, @RequestBody
    c. POST, GET, PUT, vs DELETE
        i. POST -- creating data
        ii. GET -- for reading data
        iii. PUT -- to change data
        iv. DELETE -- to remove data
    d. nouns are plural
    e. Relationships
        i. **One To Many**: one A has may Bs, and each B has many Cs, and each C has many Ds, e.g., course has many sections, college has many departments
            1. course 1 -- * section
            2. college 1 -- * departments
            3. A 1 -- * B
            4. A 1 -- * B 1 -- * C 1 -- * D⇒
                a. /A - means all As
                b. /courses - means list of all couses
                c. /A/123 - means the A with ID = 123
                d. /courses/123 - means course whose ID = 123
                e. /A/123/B - means all Bs for A with ID = 123
                f. /courses/123/sections - means all sections for course 123
                g. /sections/123/courses - makes no sense
                h. /B/123/A - no good
        ii. **Many to Many**: each A is related to many Bs, and each B is related to many As, e.g., Actors * --- * Movies
            1. A * --- * B ⇒
                a. /A - means all As

        b. /actors - means all actors
        c. /A/123 - means the A with ID = 123
        d. /actors/123 - means actor whose ID = 123
        e. /A/123/B - means all Bs for A with ID = 123
        f. /actors/123/movies - movies where actor 123 has been in
        g. /B/123/A - means all As for B with ID = 123
        h. /movies/234/actors - actors for movie 234
        i. /students/123/sections -- all sections for which a student is enrolled in
        j. /sections/234/students -- all students enrolled in a section
    f. .then() and NO async / await
    g. No query predicates -- /qwe/wer/ert?asd=sdf&dfg=fgh
4. NO UML
    a. A 1 -- * B
    b. A * -- * B
    c. One to Many
    d. Many to Many
    e. Inheritance
    f. Cardinality 1, *, 1..M
5. NO JPA - NO DATABASE STORAGE
    a. @JsonIgnore
    b. @OneToMany(mappedBy), @ManyToOne
    c. @Entity, @Table, @Id, @GeneratedValue
    d. .save(), .findAll(), .findById(),
    e. NO Repositories
6. React
    a. [no] index.html
        i.
    b. [no] index.js:
        i. ReactDOM(<App/>, document.getElementById("**root**"));
    c. function components
        i. const [stateVariable, setStateVariable] = useState(initialValue);
        ii. const [qwe, wer] = useState(false);
        iii. console.log(qwe); ⇒ false
        iv. wer(true); ⇒ qwe = true;
    d. Maps
        i. const todos = [123, 234, 345];
        ii. return(
            1.
                a. {
                    i. todos.map(t => {t})
                b. }
            2.
        iii.)
        iv. return(
            1.
                a. {
                    i. todos.filter(t => t<300).map(t => {t})
                b. }

            2.
        v.)
    e. Conditional rendering
        i. const loggedIn = true;
        ii. return(
            1. {loggedIn && <h1>Welcome</h1>}
            2. {!loggedIn && <h1>Go away</h1>}
        iii.)
    f. [NO] const FunctionComponent = () => {} vs class ClassComponent extends React.Component { }
        i. class components
            1. state = { … } NO
            2. this.setState() NO
                a. this.setState({ … set new state disregarding old state ... }) NO
                    i. it's bad practice to use this.state to calculate next state
                b. this.setState(prevState => { … return new state based on old state … }) NO
            3. event handlers NO
            4. life cycle functions NO
                a. componentDidMount() NO
                b. componentDidUpdate() NO
            5. implementing the controller NO
            6. render() { return(…) } YES
        ii. function component
            1. easier to test
            2. focus on implementing the view
    g. onClick={callSomeFunction~~(p1, p2)~~} vs onClick={() => callSomeFunctionWithParameters(p1, p2)}
    h. onChange()
        i. onChange((event) => {updateValue(event.target.value)})
7. Router
    a. <Router></Router>, <Route path="/path1" element={<MyComponent>}/>, <Link to={}></Link>
    b. <Route path="/some/path/with/:parameter1/:parameter2" element={ComponentExpectingParametersInProperties}/>
    c. Reading property changes useParams()
    d. NO history.push() -- NO
8. ES6
    a. export default value vs export value
        i. export default Something
        ii. export default {findAllTopics, findTopicById}
        iii. [NO] export default class MyClass { … }
            1. import Something from "..."
            2. import React from "react"
            3. Only one default is allowed
        iv. export Something1 // without default
        v. export SomethingElse2 // without default
            1. import {Something1, SomethingElse2} from "..."
            2. import {Provider, createStore} from "react-redux"
    b. import defaultExportedThing, {versus, exported, values} from './from/some/js/file'

c. Spreader function
    i. const newObjectCopy = {...oldObject, overrideSomeProperty: widthNewValue}
    ii. return {...state, widget: action.widget}
    iii. const newArray = [...oldArray, appendNewElement]
    iv. const newArray = [prependNewElement, ...oldArray]
    v. return {
        1. widgets: [...state.widgets, action.widget]
    vi. }
d. map, filter JS functions
    i. const newArray = oldArray.**map**(instance => { return element })
    ii. const newArray = oldArray.**map**(instance => {
        1. if(instance._id === action.instance._id) {
            a. return action.instance
        2. } else {
            a. return instance
        3. }
    iii. })
    iv. .filter()
e. Destructor syntax: {p1: v1, p2: v2} ⇒ f = ({p1}) => console.log(p1) ⇒ v1
    i. http://es6-features.org/#ObjectMatchingShorthandNotation
f. If property and value have same name, you can use an abbreviated syntax
    i. const a;
    ii. const b;
    iii. const c = {a, b} ⇒ c = {a: a, b: b}
    iv. const findCourses = () => {}
    v. const findCourseById = () => {}
    vi. const api = {
        1. findCourses, findCourseById
    vii. }
    viii. const api = {
        1. findCourses: findCourses, findCourseById: findCourseById
    ix. }
g. arrow function
    i. const someFunction = () => (5) ⇒ const someFunction = () => {return 5}
    ii. const someFunction = oneParameter => "doesn't need paren"
        1. then(actualLesson => dispatch({type: "CREATE_LESSON", lesson}))
            a. then((actionLesson) => {
                i. return dispatch({type: "CREATE_LESSON", lesson: lesson})
            b. })
    iii. const someFunction = (oneParam, twoParam) => "do need paren"

9. Redux
a. Just one question
b. create a reducer
    i. pure function (state={}, action) => { … calculate new state … }
    ii. implements a FSM
    iii. dont need to import react
c. combine reducers [NO]
    i. create a namespace for each reducer

ii. import reducerA from "..."
iii. import reducerB from "..."
iv. import reducerC from "..."
v. const rootReducer = combineReducers({
1. reducerA: reducerA, reducerB: reducerB, reducerC
vi. })

d. create a store
e. provide that store
f. [NO] state mapper
g. [NO] dispatch mapper
h. [NO] connect(state mapper, dispatch mapper)(the component)
i. [NO] hook it up to a async data source
i. const dm = dispatcher => ({
1. propertyFunction: (param) =>
a. fetch()
i. .then(response => response.json())
ii. .then(data => dispatcher({type: …, data})
ii. })
j. [NO] best practices
i. actions in their own functions
ii. reducers in their file
iii. action strings as constants

10. Yes jQuery
a. $ vs jQuery
b. order of loading jQuery
i. $(main) ⇒ main is called when document is ready
ii. or declare at bottom of body
c. $(...CSS…) -- grabs element from DOM that matches the CSS
d. $(...HTML…) -- creates a new element
e. $.append(), .empty(), .click(), .html(),
11. Abbreviations
a. HTML, DOM, CSS, SPA, SQL, JPA, etc…

Q11 → 1pt