Back-end and frontend exams,

Addition: Full-stack open reading list

by Juhani Välimäki

Lot of what we already had in backend and frontend exams. But also some completely new topics.

This list is more for tracking the reading than real list of what can be formulated as exam question. So there are less exam question topics than points in these lists.

Not yet divided by front and back exams. As the list follows the Full-stack open course chronologically.

Part 0b. Fundamentals of Web apps

- Not really much 100% new stuff for you, don't worry, just recapping.
- Go through rather fast
- In Full-stack open you can, but don't have to, visit the offered extra info links. If do so, just spend max 30 seconds or something like that there in the side track link
- It is good to recap at least these concepts
    - http request, GET method
    - looking into http request headers
    - DOM manipulation with (DOM API) JavaScript running in browser
    - POST method, request with headers and body
    - AJAX, https://en.wikipedia.org/wiki/XMLHttpRequest, the xhr object given to you by the DOM API JavaScript implementation of the browser
    - SPA idea (Frontend)
    - JavaScript libraries for Web UI and/or datamodels: jQuery, BackboneJS, AngularJS, React, Redux, VueJS
- "JavaScript fatigue" is true. Many new versions of the libraries (e.g. React) utilize _all_ new features that appear in EcmaScript standard. Thus people need to learn both new version of the library and new version of ECMAScript.
- Do the exercises if have a lot of time. Otherwise you can just learn the theory fast first

Part 1a. Introduction to React

- npx **both** downloads and runs the create-react-app tool
- what kind of React project template is created by it?
- learn/recap the function based React
- JSX, XML-like but not really XML, e.g. this is not valid XML: <xyz def={abc} />   (attribute values should be inclosed in " " or ' '
- props as React component function parameter
- start the JSX code on same line as return statement, or wrap it into (   )
- React function should return
  - one root component,
  - or an [ ] array of components
  - or a fragment, empty mother element with other elements inside
- exercises

Part 1b. JavaScript

- babel – transpiler library. From new ECMAScript versions to older versions
- const t = [1, -1, 3];   const t2 = t.**concat**(5); // doesn't change original array, returns the new
- destructuring assignment
- object literals are not JSON, even if they look almost similar. JSON is "text" but object literals are JavaScript code.
- referring to the object members with 'indexer' brackets: person1['first name'] = "Joe";
  - works even for illegal member names, like having spaces or starting with digits
- object-oriented JavaScript **was** used in old React, now that is forgotten. Nowadays we use function and Hooks -way React to define React components
- arrow functions, nothing new
- exercises
- old stuff: arrow function doesn't tie the keyword 'this' to itself (to the arrow function object) but refers to outer lexical context. That is, to the React component object.
- method = function that has been attached to an object. When run, the keyword this refers to the object if you have used arrow function notation => please use
- (funny fact, if you take the function=method object out of the object, it loses the this -reference and kind of becomes a non-method function)
  const `referenceToGreet = arto.greet`
- setting up timers in JavaScript is easy:
  - `setTimeout(arrowFunctionToBeCalled, 1000)  one shot after 1s`
  - `setInterval(arrowFunctionToBeCalled, 5000) periodically run`
- multiple timers are independent of each other
- again class in JavaScript can be skipped, we rarely use nowadays
- Maybe this fast: https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

Part 1c. Component state, event handlers

- component helper functions wrapped inside React component functions have access to e.g. props of the React function/component
- destructing the props with three different ways (so that it's not too simple 😊)

```
        o  const name = props.name;
        o  const age = props.age;

        o  const { name, age } = props


        o  ({ name, age }) => {
      // looks like an object created, but no. To vars created and the
props is destructured to those vars.
```

- relax and try to see how the three ways above lead to almost same (the last one is almost same even if two vars are created.
- useState -hook => Stateful component the modern way (Forget the old object-oriented React and setState)
- useState is used to map 1. one state value holder and 2. its setter function and 3. provide the initial value
- don't be fooled by it being const - React will handle the really running of the components, we just attach hooks to certain events
- passing eventhandler arrow function objects to components in their probs
- if you need e.g. button click to call function foo(), do not call there immediately foo(), but create an arrow function which would, when later called, call the foo() inside its function body. = you are not doing something, you are giving the system chance to do something later, right?
- passing data (e.g. pieces of parent state) to child components via props
- the concept of different kind of React components. Some e.g. only show data and provide buttons for doing something to just that data item. Some are parent components that have several child components. Those typically fetched all the data.
- (re-)rendering at least in these cases: 1. state changes, 2. props change, (((~~3. render method called~~))) I need to think what are the other real cases in modern function + hooks way.
- we want to have the React components follow SRP, single-responsibility principle. Doing just one thing and if not, then refactoring, e.g. splitting it potentially to sub-components. Sub-components defined outside of the parent component, so that you can reuse them possibly in another view too.

Part 1d. A more complex state, debugging React apps

- several pieces forming the state OR one more complex object forming the state
- initial value kind of gives the format of the state, and methods that update or read the state need to follow the same format, e.g. member names.
- methods updating the state might **first** utilize the … spread notation to take a copy of the current state as basis, and **then** change the wanted parts in it.
- we will thus create a new object that will replace the old state object. We cannot mutate the old state object. (this was the same in the traditional class-based React)
- array.concat (from earlier) will be handy when the model is an array. As it will not mutate original array, but will return a totally new array to be used as the state value. (unlike e.g. array.push() that would mutate the original object)
- in conditional rendering the authors have used if return return -structure (two returns in row). If true, return something. If was true, the method will end. Otherwise return what the second return returns. You like that or not? To me if-else would be more elegant. A bit nerdy solution. Even the block for the if is left out. Well it's JSX+JS combo so…
- Old React, read but skip mostly
- console.log('props value is', props)   // For better debugging messages, try it out
- React developer tools
- rules of Hooks we have also in the Frontend exam reading list already, but good to read also here
- once again the idea of passing a function object (defined with arrow function syntax) to the onClick, not executing code, but passing executable code
- all in all read the code examples here, and ask if don't understand everything
- in addition to reuse, components should be defined outside of other components so that the system can treat and optimize them as separate, but possibly together orchestrated components.
- Useful reading for React, two links to learning sites. Certainly good, now just think do you want to advance fast to tasks, or make a deeper dive into React. Up to you and depends on your situation
- exercises: nice recap of the topics. Recommended if not in hurry to next chapter.

Part 2a. Rendering a collection, modules

- Again about debugging and debugging tools. Never emphasized too much!
- (Snippets maybe come more handy in daily work than in learning. I like to type all myself when learning)
- Higher order functions (functions that are passed functions, functions that return functions to be executed, map/foreach/filter/find/reduce. Interesting topics, we have seen a lot of that topic, but not all)
- (again event-handler arrow functions returned from another arrow function that is passed some needed id:s or other config)
- (BTW. All in all you might want to use only arrow function syntax with modern React!)
- Rendering collections involves: 1. outer mother element (finally e.g. ol, ul, table, div), 2. map/foreach/… creating child elements (finally e.g. li, tr, div) **and** 3. the unique attribute 'key' to the child element
- Nowadays, when a React component can return an array of elements, the map-function is the most used.
- Never use array index for the key values in real cases. Use instead the UID or id that the item has in database. Or a encrypted hash based on that id. Those id:s work.
    - whereas the one from array gets ruined when an item in the collection is removed. '
- Again visiting prop destructing, React component refactoring and splitting all views into single-responsibility principle components with child components further inside.
- And again remember not to define Component inside the mother component's code.
- How the root React component (Start of the SPA application) get's 'injected' to the web page = to the DOM? What libraries and templates and components are involved? Just study it fast. index.js, App.js, index.html, react-dom, react
- node modules referred to without the folder path, own react components etc. JS files with ./and_so_on
- exercises

Part 2b. Forms

- `event.preventDefault()` used to stop the normal behavior of HTML forms, with submission to backend etc. We want to stay in the same SPA and let SPA do the routing of the views. And React code to handle the “submit” button behavior too.
- The ‘event’ refers to the Web browser UI event object. Has information about e.g. event target, how long was some button pressed. Etc. etc. Think e.g. all things related to what you can do with mouse in UI.
- `onSubmit={addNote} //`When user presses the submit, what event-handler function should be called (addNote).
- ‘controlled conmponents’ having own mapping between html input fields and the state of the component. When value in input field value changes, change event-handlers also update the components state, and thus also update any components showing the state.
- used the model where in the state there are all the items, plus the ‘temp’ for just in-read input value(s) before they/it go/goes to the all items collection
- nice example, but need to read the final/full version below
- ternary operator (conditional operator)   a ? b : c used in conditional rendering in JS or JSX code
- always use === in JavaScript, unless know exactly some rare special case, and have tested all possible probs out.
- quite nice example continues. Not a bad idea to write all of it to a pristine create-react-app project template?
- exercises

Part 2c. Getting data from server

- Using tool json-server (Instead of Node.js server with e.g. Express) in this first example.
- fast output API generated based on the JSON file in the folder
- (possibly JSONView plugin needed for Browser if the browser doesn't show the results nicely)
- use AJAX via some promises-based library, e.g. Axios, no need to learn those old directly-using-the-DOM-api-ajax-objects -ways used here
- JavaScript engines are single-threaded, even the one behind the Node.js server. (But Node.js simulates multi-threading as well as it can. Browser JavaScript on the otherhand is eager to hang or become non-responsive.)
- Web workers are for running multiple threads in Browser JavaScript. You might see one in the create-react-app
- npm, the **n**ode **p**ackage/module **m**anager, that should not be called node package manager…
- npm used also to start / build / clean the project
- package.json   (file you don't want to use. package-lock.json etc. can be removed and generated again. But this is the place where the dependencies and some other tooling/scripting are defined)
- development dependency = only included in the development version of the product, not to the build version!
- npm run server   (read this script from the package.json and understand how it works)
- like always, just the last version of Axios usage starts to be what we would use.
- Even there the request object could be maybe configured first and then sent. (URL, parameters, body data)
- useEffect -hooks   // do something else too, than the React component state/props <-> render cycle. =side effects. E.g. run the AJAX call to backend.
- the nice example starts to expand, but of course in real create-react-app projects we would have a lot of folders and files. E.g. one for views, one for SPA routing files, one for other components, …
- the create-react-app project is, by default, development environment with e.g. autorefresh of the running code after changes
- the real build version of the project does not run dynamic code in Node.js server, and doesn't have all your folders and files separately. The built version runs only in the browser JavaScrip environment after it has downloaded the few generated and bundled/packed HTML and JS and CSS files. Just running then JavaScript in browser and of course also making AJAX calls to backend.

Part 2d. Re

- Aaaa

Part 2e. Re

- Aaaa

UNDER CONSTRUCTION