**THEORY QUESTIONS ASSIGNMENT**

Full Stack Stream

(Maximum Score: 100)

KEY NOTES

* This assignment is to be completed at the student's own pace and submitted before the given deadline.
* There are **8** questions in total and each question is marked on a scale 1 to 20. The maximum possible grade for this assignment is 100 points.
* Students are welcome to use any online or written resources to answer these questions.
* The answers need to be explained clearly and illustrated with relevant examples where necessary. Your examples can include code snippets, diagrams or any other evidence-based representation of your answer.

|  |  |
| --- | --- |
| **Theory questions** | **Points allocated per Question** |

1. What is React? (*E.g. Consider: what is it? What is the benefit of using it? What is its virtual DOM? Why would someone choose it over the standard HTML / CSS stack?)* (15 marks)

React is a JavaScript library developed by Facebook, which is used to create user interfaces. It allows you to break down code into small, individual parts called components, which when combined can form a complex interface. This is beneficial as it means that you can change one section of an app at a time by editing the relevant component rather than having to update the entire app. Components are also reusable, increasing efficiency. A DOM (document object model) is a live tree-structured representation of the objects in a webpage which is created when the page is loaded and used to display the page on screen. In react, a virtual DOM is used alongside this. This is a representation of the DOM, but it does not have the ability to change what the page displays. Because of this, it is much faster to edit the virtual DOM than the DOM itself because time is not spent updating the page. When a change is made, the entire virtual DOM is updated, but because this doesn’t require the page to reload it happens quickly. Then, the virtual DOM is compared to how it was right before it was edited, and the changes are identified. React is then able to figure out which parts have changed, and only those objects are updated in the real DOM, rather than the entire thing. This is therefore fast and efficient. This is one of the advantages that React has over the standard HTML / CSS stack, as for example if you’re developing a website with a lot of user interaction, it makes sense as the entire page won’t have to be rendered every time a user does something so will lead to better performance. React also makes debugging easier than in the standard stack, as the bugs are easier to find if the components are separated. Components being reusable means that they can also be used across multiple interfaces, increasing productivity. JavaScript-heavy pages can be difficult for search engines to read but React can use server-side rendering to solve this. React also has a structured way of writing code, so makes it easier for collaboration as others are more able to work out what the code is doing, and the components also help this as it’s easy to find the code for specific functions.

1. What are Props? What is State? What is the difference between them? (10 marks)

Props are properties of objects, used in React. They are arguments which are passed to functions and are like attributes in HTML. They are useful when reusing components, as for example if a button is used on multiple pages, the same component can be used but the prop can change the text that is displayed on the button, e.g. the parent component could say:

function Page(props) {

return <button type=“button”> Click to go to page { props.number } </button>;

}

And then a child component would be like:

function Number() {

return (

<>

<Page number="seven"/>

</>

);

}

This would then produce the following on the webpage:

Click to go to page seven

While props are used to pass data between components, state allows components to manage their own data. The difference is that while props are data passed from a parent component and can’t be modified by the child, states allow components to create and manage their own data. Therefore, props are more useful for when a component needs to be used multiple times. When the state changes, it triggers the affected component to update in the DOM, reloading that component in the webpage. Props however are not reactive, and do not cause the component to reload when they are changed. This is a simplified example of using state:

this.state = {

page: "seven"};

}

render() {

return <button type=“button”> Click to go to page { this.state.page } </button>;

}

Which would produce the same result as the previous example.

1. What are React Hooks? How do they differ from existing lifecycle methods? (10 marks)

Hooks allow functions to have access to features such as state without having to use a class. This makes the code much simpler. They differ from existing lifecycle methods as without hooks, classes would have to be written with dedicated render functions to be able to access the lifecycle methods. For example, the lifecycle method componentDidUpdate would have to be called to update the DOM in response to a prop changing. With hooks, there does not need to be a class, the required lifecycle method doesn’t need to be specified, and the code does not need to include render. Here is an example of without hooks:

class Component extends React.Component {

componentDidUpdate()

console.log(“abc”);

}

render() {

return <h1>xyz</h2>;

};

The same could be done in a more simple way with a useEffect hook:

const Component = () => {

useEffect(() => {

console.log(“abc”);

});

render() {

return <h1>xyz</h2>;

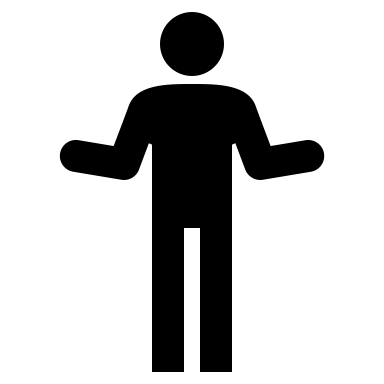
};

1. Design the perfect door - what should it look like, what are the components for it? What design heuristics should it follow, and how does your design match? What made you choose this design? (20 marks).
   1. *Consider in particular (likely need to do independent learning*): *who are your stakeholders? What are their personas? What are the doors requirements and how will your stakeholders benefit from your solution?*

For the door to be perfect, it would need to have the following functions:

* Two states, open and closed
* In the closed state, physically separates two environments
* In the open state, it should allow free movement between the two environments.
* Must have a mechanism to switch between the two states, for example a hinge or a sliding mechanism [1]. It should have an intuitive method of actuating the mechanism, for example a door handle [2].
* It should be able to be installed in a wall [3].
* It should be made of materials that are suitable for withstanding the environmental conditions it is exposed to [4].
* It should be appropriately sized for its intended user [5].
* The door should be easy to install to repair.
* Sturdy but not too heavy to open.
* Obvious whether it requires something to open, such as a key or an ID card.
* The door has an appropriate level of security and privacy for what it’s being used for.

Based on the required functions, a design of a door could look like:



[1] Hinges

[2] Door handle

[3] Door frame

[4] Timber

[5] 2m Height

This design follows the heuristics of being intuitive to open, reducing the mental load on the user as they see it and know that the handle needs to be pulled. It’s simple and only requires one step, pulling the door open. The required component needed to open the door, the handle, is easy to see so the user doesn’t waste time. The height is standardised to fit users. I chose this design as it an example of a door that is commonly seen as it’s effective and intuitive.

Stakeholders include:

* Manufacturers, such as a timber processing plant. They look for a design that is cheap and quick to manufacture to reduce costs
* Suppliers, such as a wholesale construction materials company. They benefit from a door that is easily marketable to customers.
* Customers, benefit from it being reasonably priced so they can justify buying it, and easy to install and repair so it’s less maintenance from them.
* A user, who is anyone that will utilise the final product. Users prefer a design that is easy to use with a visually attractive, intuitive design so they don’t waste time and it’s aesthetically pleasing.

1. What is Angular, and how does it differ from React? *You may need to conduct independent research and learning for this* (10 marks)

Like React, Angular uses components. However, React is a JavaScript library used to build user interfaces while Angular is a front-end framework based on TypeScript which is used to develop web applications. Being a framework means Angular provides more functionality than React, such as templates, routing, and testing/debugging with a single tool, however this also means it’s less flexible – React allows choice of tools, architecture, and other libraries which is can be packaged with. While React is based on the virtual DOM, Angular uses the Model View Controller to render data - the user interacts with the Controller which the Model takes commands from to send updates to the View, which is what the user sees. As Angular does not have a virtual DOM, React is much faster. React uses a one-way data binding method while Angular uses two-way. This means that to change an element of the user interface, the corresponding model state must also be updated. The two-way method in Angular means that the model state is automatically changed when a change is made.

1. Please describe Redux in as much detail - especially consider: *why would someone use it? What is it? What's the benefit of using it? Are there any potential drawbacks to using it? How can it be added to a project? What is dispatch, provider, actions, etc?* (15 marks)

Redux is a state container, storing the state of variables within an app, and it is often used as a state management tool with React. For example, if a user votes for something, the application must add that vote to the count and maintain it internally. State is read-only, and the storage is accessed by components by dispatching actions, which triggers the change by describing what change needs to be made to the state, and then going through a reducer, which accepts the action and the current state and returns the updated state. The provider component is used to make the Redux store available to components which need to access it. Redux is used as it is useful for large scale applications, where multiple states must be handled from multiple components efficiently. Redux prevents re-renders, improving performance. Redux also provides useful features such as editing live code and time travel debugging. However, a drawback is that returning as states are immutable, new states must be returned every time there is an update, and this can use a lot of memory. To add Redux to an app, after installing the packages a reducer must be created which accepts a state and an action. Then, a Redux store is created which allows actions to be dispatched to modify the state. Redux and React are connected using an NPM library, and the provider is used to connect the component with the redux.

1. Please describe Linux in as much detail as possible (feel free to use notes made during lessons, or draw from the lesson directly!). Especially consider: *what is its history? Why would someone use it over other existing operating systems? How does Windows and Mac OSX differ to Linux? How does Linux function, what are some unique features to it? How can it be installed today?* (10 marks)

Linux is based on UNIX = a simple operating system first developed in the 1960’s. Linux was created by Linus Torvalds, who combined his name and UNIX. Linux was created to replace UNIX, copying parts of it such as commands and file hierarchy while being more efficient, and became popular as other GNU’s were taking too long to create OS kernels. Linux was shared freely on the basis others would share changes made. There are many versions, such as Ubunu and Debian. Linux is a very popular choice for servers, as unlike Windows and MAC it is free and open source, and is more secure, flexible, stable, and reliable. Cost of maintaining a Linux server is lower and it has more support for different hardware too. Linux is unique as it can be customised - a user can pick and choose the features they need, and as Windows and Mac are much more popular it’s less likely to be targeted by hackers and malware. Linux is also better for privacy as it does not collect data.

Mac OSX has a similar origin to Linux – based on UNIX, but with a kernel and interface developed by Apple to differentiate themselves. As they are both based off of UNIX, Mac OSX and Linux share many similarities. However, Linux can be used on any device while Mac can only be used on Mac devices. Parts of MacOS are closed while Linux is completely open. Linux can be installed by downloading the OS onto a bootable USB/hard drive and then booting it on the destination system and following the steps to install.

1. What are they, and which is better between Class components and Functional components? Provide a discussion. Consider: *Go deep - how does each one work? What is the unique properties or behaviours to each one? Why would someone use one over the other? What are the advantages and disadvantages of each one? Who benefits from these advantages and disadvantages, who is it suitable for?* (10 marks)

Components are self-contained, reusable sections of code which can be defined as classes or functions. A class is a broad category used to create objects, which are instances of the class. A function is a chunk of code that does a specific task.

Class components work by returning a React element when a React component is extended, and a render function is created. E.g.:

class Colour extends React.Component {

render() {

return <h1>Red</h1>;

}

}

Functional components are simpler, defining components by writing a JavaScript function:

function Colour {

return <h1>Red</h1>;

}

Unique properties of each are that class components require extension of the react component and a render function to return the element, while functional components don’t require this. Because of the simplicity of functional components, setState cannot be used as it comes from React.Component which is extended from in class components, so in the past class components or props needed to be used to set states. However now that hooks have been introduced, the useState hook can be used in functional components. Functional components also were not able to be use lifecycle events for the same reason before the introduction of hooks. This meant that before hooks class components were easier to use for lifecycle events and setting state. Now, not only do hooks provide a solution, but functional components have other benefits compared to class components such as less code which is easier to read and test. However, classes still come in useful for tasks which require polymorphism for example, and hooks aren’t available for every single lifecycle method yet. Class components also still need to be understood for working with legacy code.