Planning React Apps

Prior Knowledge

- Be able to create React components.
- Be able to create and update state using `useState`.
- Understand the flow of props in a React app.

Learning Objectives

- Be able to break a React App into components from a wire-frame.
- Be able to plan what data will need to be held in state.
- Understand how to decided which components to hold state in.

Creating new react apps

When creating a new React app there are a lot of potential configuration options to get things up and running. In a production grade app we are very likely to be using a framework for addition features. We'll add these in as we go so will be using Vite to start a new react project without a framework

This is a tool that will allow us to setup new React applications with the default recommended settings and configure build tools to cut down the amount of boilerplate code we have to write.

New apps can be made using

```
npm create vite@latest
```

This will prompt you to setup your app with the following

1. A name. A new folder with your project name will be created
2. A framework: Vite can setup several frameworks but we'll choose `React`
3. A variant: We're not using Typescript or SWC so choose `Javascript`

nb When naming your apps it's generally a good idea to use snake or kebab case to avoid capital letters (directories are case sensitive on linux but not on MacOS or Windows so always using lowercase is a good idea.)

Once this app is setup you can remove any of the files you don't need and start creating your own app.

Package managers

An alternative package manager to `npm` is yarn. They both work from `package.json` files and are essentially interchangeable these days (there used to be more tradeoffs in the past but both are solid options.) You may see it mentioned in docs from time to time but there's no need to use it, npm works just fine.

```
yarn create vite
```

If you don't have `yarn` installed then you can use the command normally.

Wireframing and Components

When planning an app the first steps are to do some basic design work. In a professional environment this will often be handled by dedicated designers and for our purposes we will use some basic wireframing. Wireframes are a schematic, or blueprint, of the structure of a website. They do not need to come up with every aspect of the style or visual design and are focused around the functionality and general structure of our app.

These can be created using a number of tools or good old fashioned pen and paper. A quick sketch is enough for now but if you'd like to try them out some popular free tools are:

- Excalidraw
- Figma
- Miro

Once you are happy with the structure of you App separate out the functionality into React components. You can have as many or as few components as you would like and the decision about what to group together is up to you. Things to consider when deciding on your components are:

- Abstraction - Grouping related functionality together. Websites tend to be made up of sections which make for ideal components. Things like `header`, `footer`, `nav` and `section` tags typically make for good components as all of the logic to do with that section can be grouped together.
- Re-useability - Components are re-useable. If you need some functionality repeating then abstracting the logic into it's own component allows you to re-use that component in several places.
- Complexity - Adding more components makes your component tree more complex. Is the functionality you are extracting complicated enough to warrant adding another component. If it's simple enough to just render in the parent you may not gain any value from extracting it to it's own component.

Once you have decided on your components sketching out a `component tree` is often helpful to visualise the parent-child relationship.

Planning State

Once your component tree is established decide on what information your app will need to hold in state. Keep this state to a minimal, single source of truth. The structure of the state is up to you, as long as you can track all the information which will vary over time.

For each state variable make a list of which components will need access to that state in order to either read or update that state. This state needs to be declared in one of your components. That component must be high enough on the component tree so that every component that needs access is beneath it. This way the state and functions to update it can be passed down on props.

A good rule of thumb to follow here is to `place your state as low as possible, but as high as necessary`.

Repeat this process until all of your state has been placed on your component tree. This should give you a nice structure to work from with the important decisions about state and components being made before you jump into the code.

Don't worry if you make a mistake and place state to low on the component tree. You can always lift state up and move it to a higher component later on.