1. **The code base has put each component in a separate file and directory structure.**
   * **Why do you think that was done, what are the advantages / disadvantages?**

**Advantages**

You want to have your components stored in unique directories so that there is no chance of confusion as to what file you’re using when the code base becomes more complex.

Having components in separate folders also means the components are modular, they can be reused and maintained separately from other parts of the application. This allows for less chance of breaking changes when editing the component. It also makes debugging easier as you can narrow in on a single file that has a single responsibility rather than having to search through components and potentially many lines of code stored in one file.

Components being kept separately also makes unit testing easier, as the component can be tested in isolation.

**Disadvantages**

Using this concept can significantly increase the amount of files and directories in a project. This can increase overhead and reduce efficiency if many files are being read. This can add latency.

Having many files and directories can also make navigation within the project more difficult as the project becomes more complex. This can also lead to fragmentation where it is difficult to discern how many different components are interacting with each other.

1. **Thinking about a production ready app, what do you think is missing from all the examples?**

There needs to be extensive testing. There should be unit and E2E tests, which can be run locally but also embedded into the CI/CD pipeline to the production repo to ensure quality control.

I would also hope there were more comments throughout the code. Code has to be picked up by people that haven’t written it, and having comments helps speed up other colleagues being able to digest your code and what a certain function or component is doing.

1. **Are you familiar with Redux? If so what is the basic idea behind it and what would be a good case for using it?**

Redux is a state management tool. It allows you to create a ‘store’ of global state that can be accessed from anywhere within the application rather than having to pass props to components individually.

With Redux, the current state is immutable, so behind the scenes it creates a copy of the original state, changes that copy and then stores that as the latest state. You create ‘Reducers’ which are predefined functions as to what will happen to the state. These are exported and create a unified syntax for dealing with changes to global state.

Redux is useful in applications that require lots of global state variables. It stores all of these variables in a single place (which can be local storage for persistent data), and provides a unified syntax for managing state changes which can cut down on cognitive load.

1. **Are you familiar with useEffect React Hook? When would you use it? What are some disadvantages of its overuse?**

UseEffect is a React hook which allows you to ‘watch’ a variable. When that variable changes, the useEffect hook triggers the function written inside it.

It is often used to trigger data fetching functions, for instance on a user login page. When the username and password is submitted, the data can be sanitised and then stored as state. An instance of useEffect can be ‘watching’ this state variable, and when it updates, it triggers a function written inside the hook that attempts an api call to retrieve the user's information.

Disadvantages of it’s overuse are that it can become very complex to untangle where data is coming from if a useEffect hook has many dependencies or the function inside is triggering other useEffect functions.

It can also be the cause of frequent updates and re-renders of the application which can cause performance bottlenecks if misconfigured.