1. In design Heuristics, what does the term “advantages of Matching between system and the real world” mean? What are the advantages?

Meaning

“Match between system and the real world" means that a system's design should resemble real-world concepts familiar to users. For example, a digital calendar should look and feel like a physical one or a bin would indicate that the user will be throwing something away/ deleting an item.

Advantages

- Familiar designs reduce user errors resulting in fewer mistakes.
- Using real-world conventions makes systems more instinctive and easier to grasp.
- Users can work faster with systems that feel familiar.
- Users prefer designs that align with real-world logic.
- It's easier to remember how to use systems that mirror real-world concepts.
- This approach promotes a predictable user experience.

In short, systems that align with real-world logic offer a smoother and more user-friendly experience.

2. What do you understand by “Single source of truth”? and how does it relate to redux? What are the advantages?

   Single Source Of Truth (SSOT) is a data management concept where a central hub of information is accessible to all those on the project. The hub contains information that is reliable and up-to-date. For example, in Github, multiple developers may be working on a project where multiple versions are being created. However, there will always be an SSOT of the main branch which holds a version of the code that is stable and reliable at that point in time.

   Redux illustrates the SSOT concept. In Redux, it achieves this by storing all the data in one place called the ‘store’. State changes are created through actions, which are then processed by reducers which decide how to update the data in the store. Different parts of the application can effectively look to this state, ensuring the display of the latest data.

   Advantages

   - With just one source, there's no mixup or difference in data, so every part of the app shows the same information.

   - With Redux, if you have the same starting point and do the same thing, you'll always end up with the same result. This makes it easier to see what's going on and fix issues.

   - Having one main place for data helps manage updates happening at the same time, so things don't get mixed up.

   - With one main source, data doesn't get out of date or repeated.

   - It's easier to keep everything in one place, which means it's simpler to test.

   So, using the SSOT idea in Redux makes apps more organised and easy to handle, especially when they get big.

3. What is the difference between a stateless component and a stateful component in React?

   In React, components can be organised into stateless and stateful based on their ability to maintain state and their functionality.

   Stateless Components
   Simple components don't hold their own data. They acquire information through props and create elements to present to the user. Without their own data, they just show the information that is provided. Even when given the same input, they'll always display it the same way. These components are ideal for displaying content without the need for complex processes. These simple components don't have special rules, but with new tools like useEffect, simple components can do similar tasks if needed. They're also usually shorter and easier.

   For example:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

   Stateful Components
   Stateful components handle their own data and can trigger extra actions, like changing a variable or enabling/disabling a button on the user interface. They can update what they show as time goes on, managing their own specific information. We can use stateful components when parts of the app need to react to user actions, keep track of data, or work with lifecycle processes. Stateful components, in the form of classes, include lifecycle methods such as componentDidMount, componentDidUpdate, and componentWillUnmount. These methods let you do specific tasks at certain points in the component's life.

   For example:

```
class Welcome extends React.Component {
  constructor(props) {
    super(props);
    this.state = { name: 'React' };
  }
  render() {
    return <h1>Hello, {this.state.name}</h1>;
  }
}
```

4. List out the advantages and disadvantages of exploratory testing (used in Agile) and scripted testing.

Both exploratory and scripted testing are important testing methods, each with its own set of advantages and disadvantages.

Exploratory Testing

Advantages

- Flexibility: Testers aren't bound to a strict plan; they can adjust their testing strategies based on what they observe, giving them the freedom to explore.

- Discovery of New Issues: When approaching the programme without limiting themselves to predetermined tests, testers can often uncover problems that might not have been anticipated.

- Immediate Start: There's no need to wait for detailed test cases to be written. Testers can dive right in.

- Mimics Real User Behaviour: Testers often approach features in a way that's similar to end-users, highlighting potential real-world issues.

- Active Engagement: Testers are encouraged to critically think and engage with the application.

Disadvantages

- Recreating Issues: If testers find a bug, it might be harder to recreate it without detailed steps.

- Missed Tests: Without a plan, some parts of the application might not get tested.

- Tester’s Knowledge and Skills: The success of exploratory testing relies heavily on the tester's experience and knowledge.

Scripted Testing

Advantages

- Reproducible: Since there are specific steps to follow, any identified bugs or issues can be easily reproduced, aiding developers in fixing them.

- Thorough Coverage: Detailed scripts ensure that identified areas or functions of the application are checked thoroughly.

- Consistency: Different testers can use the same script, ensuring consistency in how tests are performed.

- Planning: Clearly outlined steps make it straightforward to develop automated tests later on.

- Measurable Outcomes: With predefined tests, it's easier to measure results and track progress.

Disadvantages

- Inflexibility: Scripted tests might not adapt well to changes, and their strict nature can restrict the tester.

- Time-consuming: Writing detailed, effective test scripts can be time-consuming.

- Overlooking Issues: If an issue wasn't considered in the initial script, it might not be identified.

- Varied User Behaviours: The set steps may not always mirror the variety of ways users might interact with a feature.

Both methods are important in a well-rounded testing plan. In an ideal world, teams will find a balance, utilising each method where it's most suitable in their Agile process.