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

In design heuristics, the term "matching between system and the real world" means that a system or product should work and look like something users are already familiar with in the real world. This helps users understand how to use it more easily because it follows familiar patterns and conventions.

Advantages of this approach include:

1. Ease of Learning:
    - Description: When a system reflects real-world conventions, users can leverage their existing knowledge and experiences to learn how to use the system more quickly.
    - Example: A software application that uses familiar icons (like a trash can for deleting files) helps users understand its functions.
2. Intuitive Interaction:
    - Description: If the system follows real-world patterns, users can predict how it will behave and interact with it in ways that feel natural and expected.
    - Example: A digital calendar that mimics a physical calendar layout helps users intuitively understand how to schedule and view events.
3. Reduced Errors:
    - Description: When users can rely on their real-world knowledge to interact with a system, they are less likely to make mistakes. This is because the system's design aligns with their expectations and experiences.
    - Example: A form with clearly labeled fields that match real-world forms reduces the chances of users entering information incorrectly.
4. Increased Comfort:
    - Description: Users feel more confident and comfortable when the system behaves in ways they are already familiar with, leading to a smoother and less frustrating experience.
    - Example: An e-commerce site that follows the common shopping cart model and checkout process helps users navigate the purchase process with ease.
5. Enhanced Efficiency:

- Description: Users can perform tasks more efficiently because the system supports familiar workflows and eliminates the need for learning new procedures.
- Example: A photo editing tool with standard tools and options similar to other widely-used photo editors allows users to work more quickly and effectively.

6. Consistency with User Expectations:

    - Description: Systems that match real-world norms meet users' expectations and provide a consistent experience, which enhances usability and satisfaction.
    - Example: A navigation app that uses familiar map symbols and controls aligns with users' prior experiences with physical maps and navigation devices.

Matching the system to the real world makes it more user-friendly and less confusing.

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" means having one central place where all important data is stored and managed. This ensures that everyone and everything in a system uses the same data, avoiding inconsistencies.

How it relates to Redux:

- Redux is a tool used in software development (especially with React) to manage the state of an application.
- Single source of truth in Redux means that all the application's state is stored in one place called the store. This store holds the entire state of the application, making it easy to access and update data consistently.

Advantages of a single source of truth with Redux:

1. Consistency: Since all data comes from one place, there's no confusion or duplication, making the application behave consistently.

2. Easier Debugging: When all data is centralised, it's simpler to track changes and find issues because you know exactly where to look.

3. Simplified State Management: Managing and updating data is easier because you only need to deal with one store rather than multiple sources.

4. Predictable State Changes: The data flow is more predictable and controlled, which makes it easier to understand and test how the application responds to changes.

5. Improved Collaboration: With a single source of truth, team members can work more effectively together because they are all using and updating the same data.

Using a single source of truth in Redux helps keep your application's data consistent, easier to manage, and simpler to debug.

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

In React, stateless components and stateful components are two types of components that handle data and behaviour differently.

Stateless Component

- Definition: A stateless component doesn't manage its own state. It simply receives data and functions through props and renders the UI based on that data.
- Characteristics:
  - Does not keep track of any data or state within itself.
  - Typically used for presenting data.
  - Reusable and simpler since it only depends on props.
- Example: A component that only displays a list of items passed as props.

Stateful Component

- Definition: A stateful component manages its own state. It can hold and modify its own data internally and can update the UI based on that state.
- Characteristics:
  - Maintains and manages its own state using React's `useState` hook or class-based state.
  - Can handle user interactions and update its own data.
  - More complex because it involves managing and updating internal state.

- Example: A component with a form that keeps track of user input and updates the UI as the user types.

Stateless Components: Focus on presenting data; they don't manage or store any data themselves. Example:

```
function Greeting({ message }) {
      return <h1>{message}</h1>;
}
function App() {
      return <Greeting message="Hello, World!" />;
      }
```

Stateful Components: Manage their own data and can update it based on user interactions or other events.

```
import React, { useState } from 'react';
function Counter() {
      const [count, setCount] = useState(0);
      const increment = () => {
      setCount(count + 1);
};
return (
      <div>
      <p>Current Count: {count}</p>
      <button onClick={increment}>Increment</button>
      </div>
);
}
function App() {
      return <Counter />;
}
```

Stateless components are simpler and focused on rendering, while stateful components handle more complex logic by maintaining and updating their own state.

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

Exploratory Testing

Advantages:

1. Flexibility: Testers can explore the application in any way they choose, which often uncovers unexpected issues.
2. Real-world Scenarios: Allows testers to use the application as real users would, potentially finding bugs that scripted tests might miss.
3. Quick Feedback: Testers can start testing immediately without waiting for detailed test cases to be created.
4. Creativity: Encourages testers to think creatively and explore different ways to interact with the application.

Disadvantages:

1. Less Coverage: Without a defined script, it might be hard to ensure that all areas of the application are tested.
2. Inconsistency: Different testers might test the same features in different ways, leading to inconsistent results.
3. Hard to Reproduce Bugs: Finding and reproducing issues can be more challenging without a structured approach.
4. Documentation: It may be harder to document what was tested and what was not, which can impact tracking and reporting.

Scripted Testing

Advantages:

1. Comprehensive Coverage: Test cases are planned in advance, so you can ensure all features are tested systematically.
2. Consistency: Every test is executed the same way, leading to consistent results and easier comparison.
3. Reproducibility: Easier to reproduce tests and verify if a bug is fixed because the steps are well-documented.
4. Clear Documentation: Detailed test cases provide a record of what was tested and how, which helps in tracking progress and reporting.

Disadvantages:

1. Less Flexibility: Testers follow predefined steps, which may not account for unexpected issues or new ways users might interact with the application.
2. Time-Consuming: Creating and maintaining detailed test scripts can be time-consuming and may slow down testing.
3. Potential for Missing Bugs: Focused on specific test cases, which might overlook unexpected issues or edge cases.
4. Less Adaptable: Changes in the application may require frequent updates to test cases, which can be burdensome.

Exploratory Testing is great for flexibility and finding unexpected issues but may lack coverage and consistency, when Scripted Testing ensures thorough testing and consistent results but can be rigid and time-consuming.