Questions and Answers

1. What's a closure? Where in the code is there a closure?

A closure in JavaScript is created when a function remembers its lexical scope even when it is executed outside that scope. Closures allow inner functions to access variables from an outer function even after the outer function has finished executing.

Where in the code is there a closure?
In the provided React code, closures occur in multiple places:

1. State updater with `useState` hook:
```
<button
 className="btn btn-success"
 onClick={() => setShowAddForm((prev) => !prev)}
>
 {showAddForm ? 'Close Add Joke' : 'Add Joke'}
</button>
```

- The `setShowAddForm` function closes over the `prev` state value.

2. Arrow functions in `map` function:
```
currentJokes.map((joke) => (
 <JokeCard
  key={joke.id}
  joke={joke}
  onLike={() => updateLikes(joke.id, 'like')}
  onDislike={() => updateLikes(joke.id, 'dislike')}
 />
));
```

- The arrow functions close over the `joke.id` value for each iteration.

2. Which are the potential side-effects in any function? Could you point out any of these cases in your code? Are they expected? Can they be avoided?

Potential Side-Effects in Functions:
Side-effects occur when a function interacts with the external world or modifies state.

Common examples include:
1. Modifying external state or variables.
2. Performing I/O operations (e.g., API calls, LocalStorage).
3. Mutating function parameters.
4. Non-deterministic operations (e.g., `Date.now()`, `Math.random()`).

Where are side-effects in the code?

1. Fetching and caching data:

```
const loadJokes = async () => {
 const cachedJokes = localStorage.getItem('cachedJokes');
 if (cachedJokes) {
  const jokesData = JSON.parse(cachedJokes);
  setJokes(jokesData);
 } else {
  const data = await fetchJokes();
  setJokes(data);
  localStorage.setItem('cachedJokes', JSON.stringify(data));
 }
};
```

- ide-effect: API calls and LocalStorage updates.
- Expected: Yes, these are necessary for fetching and caching jokes.
- Can they be avoided?** Not entirely, but they are controlled using `useEffect`.

2. Updating likes and dislikes:

```
const updateLikes = (id, type) => {
 const updatedJokes = jokes.map((joke) => {
  if (joke.id === id) {
   return {
    ...joke,
    likes: type === 'like' ? joke.likes + 1 : joke.likes,
   };
  }
  return joke;
 });
 setJokes(updatedJokes);
 localStorage.setItem('cachedJokes', JSON.stringify(updatedJokes));
};
```

- Side-effect: State updates and LocalStorage persistence.
- Expected: Yes, as they track user interactions.

How to avoid/control side-effects:
1. Use `useEffect` for API calls and cache updates.
2. Keep functions pure when possible.
3. Encapsulate side-effects in isolated functions.