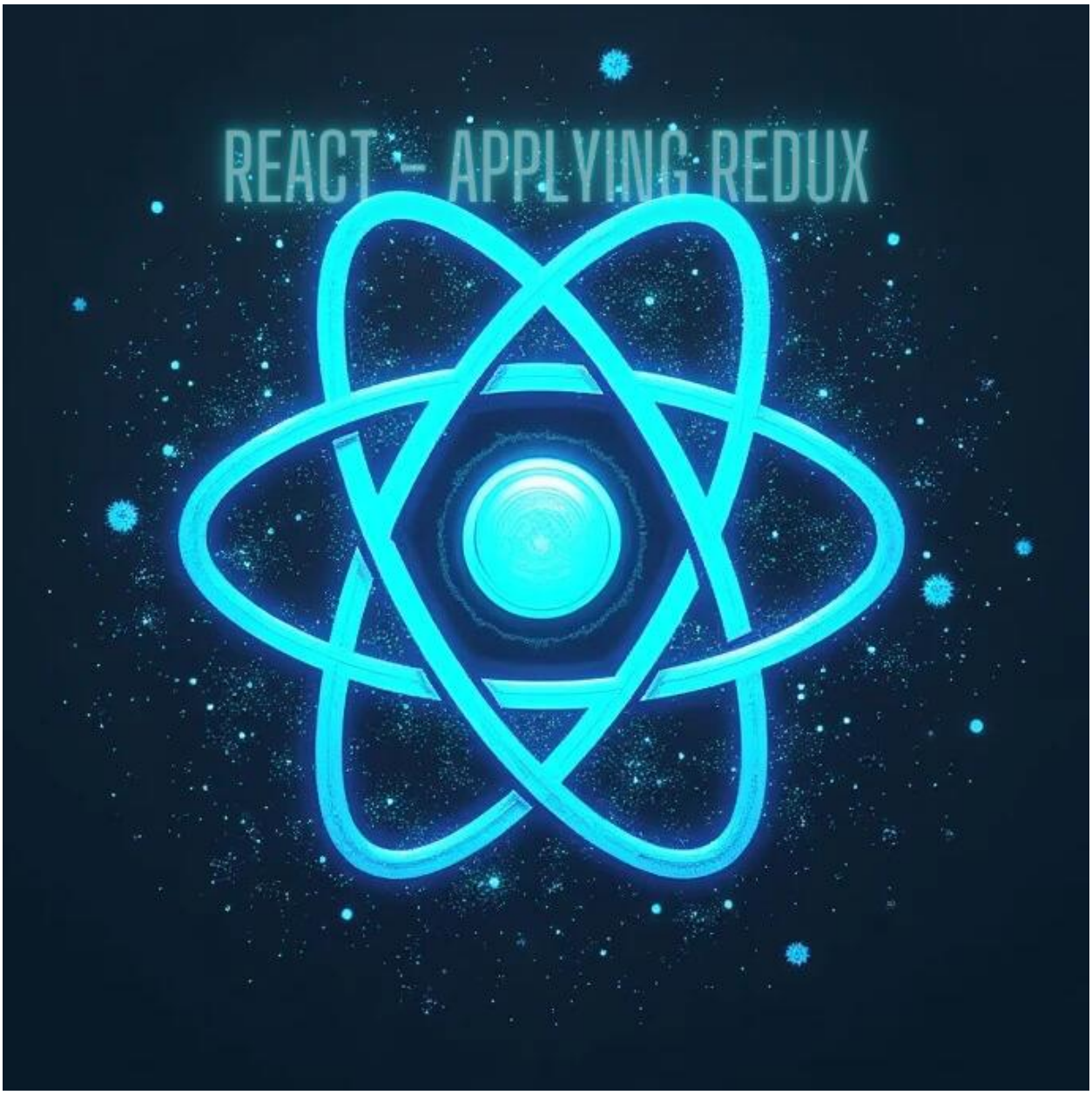

Redux, Redux-Toolkit

Question 1: What is Redux, and why is it used in React applications? Explain the coreconcepts of actions, reducers, and the store.

Ans.

Redux is a state management library used in React applications to manage the application's global state in a predictable way. It helps :

1. **Centralized State**: Stores all app state in a single location (store).
2. **Predictability**: Changes to the state happen using pure functions called reducers.
3. **Easier Debugging**: Tools like Redux DevTools help track state changes over time.

It is used to avoid "prop drilling" and to efficiently share data between components.

In **Redux**, three core concepts—**actions**, **reducers**, and the **store**—work together to manage the application's state in a predictable way. Here's a simple explanation of each :

1. Actions :

- **What are actions ?**

- Actions are plain JavaScript objects that describe what happened in the application.
- They must have a type property (a string) that specifies the kind of action being performed.
- Optionally, they can include additional data (called a **payload**) to provide details about the action.

❖ Example :

```
const incrementAction = {
      type: 'INCREMENT',
      payload: { amount: 1 },
};
```

❖ Why use actions ?

- Actions define what the user or app wants to do, like updating the state or fetching data.

2. Reducers :

❖ What are reducers ?

- A reducer is a pure function that takes the current state and an action as inputs and returns a new state.
- It determines how the state should be updated based on the type of action received.

❖ Key principles of reducers:

1. They must not mutate the state directly (always return a new state).
2. They must be pure functions (no side effects like API calls inside reducers).

❖ Example :

```
const counterReducer = (state = { count: 0 }, action) => {
      switch (action.type) {
      case 'INCREMENT':
            return { ...state, count: state.count + action.payload.amount
};
      case 'DECREMENT':
            return { ...state, count: state.count - action.payload.amount
};
      default:
            return state;
      }
};
```

❖ Why use reducers?

➢ Reducers handle the logic for how actions transform the application's state.

3. Store

❖ What is the store ?

➢ The store is a centralized object that holds the application's entire state.

➢ It is created using the createStore method from Redux and connects actions and reducers.

❖ Responsibilities of the store :

1. Holds the state.
2. Provides getState() to access the state.
3. Provides dispatch(action) to send actions to the reducer.

❖ Example :

```
import { createStore } from 'redux';

const store = createStore(counterReducer);

console.log(store.getState()); // { count: 0 }
store.dispatch({ type: 'INCREMENT', payload: { amount: 1 } });
console.log(store.getState()); // { count: 1 }
```

❖ Why use the store ?

- ➢ It acts as a single source of truth for the state, making it easier to manage complex state transitions.