**How would you handle asynchronous actions within a state machine? Provide examples of handling asynchronous actions (like API calls) within a state machine using JavaScript.**

To handle asynchronous actions (e.g., API calls), you can introduce Promise-based state transitions.

class AsyncTrafficLightStateMachine {

constructor() {

this.states = {

green: {

next: "yellow",

duration: 5000

},

yellow: {

next: "red",

duration: 2000

},

red: {

next: "green",

duration: 8000

}

};

this.currentState = "red";

}

async makeAPIRequest(state) {

console.log(`Making API call for state: ${state}...`);

// Simulate an asynchronous API call (e.g., logging state transition)

return new Promise((resolve) => {

setTimeout(() => {

console.log(`API call for state "${state}" completed.`);

resolve();

}, 1000); // Simulated API delay of 1 second

});

}

async transitionToNextState() {

console.log(`Transitioning from ${this.currentState}...`);

const state = this.states[this.currentState];

// Handle asynchronous action (API call)

await this.makeAPIRequest(this.currentState);

setTimeout(async () => {

this.currentState = state.next;

console.log(`New state: ${this.currentState}`);

await this.transitionToNextState(); // Continue with the next state

}, state.duration);

}

start() {

console.log("Starting traffic light with async API calls...");

this.transitionToNextState();

}

}

// Initialize and start the async traffic light

const asyncTrafficLight = new AsyncTrafficLightStateMachine();

asyncTrafficLight.start();

Uses setTimeout to simulate delays for state transitions.

**Asynchronous State Machine**:

* Introduces an async makeAPIRequest function to simulate API calls.
* Uses await within the state transition logic to ensure the asynchronous action completes before moving to the next state.