Rules and Guidelines

● To attempt this test, you have 1.5 Hours.
● Candidates must complete 2 out of the 3 provided questions within the allocated timeframe. Attempting all 3 questions will be considered as a performance multiplier in the final evaluation.
● You can refer to the official documentation of ReactJS or use libraries from NPM.
● Looking for solutions or taking help from AI tools is NOT allowed.

Q1) Create a single-page to-do list app using the following mock APIs.

GET: https://60eedea7eb4c0a0017bf4685.mockapi.io/api/test/todo/
POST: https://60eedea7eb4c0a0017bf4685.mockapi.io/api/test/todo/

request body:

```
{
    "task": "This is my Task 1"
}
```

DELETE: https://60eedea7eb4c0a0017bf4685.mockapi.io/api/test/todo/:id

id: id of task you want to remove

Requirements:

1. Create an Atomic Folder Structure

2. Use Redux Toolkit with proper middleware setup

3. Data Management:

- Fetch and render tasks with pagination (10 items per page)
- Implement filtering by status, priority, and category
- Add search functionality with debouncing
- Enable sorting by due date, priority, and status

4. Task Properties:

- Description
- Priority (High, Medium, Low)
- Due Date

- Status (Pending, In Progress, Completed)
- Categories/Tags
- Subtasks

5. User Interactions:
- Add, edit, and delete tasks
- Bulk actions for multiple tasks
- Drag-and-drop reordering
- Undo/redo functionality
- Task completion animations

6. Performance & Technical:
- Implement request caching
- Add retry mechanism for failed requests
- Optimize re-renders
- Add loading states and skeleton screens
- Implement offline functionality

7. Error Handling:
- Implement comprehensive error boundaries
- Add user-friendly error messages
- Handle network errors gracefully
- Add validation for all user inputs

8. Testing & Documentation:
- Write unit tests for reducers and components
- Add E2E tests for critical paths
- Document component architecture
- Include performance benchmarks

2) Create an E-commerce Product Management System App

Background:

You are working on an e-commerce platform that needs to manage products, cart functionality, and filtering capabilities.

The base code uses Redux Toolkit and the Fake Store API (https://fakestoreapi.com/).

Tasks:

1. State Management

- Complete the productSlice by adding the extraReducers for handling async operations
- Implement proper loading and error states
- Add a selector to calculate the cart total
-

2. Cart Features

- Implement a quantity limit (max 2 items per product)
- Add three products in cart
- Create a selector for filtered products based on searchTerm and category

3. Async Operations

- Add error handling for network failures
- Implement a retry mechanism for failed API calls (maximum 3 retries)
- Add a cache system for previously fetched products

4. Advanced Features

- Create a middleware to sync cart with localStorage

5. Testing Requirements

- Redux reducers
- Async operations

- Integration tests:
  * API interactions
  * Cart operations
  * State management
- E2E tests with Cypress:
  * Critical user paths
  * Edge cases
- Performance testing:
  * Load testing
  * Bundle size

Requirements:

1. Create Atomic Folder Structure
2. Use Redux Toolkit's
3. Implement proper error handling
4. Write clean, maintainable code with comments
5. Consider performance optimizations
6. Use React Bootstrap or Tailwind for responsive UI

Q3) Create an app for a dynamic form builder

Background:

This form should handle a variety of field types (e.g., text input, dropdown, radio buttons, checkboxes), where each field's configuration is dynamically passed as props based on a schema defined by an API.

1. Dynamic Field Generation & Management

- Multi-step form capability
- Nested form fields (forms within forms)
- Field dependencies and conditional rendering
- Dynamic field ordering and grouping
- Field templating system
- Custom field layouts (grid, columns, responsive)

2. Advanced Validation

- Async validation with API calls
- Cross-field validation
- Custom validation rules
- Real-time validation
- Validation groups
- Error boundary handling
- Form-level validation

3. Dynamic Field Types & Props

- Complex field types (rich text, file upload, date range)

- Custom field renderers
- Field composition patterns
- Nested field arrays
- Dynamic field masking
- Field permissions system

4. State Management

- Form versioning
- Undo/Redo functionality
- Partial form saves
- Form state persistence
- Multi-form state coordination
- Form session management
- State migration strategies

5. Advanced Features

- Form templating system
- Form analytics tracking
- A/B testing capability
- Form versioning control
- Multi-language support
- Theme customization
- Form preview mode

6. Performance Optimization

- Field lazy loading
- Form splitting and code splitting
- Render optimization
- Memory management for large forms
- Cache strategies
- Performance metrics tracking

7. Enhanced Accessibility

- Screen reader optimization
- Keyboard navigation
- ARIA attributes management
- Color contrast compliance
- Motion sensitivity considerations
- Accessibility testing requirements

8. Advanced Submit Handling

- Multi-step submission
- Partial form submission
- Offline support
- Retry mechanisms
- Progress tracking
- Submit queue management

9. Data Management

- Data transformation pipeline

- Data normalization
- Import/Export functionality
- Data validation rules
- Data persistence strategies
- Change tracking

10. Testing Requirements

- Unit test coverage
- Integration testing
- Performance testing
- Accessibility testing
- Visual regression testing
- Load testing for large forms

11. Performance Metrics

- Time to Interactive
- Form render performance
- Memory usage optimization
- Network payload optimization
- State update performance