Frontend Developer Assignment

Project Overview

Build a **Custom Data Grid Component** with advanced features similar to MUI DataGrid, but implemented entirely from scratch without using any external data grid libraries.

Tech Stack Requirements

- **Framework**: Next.js 14+ (App Router)
- **Language**: TypeScript (.tsx files)
- **Styling**: Tailwind CSS (responsive design)
- **State Management**: React Context API + useReducer
- **Animations**: CSS animations/transitions + Framer Motion
- **API Integration**: Custom dummy API or JSONPlaceholder integration

Core Features to Implement

1. Data Grid Core Functionality

- **Dynamic Column Rendering**: Render columns based on data structure
- **Row Virtualization**: Handle large datasets efficiently (1000+ rows)
- **Sorting**: Multi-column sorting with visual indicators
- **Filtering**: Column-specific filters (text, number, date, select)
- **Pagination**: Client-side and server-side pagination
- **Search**: Global search across all columns

2. Column Management Features

- **Show/Hide Columns**: Toggle column visibility
- **Column Reordering**: Drag and drop columns
- **Column Pinning**: Pin columns to left or right
- **Column Resizing**: Drag to resize column widths
- **Column Freezing**: Freeze columns while scrolling
- **Column Grouping**: Group related columns with headers

3. Advanced Grid Features

- **Row Selection**: Single and multi-row selection with checkboxes
- **Inline Editing**: Edit cells directly in the grid
- **Custom Cell Renderers**: Different cell types (text, number, date, actions)
- **Row Actions**: Edit, delete, view buttons per row
- **Bulk Actions**: Operations on multiple selected rows

- **Export Functionality**: Export to CSV/JSON
- **Density Control**: Compact, standard, comfortable row heights

4. State Management Requirements

- **Context API**: Global state for grid configuration
- **useReducer**: Complex state updates for grid operations
- **Custom Hooks**: Reusable logic for grid operations
- **Persistence**: Save user preferences (column order, visibility, etc.)
- **Optimistic Updates**: Immediate UI updates for better UX

5. UI/UX Requirements

- **Responsive Design**: Mobile-first approach
- **Dark/Light Theme**: Toggle between themes
- **Loading States**: Skeleton loaders and spinners
- **Error Handling**: Graceful error states with retry options
- **Accessibility**: ARIA labels, keyboard navigation, screen reader support
- **Touch Support**: Mobile gestures for scrolling and selection

6. Animation Requirements

- **Smooth Transitions**: Column reordering, resizing, show/hide
- **Micro-interactions**: Hover effects, button states
- **Loading Animations**: Data fetching indicators
- **Staggered Animations**: Row appearance animations
- **Gesture Feedback**: Visual feedback for touch interactions

Technical Specifications

API Integration

Create or integrate with a dummy API that provides:

typescript

```typescript
interface User {
  id: number;
  name: string;
  email: string;
  role: string;
  department: string;
  salary: number;
  joinDate: string;
  status: 'active' | 'inactive';
  avatar?: string;
}

interface ApiResponse<T> {
  data: T[];
  total: number;
  page: number;
  pageSize: number;
  totalPages: number;
}
```

State Management Structure

typescript

```typescript
interface GridState {
  data: any[];
  columns: Column[];
  visibleColumns: string[];
  pinnedColumns: { left: string[]; right: string[] };
  sortModel: SortModel[];
  filterModel: FilterModel;
  selectedRows: Set<string>;
  pagination: PaginationState;
  loading: boolean;
  error: string | null;
}
```

Performance Requirements

- **Virtual Scrolling**: Handle 10,000+ rows smoothly
- **Debounced Search**: 300ms debounce for search input
- **Memoization**: Prevent unnecessary re-renders
- **Lazy Loading**: Load data as needed
- **Caching**: Cache API responses appropriately

File Structure

```
src/
├── components/
│   ├── DataGrid/
│   │   ├── DataGrid.tsx
│   │   ├── DataGridHeader.tsx
│   │   ├── DataGridRow.tsx
│   │   ├── DataGridCell.tsx
│   │   ├── ColumnManager.tsx
│   │   ├── FilterPanel.tsx
│   │   └── Pagination.tsx
│   ├── ui/
│   │   ├── Button.tsx
│   │   ├── Input.tsx
│   │   ├── Select.tsx
│   │   └── Modal.tsx
├── contexts/
│   ├── DataGridContext.tsx
│   └── ThemeContext.tsx
├── hooks/
│   ├── useDataGrid.ts
│   ├── useVirtualScroll.ts
│   ├── useLocalStorage.ts
│   └── useApi.ts
├── types/
│   ├── grid.types.ts
│   └── api.types.ts
├── utils/
│   ├── gridHelpers.ts
│   ├── exportUtils.ts
│   └── validators.ts
└── app/
    ├── page.tsx
    └── layout.tsx
```

Evaluation Criteria

Code Quality (25%)

- **TypeScript Usage**: Proper typing, interfaces, generics
- **Component Architecture**: Reusable, maintainable components
- **Custom Hooks**: Clean separation of logic
- **Error Handling**: Comprehensive error boundaries
- **Code Organization**: Clear file structure and naming

State Management (25%)

- **Context API Implementation**: Proper context setup and usage
- **useReducer Logic**: Complex state updates handled correctly
- **Performance Optimization**: Minimal re-renders
- **State Persistence**: User preferences saved locally
- **Data Flow**: Clear and predictable state flow

UI/UX Implementation (25%)

- **Responsive Design**: Works on all screen sizes
- **Animation Quality**: Smooth, purposeful animations
- **Accessibility**: WCAG compliance
- **User Experience**: Intuitive interactions
- **Visual Design**: Clean, modern interface

Technical Implementation (25%)

- **Performance**: Handles large datasets efficiently
- **API Integration**: Proper data fetching and caching
- **Custom Implementation**: No external grid libraries used
- **Browser Compatibility**: Works across modern browsers
- **Mobile Experience**: Touch-friendly interactions

Deliverables

1. **Complete Next.js Application**
    - All source code with proper TypeScript
    - Comprehensive README with setup instructions
    - Environment configuration files
2. **Documentation**
    - Component API documentation
    - Usage examples
    - Performance optimization notes
    - Known limitations and future improvements
3. **Demo Data**
    - Mock API implementation or integration
    - Sample dataset (500+ records)
    - Different data types for testing
4. **Testing**
    - Unit tests for core components
    - Integration tests for state management
    - E2E tests for critical user flows

Bonus Features (Optional)

- **Keyboard Shortcuts**: Power user features
- **Custom Themes**: Multiple theme options
- **Advanced Filtering**: Date ranges, multi-select filters
- **Real-time Updates**: WebSocket integration
- **Print Support**: Printer-friendly layouts
- **Internationalization**: Multi-language support

Timeline

- **Phase 1**: Core grid functionality (40% - 3 days)
- **Phase 2**: Advanced features (40% - 3 days)
- **Phase 3**: Polish and optimization (20% - 1 day)

Success Metrics

- Grid loads 1000+ rows in under 2 seconds
- Smooth 60fps animations
- Mobile responsive on all devices
- Accessible to screen readers
- Clean, maintainable codebase

Submission Guidelines

1. Create a GitHub repository
2. Include comprehensive README
3. Deploy to Vercel/Netlify
4. Record a 5-minute demo video
5. Submit repository link and deployed URL