📋 Guida Completa al Componente Kanban

🎯 Cos'è il Componente Kanban

Il componente Kanban è una board interattiva drag & drop che simula workflow tipo Trello/Jira. Permette di spostare entità tra colonne rappresentanti diversi stati del processo.

Caratteristiche principali:

- Drag & drop fluido con feedback visivo
- Optimistic updates con rollback automatico
- Conferma utente per cambi di stato critici
- Architettura generica riutilizzabile
- Integrazione con Material-React-Table

🏗️ Struttura e Tipi

Tipi Principali

typescript

```typescript
// Singolo elemento trascinabile
type KanbanItem<E> = {
  entity: E;                // Entità dati (task, ticket, etc.)
  cardContent: ReactNode;   // Contenuto JSX della card
}

// Colonna della board
type KanbanColumn<E, S> = {
  id: string;               // ID univoco colonna
  title: string;            // Titolo visualizzato
  color: string;            // Colore bordo superiore
  state: S;                 // Stato logico associato
  items: KanbanItem<E>[];   // Array elementi
}

// Board completa
type KanbanBoard<E, S> = {
  columns: Record<string, KanbanColumn<E, S>>;
}
```

Parametri di Configurazione

typescript

```typescript
interface KanbanParams<E, S> {
  flowStates?: S[];                    // Stati del workflow
  rowModel?: MRT_RowModel<E>;          // Dati tabella
  equalStatus?: (entity: E, state: S) => boolean;  // Comparatore stato
  getEntityCard?: (entity: E) => ReactNode;        // Renderer card
  getStateParams?: (state: S) => {                 // Parametri colonna
    colId: string;
    title: string;
    color: string;
  };
  commitState?: (                      // Salvataggio stato
    elementId: any,
    state: S,
    successCallback: (res: GenericResponse) => void,
    errorCallback: (error: ErrorResponse) => void
  ) => void;
  doRefech?: () => void;               // Refresh dati
}
```

🔄 Gestione Drag & Drop

Flusso onDragEnd

typescript

```
const onDragEnd = (result: DropResult) => {
  // 1. VALIDAZIONI
  if (!destination) return;
  if (source === destination) return;
  if (!params?.commitState) return;

  // 2. OPTIMISTIC UPDATE
  // Aggiorna immediatamente l'UI locale
  const item = sourceCol.items[source.index];
  // Rimuovi dalla sorgente
  newSourceItems.splice(source.index, 1);
  // Aggiungi alla destinazione
  newDestItems.splice(destination.index, 0, item);
  setData(newBoardState);

  // 3. CONFERMA UTENTE (solo se cambia stato)
  if (sourceCol.id !== destCol.id) {
    setWaitingConfirm(true);
    setConfirmDialog(/* ... */);
  }
```

```
    // 4. COMMIT PERSISTENTE
    params.commitState(
      item.entity.id,
      destCol.state,
      successCallback,  // → doRefech() + cleanup
      errorCallback     // → doRefech() + rollback + error
    );
  };
```

Pattern Implementati

- **Optimistic Update**: UI aggiornata immediatamente
- **Rollback**: in caso di errore, ripristina stato precedente
- **Conferma Utente**: per cambi di stato importanti
- **Loading States**: feedback visivo durante operazioni async

---

🎨 Struttura Visiva

Layout Colonne

typescript

```jsx
// Container principale con scroll orizzontale
<Box sx={{ display: "flex", overflowX: "auto", gap: 2 }}>

  // Ogni colonna: Paper con bordo colorato
  <Paper sx={{
    minWidth: 300, maxWidth: 300,    // Larghezza fissa
    height: "80vh",                   // Altezza viewport
    borderTopColor: col.color,        // Bordo identificativo
    borderTopWidth: theme.spacing(3)
  }}>

    // Header colonna
    <Typography variant="h6">{col.title}</Typography>

    // Container card scrollabile
    <Box sx={{ overflowY: "auto", flexGrow: 1 }}>
      {/* Card draggabili */}
    </Box>
  </Paper>
</Box>
```

Styling Card

typescript

```typescript
<Card sx={{
  mb: 1,
  minHeight: 120,              // Altezza minima consistente
  display: "flex",
  alignItems: "center",
  justifyContent: "center"
}}>
  <CardContent>{item.cardContent}</CardContent>
</Card>
```

⚙️ Configurazione e Utilizzo

Setup Base

typescript

```
// 1. Definire gli stati del workflow
enum TaskStatus { TODO = 'todo', IN_PROGRESS = 'progress', DONE = 'done' }

// 2. Configurare i parametri
const kanbanParams: KanbanParams<Task, TaskStatus> = {
  flowStates: [TaskStatus.TODO, TaskStatus.IN_PROGRESS, TaskStatus.DONE],
  rowModel: tableInstance.getRowModel(),

  equalStatus: (task, status) => task.status === status,

  getStateParams: (status) => ({
    colId: status,
    title: statusLabels[status],
    color: statusColors[status]
  }),

  getEntityCard: (task) => (
    <TaskCard task={task} />
  ),

  commitState: async (taskId, newStatus, onSuccess, onError) => {
```

```
    try {
      await updateTaskStatus(taskId, newStatus);
      onSuccess({ message: 'Status updated' });
    } catch (error) {
      onError({ error: 'Update failed' });
    }
  },

  doRefech: () => queryClient.invalidateQueries(['tasks'])
};

// 3. Renderizzare il componente
<Kanban {...kanbanParams} />
```

Integrazione con React Query

```typescript
const { data: tasks, refetch } = useQuery(['tasks'], fetchTasks);

const kanbanParams = {
  // ... altri parametri
  rowModel: {
    rows: tasks?.map(task => ({ original: task })) || []
  },
  doRefech: refetch
};
```

🔧 Customizzazioni Avanzate

Card Personalizzate

typescript

```typescript
const getEntityCard = (task: Task) => (
  <Box>
    <Typography variant="h6">{task.title}</Typography>
    <Typography variant="body2" color="text.secondary">
      {task.description}
    </Typography>
    <Chip
      label={task.priority}
      color={priorityColors[task.priority]}
      size="small"
    />
    <Avatar src={task.assignee.avatar} />
  </Box>
);
```

Stati Complessi

typescript

```
const getStateParams = (status: TaskStatus) => {
  const configs = {
    [TaskStatus.TODO]: {
      colId: 'todo',
      title: '📋 Da Fare',
      color: '#f44336'
    },
    [TaskStatus.IN_PROGRESS]: {
      colId: 'progress',
      title: '⚡ In Corso',
      color: '#ff9800'
    },
    [TaskStatus.REVIEW]: {
      colId: 'review',
      title: '👀 In Revisione',
      color: '#2196f3'
    },
    [TaskStatus.DONE]: {
      colId: 'done',
      title: '✅ Completato',
      color: '#4caf50'
```

```
    }
  };
  return configs[status];
};
```

⚠️ Limitazioni e Best Practices

Limitazioni

- **Performance**: Ricostruisce board ad ogni cambio parametri
- **Memory**: Mantiene stato locale per optimistic updates
- **Browser Storage**: Non supporta localStorage/sessionStorage in artifacts

Best Practices

typescript

```
// ✅ Memoizza le funzioni callback
const getEntityCard = useCallback((entity) => (
  <EntityCard entity={entity} />
), []);

// ✅ Usa React.memo per card complesse
const TaskCard = React.memo(({ task }) => (
  <ComplexTaskDisplay task={task} />
));

// ✅ Gestisci stati di loading
const commitState = async (id, state, onSuccess, onError) => {
  setLoading(true);
  try {
    const result = await api.updateStatus(id, state);
    onSuccess(result);
  } catch (error) {
    onError(error);
  } finally {
    setLoading(false);
  }
```

```
};

// ✅ Implementa retry logic
const commitWithRetry = async (id, state, onSuccess, onError, retries = 3) => {
  for (let i = 0; i < retries; i++) {
    try {
      await commitState(id, state, onSuccess, onError);
      return;
    } catch (error) {
      if (i === retries - 1) onError(error);
      await new Promise(resolve => setTimeout(resolve, 1000 * Math.pow(2, i)));
    }
  }
};
```

🔄 Alternative Librerie DnD

Libreria	Pro	Contro	Uso Consigliato
@hello-pangea/dnd	Semplice, performante, buona documentazione	Limitato a liste/griglie	✅ Kanban boards
dnd-kit	Moderno, accessibile, leggero, TypeScript	Curva apprendimento	✅ App moderne
react-beautiful-dnd	Maturo, stabile	Deprecato, no nuove features	❌ Evitare
react-dnd	Molto flessibile, potente	Complesso, verbose	⚠️ Solo per casi complessi

Migrazione a dnd-kit

typescript

```typescript
// hello-pangea/dnd
<DragDropContext onDragEnd={onDragEnd}>
  <Droppable droppableId="column">
    <Draggable draggableId="item" index={0}>

// dnd-kit equivalente
<DndContext onDragEnd={onDragEnd}>
  <Droppable id="column">
    <Draggable id="item">
```

🪲 Troubleshooting

Problemi Comuni

Card non si muove

```typescript
// ❌ Problema: manca key univoca
{items.map((item, index) => (
  <Draggable key={index} // ❌ index instabile

// ✅ Soluzione: usa ID entità
{items.map((item, index) => (
  <Draggable key={item.entity.id} // ✅ ID stabile
```

Stato non si aggiorna

```typescript
// ❌ Problema: commitState non chiama callback
commitState(id, state); // Mancano callback

// ✅ Soluzione: implementa callback
commitState(id, state, onSuccess, onError);
```

Performance degradata

```typescript
// ❌ Problema: funzioni inline
getEntityCard={(entity) => <Card />} // Ricreata ogni render

// ✅ Soluzione: useCallback
const getEntityCard = useCallback((entity) => <Card />, []);
```

📚 Risorse Aggiuntive

- **Demo Live**: Esempi interattivi nel repository
- **TypeScript Types**: Definizioni complete per IntelliSense
- **Testing**: Unit tests con React Testing Library
- **Storybook**: Componenti isolati per sviluppo
- **Performance**: Bundle analyzer per ottimizzazioni

Documentazione aggiornata - Versione 2.0

■ Gestione Scroll e Avvisi `@hello-pangea/dnd`

✔ Scroll Presenti nel Componente Kanban

1■■ Scroll Orizzontale (contenitore colonne):
<Box sx={{ display: "flex", overflowX: "auto", ... }}>

- Permette di visualizzare tutte le colonne affiancate (layout a riga singola).
- Scroll container necessario per UX responsive.

2■■ Scroll Verticale (contenuto colonne) – opzionale:
<Box sx={{ overflowY: "auto", flexGrow: 1 }}

- Scrolla le card interne a ogni colonna.
- **Va evitato l'annidamento di scroll container non gestiti da `Droppable`.**

■ Avviso comune:
"Droppable: unsupported nested scroll container"

■ Soluzioni:

■ Best Practice (scroll orizzontale supportato):
- Il container scrollabile deve contenere `Droppable` nel DOM.
- Evitare overflowY nei genitori diretti di `<Droppable>` se non necessario.

■ Se il layout è corretto ma compare comunque l'avviso:
- Si può filtrare il warning in fase di sviluppo:

```tsx
useEffect(() => {
  const originalWarn = console.warn;
  console.warn = (...args) => {
    if (
      typeof args[0] === "string" &&
      args[0].includes("Droppable: unsupported nested scroll container")
    ) return;
    originalWarn.apply(console, args);
  };
  return () => {
    console.warn = originalWarn;
  };
}, []);
```

■ Conclusione:
- Uno scroll orizzontale nella Kanban è corretto e previsto.
- Scroll verticali aggiuntivi devono essere attentamente gestiti per evitare warning.