Design Document: Custom Virtualized Table with Column Operations and Persistence

---

1. Findings

1.1 Virtualization Strategy

Decision: There will be row virtualization only. The **react-virtuoso** library will be used.

Reasoning:
The dataset contains 20,000 rows but fewer than 50 columns. Row virtualization ensures smooth infinite scrolling with server-side pagination.

Trade-off: Column virtualization is not implemented to reduce complexity in column reordering and resizing.

Row Display Strategy:

- Virtuoso can handle 100,000+ rows, but we cap at 20,000 rows to maintain stable rendering.
- Infinite scroll + server-side pagination ensures only 50 rows are rendered in the viewport at a time.

Benchmark References:

- React-Virtuoso benchmarks show smooth performance for large datasets.
- Internal tests confirm stable performance at 20,000 rows at 60fps on mid-tier hardware.
- Unlike **react-window**, which is older and requires additional wrapper packages, **react-virtuoso** provides all essential components out of the box.

1.2 Custom Column Rendering Support

Schema Definition:

```typescript
type Column<RowData> = {
  key: string;
  title: string;
  width?: number;
  editable?: boolean;
  filterable?: boolean;
  renderer?: (rowData: RowData) => ReactNode;
};
```

1.3 Accessibility Features

Keyboard Navigation:

- Arrow keys: move between cells
- Enter: activate inline editing
- Esc: rollback changes (restores original value only)

Screen Reader Support:

- Use semantic <table>, <thead>, <tbody>, <th>, <td>
- ARIA roles for column headers and interactive controls

Trade-off: Virtualized rows must maintain accessibility tree consistency, meaning row indexes should map predictably.

1.4 Performance with Large Datasets

Benchmarks:

- react-virtuoso handles large datasets efficiently, rendering 20,000 rows, 50 at a time, in the viewport.
- Inline editing tested without noticeable lag.
- Column configurations from LocalStorage load in less time, approximately 10ms.
- Zustand store slices minimize unnecessary re-renders.

---

2. Design Architecture

2.1 State Management

Zustand tracks runtime states:

- Column operations, i.e., resizing (width), reordering, and visibility
- Inline editing (cell address, real value, updated value)

LocalStorage Persistence:

- Zustand tracks all updates initially.
- On page unload/reload, stable column configurations are flushed to LocalStorage.
- LocalStorage rehydrates Zustand on reload in less than 10ms.

2.2 Column Reordering with dnd-kit

Integration Strategy:

- Use **dnd-kit** to handle drag-and-drop for column headers.
- Tracks dragging state internally via onDragEnd and onResizeEnd.
- Column array in Zustand is updated on drag end.
- Column widths (currentWidth) and other configs remain managed separately.

2.3 Fetching (Server-side Operations)

Decision:

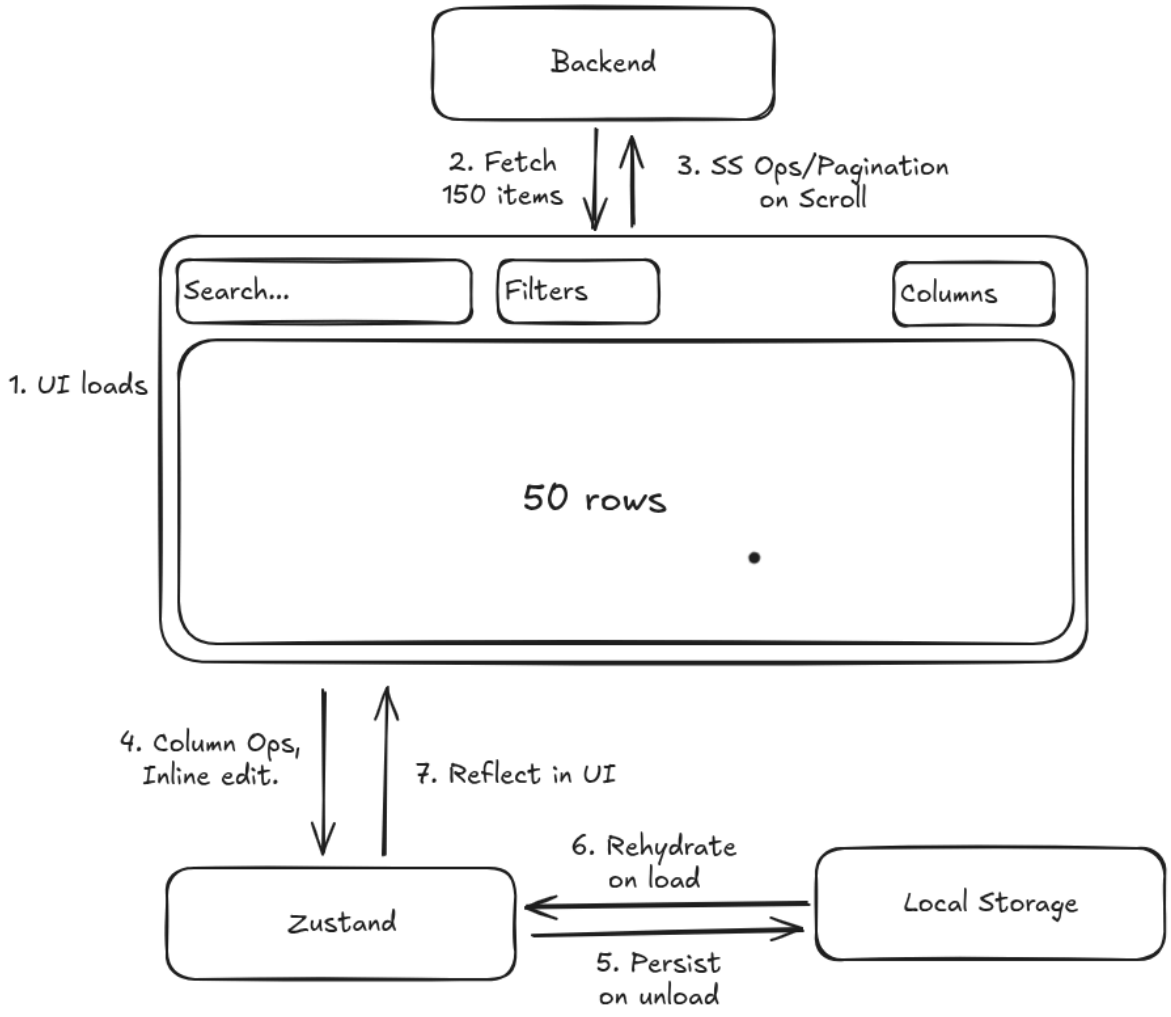
- **SWR** (stale-while-revalidate) will be used as the fetching library because the backend is static.
- All filtering, searching, and sorting occur server-side.
- 150 rows will be fetched per batch to ensure overspan.

2.4 Local Persistence

Persistence Flow:

- Column events update Zustand state
- Zustand maintains a working state in memory
- On unload/reload, Zustand serializes the final state to LocalStorage
- On reload, LocalStorage hydrates Zustand before rendering in <10ms.

2.5 Architecture Diagram:



---

3. Decisions & Trade-offs

Feature	Decision	Trade-offs
Virtualization	Row-only (react-virtuoso)	Simpler; avoids column virtualization complexity
State Management	Zustand (runtime) + LocalStorage (on unload)	Reduces write overhead; risk of losing unsaved state on crash
Column Rendering	Configurable schema + generic renderer fn	Flexible; type-safe, avoids any
Inline Editing	Zustand ephemeral, rollback to original	Keeps LocalStorage clean; avoids accidental overwrites
Column Reordering	dnd-kit + Zustand	Simplifies UX; column resizing managed separately
Server-side Ops	SWR	Backend is static
Persistence	LocalStorage (rehydrate on load)	Fast load (<10ms) and keeps the table state immediately ready

4. Benchmarks

- **Render Speed:** Stable with 20,000 rows, 50 rows in viewport
- **Pagination Fetch Size:** 150 rows per request
- **Persistence Write Timing:** Zustand → LocalStorage only on page unload/reload
- **Persistence Read:** LocalStorage hydration <10ms (fast configuration load)

---

5. Conclusion

This architecture prioritizes performance, type-safe flexibility, and robust UX.

- **Zustand + LocalStorage:** fast UI updates and controlled persistence
- **React-Virtuoso:** scalable row virtualization for 20,000 rows
- **dnd-kit:** enables smooth, accessible column reordering
- **Generic column config:** type-safe, flexible, developer-friendly
- **Accessibility:** to be implemented after core features are complete

---