Core Sheet Model — Architecture and Plan

Context

- Runs client-first inside a Next.js app route (client component).
- Real-time collaboration via Yjs over a Node WebSocket server.
- Persistence to PostgreSQL (event log + periodic snapshots).
- Rendering with Canvas2D initially; WebGL can be introduced later without changing the model.
- Minimal server logic; compute is local unless explicitly offloaded.

Primary Responsibilities

- Hold a declarative, immutable-ish model of a “Sheet” (board) as the single source of truth.
- Provide deterministic command processing (undo/redo).
- Maintain derived indices for speed (spatial index, z-order, anchors).
- Enable artifact extensibility (MDX, Mermaid, TaskList, Box, Connector, etc.).
- Support presence and multi-user merges via CRDT.
- Stream exports (SVG/PNG/JSON) without coupling to UI.

Functional Scope (Stage)

- Create/move/resize/delete artifacts.
- Connect artifacts with orthogonal connectors.
- Snap to grid, edges, centers.
- Grouping/layers/z-order.
- Multi-select and marquee.
- MDX and Mermaid artifacts (rendered client-side).
- Export to PNG/SVG.
- Real-time cursors and selections.
- Postgres persistence with snapshots.

Minimal Data Structures (Essential Types)

```
type Id = string;

type Vec2 = { x: number; y: number };
type AABB = { x: number; y: number; w: number; h: number }; // axis-aligned

type Sheet = {
  id: Id;
  name: string;
  config: { grid: number; snap: boolean; theme: 'light'|'dark' };
  // Normalized stores
  artifacts: Record<Id, Artifact>;
  edges: Record<Id, Edge>;
  order: Id[]; // z-order, back→front (only artifact ids)
  // Ephemeral derived indexes (not persisted, recomputed)
  _derived?: {
    spatial: QuadtreeIndex;
    anchors: AnchorIndex;
    selection: Set<Id>;
    dirtyAABBs: AABB[];
  };
};

type Artifact =
  | { id: Id; type: 'box'; geom: AABB; style: Style; props: { label?: string } }
  | { id: Id; type: 'mdx'; geom: AABB; style: Style; props: { source: string } }
  | { id: Id; type: 'mermaid'; geom: AABB; style: Style; props: { source: string } }
  | { id: Id; type: 'tasklist'; geom: AABB; style: Style; props: { items: TaskItem[] } };

type Edge = {
  id: Id;
  from: { id: Id; port?: string };
  to:   { id: Id; port?: string };
  type: 'ortho';
  props?: { labels?: string[] };
};

type Style = { fill?: string; stroke?: string; strokeWidth?: number; radius?: number };

type Event = { ts: number; actor: Id; type: string; payload: unknown }; // appended to event log

// Command -> Reducer op
type Command =
  | { type: 'ART_CREATE'; art: Artifact }
  | { type: 'ART_PATCH'; id: Id; geom?: AABB; style?: Style; props?: any }
  | { type: 'ART_DELETE'; id: Id }
  | { type: 'EDGE_CREATE'; edge: Edge }
  | { type: 'EDGE_DELETE'; id: Id }
  | { type: 'ORDER_BRING_FRONT'; id: Id }
  | { type: 'ORDER_SEND_BACK'; id: Id }
  | { type: 'SELECT_SET'; ids: Id[] }
  | { type: 'SELECT_ADD'; ids: Id[] }
  | { type: 'SELECT_CLEAR' }
  | { type: 'SHEET_CONFIG'; config: Partial<Sheet['config']> };
```

Notes

- `Artifact` is a tagged union. New types extend by adding a case and a schema.
- `Sheet._derived` is ephemeral; rebuilt after each reducer tick.
- Geometry is world-space AABB. Rotation deferred out of stage scope to keep router and snap simple.

CRDT + Event Sourcing

- CRDT document per sheet: `Y.Map()` containing maps for `artifacts`, `edges`, `order`, `config`.
- Local commands reduce into CRDT changes (deterministic mapping).
- The server stores incoming CRDT updates into an append-only `events` table and periodically materializes `snapshots`.
- Undo/redo on client as a command stack; CRDT stays canonical across peers.

Derived Indices

Spatial Index (Quadtree)

Purpose: O(log n) insert/search for hit-testing, marquee selection, and snap candidate lookup.

- Node stores a bounded array of entries `{ id, aabb }`.
- Split threshold `K` (e.g., 8).
- Rebuild incrementally: on `ART_PATCH` remove+insert that id; on `ART_CREATE/DELETE` insert/remove.
- Exclude non-visual artifacts if added later.

Operations:

- `queryPoint(p: Vec2): Id[]` → for click hit-test.
- `queryAABB(r: AABB): Id[]` → for marquee selection and snap candidate window.
- `nearestAnchors(p: Vec2, r: number): Anchor[]` → small radius search for snapping.

Libraries: `rbush` or `flatbush` acceptable; custom quadtree is trivial and predictable.

Anchor Index

Purpose: snapping and connector port resolution.

- For each artifact, precompute anchors: `{type:'center'|'midL'|'midR'|'midT'|'midB'|'corner', pos:Vec2}`.
- Store in a multimap `artifactId -> Anchor[]`.
- For connectors, expose route waypoints as anchors when needed.

Computation:

- Rebuild per artifact change; O(1) per artifact (constant number of anchors).

Z-Order

- Maintain a flat `order: Id[]`.
- Hit-test resolves candidates from spatial index, then sort by `order` descending to get the topmost id.
- Bring-to-front/back operations mutate the array; keep O(1) by tracking an index map if necessary.

Dirty Regions

- Each reducer returns a list of changed AABBs.
- Renderer unions overlapping regions and redraws minimal tiles.

Interaction Pipeline

1. **Pointer input** (down/move/up) captured on Canvas.
2. **Hit-test**
   - `queryPoint` for primary; if multiple, choose topmost via z-order.
   - For connectors, include edge hit area by expanding segments to a thickness for picking.
3. **Gizmo**
   - Selection gizmo, transform handles, connector dragger.
   - Emits high-level intents: `dragTranslate(dx,dy)`, `resize(handle,dx,dy)`, `connect(fromId,port,toPoint)`.
4. **Snap**
   - If `snap` enabled, gather candidates with `queryAABB` around the moving AABB expanded by a radius.
   - Compute deltas to nearest anchors or grid lines; apply strongest rule (exact overlap > edge align > grid).
5. **Command**
   - Convert intent (+ snapped delta) into `Command`.
6. **Reducer**
   - Pure function updates `Sheet`.
7. **Derived**
   - Update quadtree/anchors/z-order indices incrementally.
8. **Render**
   - Redraw dirty regions.
9. **CRDT Sync**
   - Apply equivalent Yjs updates; broadcast.

Determinism preserved: same command sequence → same state.

Connectors and Routing (Stage)

- Orthogonal (Manhattan) only.
- Ports: infer side based on nearest side of source/target AABBs.
- Router algorithm: grid-A* over inflated obstacle set.
  - Inflate all artifact AABBs by connector margin.
  - Start from a source port point, A* through grid cells to target port area.
  - Post-process with segment simplification (remove collinear bends).
- Cache route per edge keyed by `(fromGeom,toGeom,obstacleVersion)`.
- Recompute routes only when endpoints or nearby obstacles change (query obstacles via quadtree).

Dependencies: optional small `pathfinding` lib; otherwise custom A* is short.

Snapping

- Grid snap at `config.grid` interval.
- Object snap to anchors from nearby artifacts (quadtree query radius = a few grid units).
- Angle snap for connector creation (0/90/45°); implemented as clamped direction vectors.
- "Sticky" constraints deferred; initial stage creates no persistent constraints.

Rendering Model (Canvas2D)

- Back→front draw using `order`.
- Draw pipeline: grid → edges → artifacts → selection/handles → cursors.
- Text measuring cache keyed by `(font,size,text)`.
- Mermaid/MDX rendered in offscreen DOM to SVG/canvas bitmap, then drawn into the board rectangle; cache rasterization per source hash.
- Export to PNG/SVG: reuse same draw routines; SVG path generation mirrors Canvas calls.

Persistence Model (PostgreSQL)

Tables (minimal):

```sql
-- Project/sheet grouping
CREATE TABLE projects (
  id uuid PRIMARY KEY,
  name text NOT NULL,
  owner_id uuid NOT NULL,
  created_at timestamptz DEFAULT now()
);

CREATE TABLE sheets (
  id uuid PRIMARY KEY,
  project_id uuid REFERENCES projects(id) ON DELETE CASCADE,
  name text NOT NULL,
  config jsonb NOT NULL DEFAULT '{}',
  created_at timestamptz DEFAULT now()
);

-- Event log of CRDT deltas or normalized commands
CREATE TABLE sheet_events (
  id bigserial PRIMARY KEY,
  sheet_id uuid REFERENCES sheets(id) ON DELETE CASCADE,
  actor_id uuid NOT NULL,
  ts timestamptz NOT NULL DEFAULT now(),
  payload jsonb NOT NULL
);

-- Periodic materialized snapshots for fast load
CREATE TABLE sheet_snapshots (
  sheet_id uuid PRIMARY KEY REFERENCES sheets(id) ON DELETE CASCADE,
  version bigint NOT NULL,
  state jsonb NOT NULL,        -- normalized {artifacts, edges, order, config}
  updated_at timestamptz NOT NULL DEFAULT now()
);
```

Load strategy:

- On join: return latest snapshot + subsequent events cursor.
- Client reconstructs; CRDT merges incoming updates.

Save strategy:

- Append CRDT updates to `sheet_events`.
- Background job materializes `sheet_snapshots` every N events or T seconds.

API Surface (Stage)

HTTP (Next.js route handlers):

- `GET /api/sheets/:id` → `{ snapshot, eventsSince }`.
- `POST /api/sheets` → create sheet.
- `POST /api/sheets/:id/snapshot` → force snapshot (admin/dev).
- `GET /api/export/:sheetId.svg|png` → stream export.

WebSocket (Node/Yjs server):

- `join { sheetId }`
- `update { yUpdate }`
- `presence { cursor, selection }`

Security:

- Cookie session for HTTP; WS token bearing sheet access.

Extensibility Hooks (Sheet Level)

- `registerArtifact(kind, { schema, measure, draw, anchors, hit, routerHooks? })`
  - `schema`: Zod validation for `props`.
  - `measure`: precompute sizes (e.g., MDX content).
  - `draw`: uses a target-agnostic API; adapter maps to Canvas/SVG.
  - `anchors`: returns anchor list for snapping/ports.
  - `hit`: specialized shape hit-test if not rectangular.
  - `routerHooks`: optionally expose obstacle geometry for connectors.
- `registerTool(name, handlers)` for UI tools, independent from core.
- `registerExport(name, fn(sheet)->Blob)`.

Compute Placement

Client:

- Reducers and indices.
- Quadtree, anchors, hit-testing.
- Routing for small/medium graphs.
- MDX/Mermaid compilation and rasterization.
- Export PNG/SVG (fast path).

Server:

- Optional heavy routing/layout batches.
- Export PDF (headless renderer).
- Periodic snapshot job.
- AI agent endpoints that submit command batches.

Performance Targets

- 10k artifacts, 2k connectors at 60fps on modern laptop with Canvas2D.
- Hit-test: O(log n) per pointer event via quadtree; worst-case small constant candidate set.
- Drag frame latency ≤ 8ms at 120Hz for modest scenes; use dirty region redraw only.

Error/Recovery

- Deterministic reducers guarantee replay.
- Snapshot mismatch → client re-syncs from latest snapshot.
- Corrupt event payloads rejected by Zod; server logs and drops.

Testing

- Reducers: property-based tests on command sequences.
- Spatial index: randomized insert/remove/query invariants.
- Router: golden tests on fixture boards.
- Rendering: pixel-diff tests for primitives; skip platform font variance by using test font.

Milestone Breakdown (Sheet Model)

Week 1

- Core types and reducers.
- Quadtree + anchors + z-order.
- Canvas renderer (rect, text).
- Command bus with undo/redo.
- Grid + basic snapping.

Week 2

- Connectors + orthogonal router + edge hit-testing.
- Marquee select + grouping.
- MDX artifact (render→bitmap).
- Mermaid artifact (render→SVG→bitmap).
- PNG/SVG export.

Week 3

- Yjs sync + presence.
- Postgres events + snapshots + loader.
- Snapshot cron.

- E2E tests on create/move/resize/connect/export.
- Profiling + dirty-region optimization.

Justifications

- Client-first compute minimizes server cost and lowers perceived latency.
- CRDT + event log yields robust collaboration and replayability.
- Quadtree/anchor indices provide predictable, explainable performance.
- Tagged-union artifacts keep type-level safety with straightforward extensibility.
- JSONB storage matches evolving schemas without migrations on every artifact tweak.
- Canvas2D is sufficient for stage performance; WebGL is a drop-in renderer later.
- MDX/Mermaid as first-class artifacts validate the "content + flow" model early without server complexity.

Open Space ("Thinking Room")

- Persistent constraints (Cassowary/Kiwi) for true layout rules beyond snap.
- Id-buffer color picking if Canvas hit precision becomes a bottleneck.
- Incremental route graph (channel routing) to avoid per-edge A*.
- Multi-sheet projections sharing a single underlying graph.
- Pluggable dataflow execution graph for computed artifacts.
- On-sheet code artifacts with sandboxed evaluators.
- Semantic merges on artifacts beyond CRDT structural merges.
- WebGPU renderer for very large scenes if needed later.