1. Components = modular pieces in one app; services = modular pieces as separate networked programs. A component-based system is built from loosely coupled components inside the same application/process. A service-oriented system is built from services that run as separate programs (often separate machines/containers).

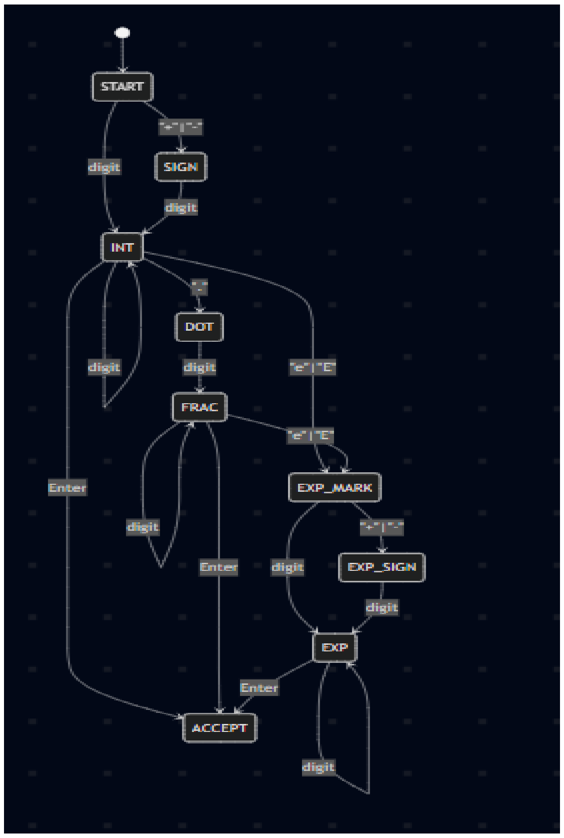
2. Most appropriate: Monolithic + Event-driven UI. No external DB, no multi-client data sharing, and no reason to introduce network layers. Phone UI is naturally event-driven. A single executable is simplest and lowest risk: monolithic fits.

3. Monolithic + event-driven UI. Still no need for servers/services if everything is local. Chess logic is more complex than tic-tac-toe, so componentizing internally is useful: UI component, Game engine rules component (legal moves, check/checkmate), Optional AI component, and Optional storage (local PGN files, settings)

4. Client/Server. Client(s): player UI + local input/board rendering. Server: authoritative game state, matchmaking/lobby, turn validation, anti-cheat checks, persistence if needed

5. Use a file-based format (e.g., custom JSON/XML/binary) representing: Canvas metadata (size, background, version) and a list of drawable objects (type + properties). And to maintain, use Mainly file management, not DB tuning. If you did store in a DB (team/collab version), then you'd talk about tables + indexing + audits. But for the base app: files + versioning + safe writes.

6.



7.

a. Properties all share

- Position (or anchor point/origin)
- StrokeColor
- StrokeWidth
- IsSelected (editor state)
- ZOrder or Layer (draw order)
- Transform-ish data: Rotation (maybe), Scale (maybe)
- BoundingBox (either stored or computed)
- Methods: Draw(), Move(dx,dy), HitTest(point), GetBounds()

b. Properties they do NOT all share

- Line: StartPoint, EndPoint (and maybe no fill)
- Rectangle/Ellipse/Star: FillColor, FillStyle, maybe IsFilled
- Star: NumPoints, InnerRadiusRatio
- Text: String, FontName, FontSize, Bold/Italic, alignment

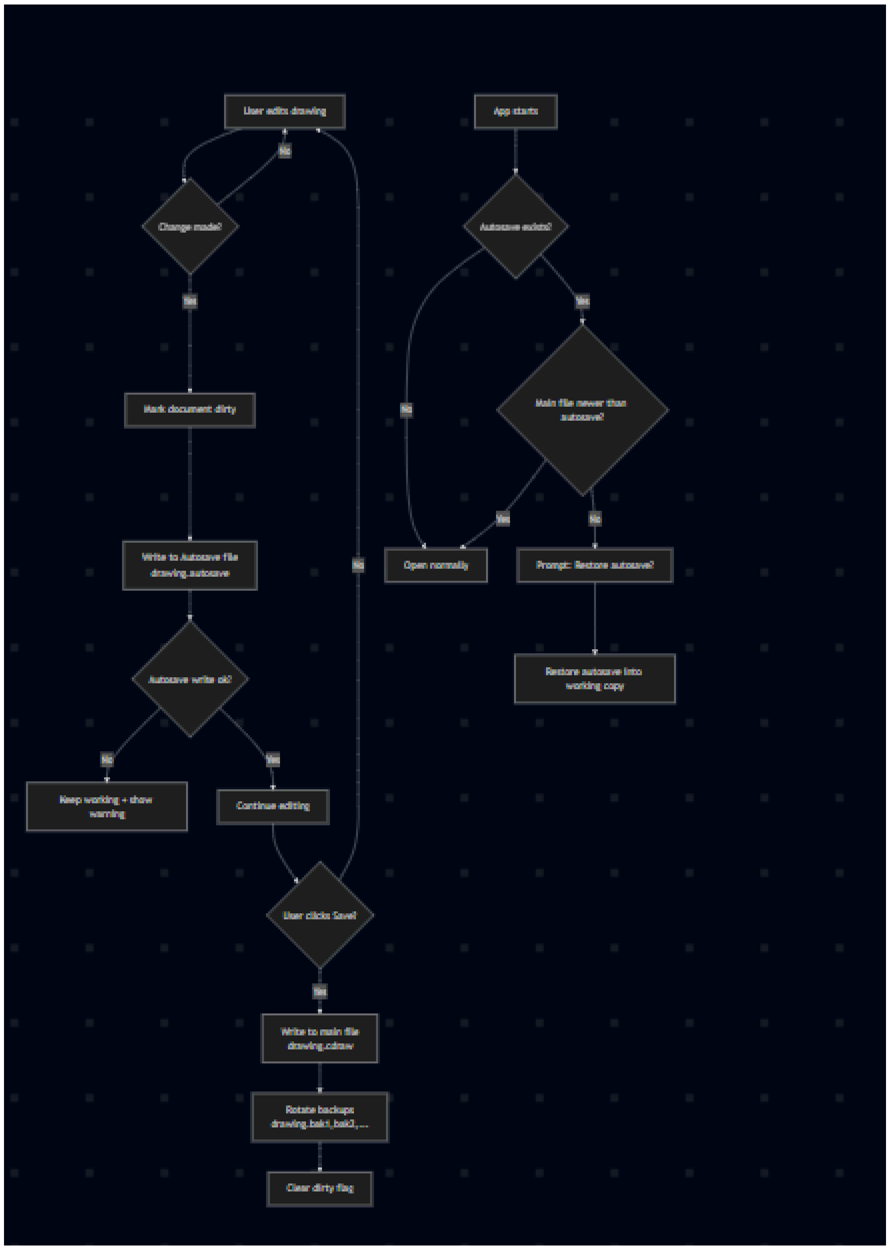
c. Properties shared by some but not others

- FillColor/FillStyle: shared by Rectangle, Ellipse, Star (but not Line; Text is special—text "fill" is basically its font color)
- Width/Height: shared by Rectangle and Ellipse (Star might use a bounding box too, but its "meaningful" params differ)
- CornerRadius: only Rectangle (if rounded rectangles exist)
- Rotation: might apply to Rectangle/Ellipse/Star/Text, but many simple editors allow rotation for all shapes; Line can also be "rotated" by changing endpoints, so rotation as a property is optional.

d. Where to implement shared/nonshared properties

- Drawable (base class): properties common to all objects + shared methods (Draw interface, Move, HitTest interface, selection state, stroke, z-order, etc.)
- Shape (optional intermediate): for geometric shapes that have outlines/fills
    - Line might be excluded or included depending on whether you define "fill" separately.
- FilledShape (optional intermediate): for Rectangle/Ellipse/Star that share fill properties
- Text: either inherits directly from Drawable (common) but has its own text/font properties.

8.


User edits drawing
No
Change made?
Yes
Mark document dirty
Write to Autosave file drawing.autosave
Autosave write ok?
No
Yes
Keep working + show warning
Continue editing
User clicks Save?
No
Yes
Write to main file drawing.cdraw
Rotate backups drawing.bak1,bak2,...
Clear dirty flag
App starts
Autosave exists?
Yes
No
Main file newer than autosave?
Yes
No
Open normally
Prompt: Restore autosave?
Restore autosave into working copy