ChatGPT

\= SPEC-1: Branching Gamebook Editor & Mobile App \:sectnums: \:toc:

\== Background

Interactive, choice-driven gamebooks such as *Lone Wolf*, *Fighting Fantasy*, and *Sorcery!* blended prose with numbered sections and branching hyperlinks. This project aims to deliver a modern descendant:

- A WYSIWYG drag-and-drop **Graph Editor** running in the browser where authors create “scene nodes”.
- Each scene node contains: **id** (sequential or UUID), **description** (rich text/markdown), **image** (optional URL/asset), **actions** (buttons) that link to other scene nodes.
- First iteration limits scope to pure branching narrative, omitting RPG mechanics, character sheets, combat, or NPC logic.
- The editor will output a self-contained content bundle that a cross-platform mobile shell (Android & iOS) can render as an offline, tappable gamebook app.

\== Requirements

// MoSCoW-prioritized requirements for Version 1 (pure branching, no RPG mechanics)

Must

- **M1** – Web-based drag-and-drop graph editor designed for **non-technical authors**.
- **M2** – Scene Node model: `id`, `description` (Markdown), optional `image`, one-tap **action buttons** → target node `id`.
- **M3** – User authentication & session management via **Supabase Auth** (email + social OAuth ok).
- **M4** – Story autosave, manual save & version history stored in **Supabase Postgres** (`stories`, `nodes`, `revisions`).
- **M5** – **Image uploads** stored in **Supabase Storage**; images linked to nodes and included in export bundle.
- **M6** – **Export** a self-contained JSON bundle that the React Native mobile shell can import.
- **M7** – Mobile shell renders story **offline** on Android & iOS, persisting reader progress locally.

Should

- **S1** – Undo / redo in the editor.
- **S2** – Live mobile-frame **preview** inside the editor.
- **S3** – Basic theming options (font size, dark/light) in reader app.

Could

- **C1** – Markdown formatting toolbar & tooltip help.

Won't Have (v1)

- **W1** – RPG mechanics, stats, combat.
- **W2** – NPC dialogues.
- **W3** – In-app monetization or paywalls.

\== Method

\=== 1. High-Level Architecture

[plantuml]

@startuml actor "Author" as A actor "Reader" as R package "Web App (Editor)" { [React + React Flow UI] as Editor [Export Module] as Exporter } package "Backend (Supabase)" { database "Postgres" as DB [Auth] as Auth [Storage] as Store } package "Mobile Shell" { [React Native App] as Mobile [Local Cache (AsyncStorage/FS)] as Cache } A --> Editor : create / edit nodes Editor --> DB : CRUD nodes, stories Editor --> Store : upload images Editor --> Auth : sign-in/out Editor --> Exporter : generate JSON bundle Exporter --> Store : attach image URLs

R --> Mobile : install app Mobile --> Store : fetch images on-demand Mobile --> Cache : cache JSON & images offline @enduml

\=== 2. Scene Node Data Model

```
{
  "id": 42,
  "description": "You stand at a fork in the road…",
  "image": "https://supabase.url/path/fork.png",
  "actions": [
    { "label": "Take the left path", "targetId": 17 },
    { "label": "Take the right path", "targetId": 55 }
  ]
}
```

\=== 3. Database Schema (Supabase Postgres)

[plantuml]

@startuml entity users { id PK email created_at } entity stories { id PK owner_id FK → users.id title synopsis created_at updated_at } entity nodes { id PK story_id FK → stories.id description text image_url text created_at updated_at } entity actions { id PK node_id FK → nodes.id label target_node_id FK → nodes.id } entity revisions { id PK story_id FK → stories.id revision_number data jsonb created_at } users ||--o{ stories stories ||--o{ nodes nodes ||--o{ actions stories ||--o{ revisions @enduml

\=== 4. Export Bundle Format

- Single `story.json` containing:
- `meta`: title, author, version, created date.
- `nodes`: array of scene nodes (as above).
- Stored alongside the app's JS bundle or downloaded into cache on first run.

\=== 5. Editor Front-End Details

Concern	Approach
Graph UI	**React Flow** for canvas, edges, drag-drop.
Rich Text	**TipTap** or **Slate.js** Markdown WYSIWYG.
State	**Zustand** for local state; autosave with debounce to Supabase.
Undo/Redo	Immer-powered history stack (max 50).
Preview	iPhone/Android frame using React Native Web & iframe.

\=== 6. Mobile Shell Details

Concern	Approach
Framework	React Native + Expo EAS Build.
Navigation	`react-navigation` stack.
Rendering	FlatList of dynamic nodes; buttons map to `navigate(targetId)`.
Image Caching	`react-native-fast-image` with disk cache; fallback to expo-filesystem.
Persistent State	`@react-native-async-storage/async-storage` (current node id).

Concern	Approach
Theming	`react-native-paper` light/dark.

\=== 7. Offline Strategy

1. On first launch the app fetches `story.json` and referenced images via HTTPS from Supabase Storage.
2. Files stored to device cache; subsequent launches read from cache unless a newer `ETag` detected.
3. If offline and cache exists → reader works fully.

\=== 8. Security & Auth Flow

- Authors authenticate via Supabase (magic-link or OAuth).
- RW access is enforced by Row-Level Security on `stories.owner_id`.
- Mobile reader is **read-only**, no auth required.

\=== 9. Error & Edge-Case Handling

- Broken image URLs show placeholder.
- Circular node paths allowed but warn author.
- Export validates graph: every action target must exist.

\== Implementation

\=== Phase 0 – Project Bootstrapping

1. **Repo & CI/CD**
2. Create monorepo (pnpm workspaces). Packages: `editor-web`, `mobile-app`, `shared-types`.
3. Configure GitHub Actions → Expo EAS for mobile, Vercel for web preview.
4. **Supabase Instance**
5. Provision project.
6. Apply SQL migrations for tables & RLS policies (see *Method §3*).
7. Enable Storage bucket `images`.
8. **Shared TypeScript schemas** (`zod`) for `Node`, `Story`, `Action`.

\=== Phase 1 – Web Editor Core

1. **Scaffold React app** with Vite + React Flow + Tailwind.
2. **Auth integration** (`@supabase/auth-ui-react`) + protected routes.
3. **CRUD API hooks** (`@supabase/supabase-js`).
4. **Graph Canvas**
5. Drag-drop node creation.
6. Edge drawing for actions.
7. Side-panel node editor (Markdown textarea, image picker).
8. **Autosave & Revisioning**
9. Debounce 1 s; upsert node JSON.
10. Nightly cron SQL to snapshot `revisions`.

11. **Undo/Redo** via Immer history stack.

\=== Phase 2 – Media Handling & Export

1. **Image Upload Widget**
2. Drag-n-drop or file-picker → Supabase Storage → returns public URL.
3. Progress bar & validation (≤ 1 MB, jpg/png/webp).
4. **Export Module**
5. Validate graph (no missing targets).
6. Generate `story.json` with `meta`, `nodes`.
7. Provide download + copy-to-clipboard Share Link (supabase storage object).
8. **Live Mobile Preview**
9. Embed RN Web build in iframe, pointing to current export.

\=== Phase 3 – Mobile Shell

1. **Expo project** with TypeScript.
2. **Initial screens**: splash → story loader → node view.
3. **JSON parsing & navigation**
4. Load from bundled asset or remote URL.
5. `navigate(targetId)` pushes new screen.
6. **Offline Caching**
7. Cache JSON & images (`expo-filesystem` + `react-native-fast-image`).
8. Connectivity listener to refresh if online.
9. **Theming** (light/dark) with context + `react-native-paper`.
10. **Persistent Progress**
11. Save last node id + history in AsyncStorage.
12. “Restart story” option.

\=== Phase 4 – QA & Distribution

1. **Unit tests** (Jest + React Testing Library).
2. **End-to-End tests** (Playwright for web, Detox for mobile).
3. **Accessibility pass** (WCAG A).
4. **App Store + Play Store prep**: icons, screenshots, privacy policy.
5. **Beta rollout**: TestFlight & Play Console Internal.
6. **Launch Day**: tag `v1.0.0`, promote to production.

\== Milestones

#	Sprint	Deliverable	Success Criteria
0	Week 1	Bootstrapped monorepo, Supabase configured	Repo builds on CI; tables exist; auth works locally.
1	Weeks 2-3	Web Editor Core	Authors can log in, create nodes/edges, autosave.

#	Sprint	Deliverable	Success Criteria
2	Week 4	Media & Export	Images upload; export validated JSON downloadable.
3	Week 5	Mobile Shell Alpha	App loads bundled story, navigates nodes offline.
4	Week 6	Full Integration	Editor → export → mobile import round-trip succeeds.
5	Week 7	QA & Beta	80% unit-test coverage; beta builds distributed.
6	Week 8	Public Launch	Apps live in stores; web editor at `v1.0`.

Total timeline: 8 weeks (2-month sprint)

\== Gathering Results

\=== Objectives

Capture actionable insights for **both authors and readers** using Supabase's built-in analytics and lightweight event tables—no external pipeline.

\=== Key Metrics

Dimension	Metric	Description
Author Engagement	**WAU / MAU**	Unique authors in editor weekly / monthly
	Stories created	Count per period
	Nodes per story	Avg. complexity
	Editor session length	Median minutes per editing session
	Export success rate	% of export attempts that pass validation
Reader Engagement	Installs	Total app installs (App / Play store figures)
	First-choice funnel	% of users who reach first node → first action
	Completion rate	% reaching an ending node
	Session length	Avg. minutes per reading session
	Image fetch errors	# failed image requests (should trend ↓)

\=== Data Collection Strategy

1. **Client Event Hook**

2. Editor and mobile app call a shared `POST /telemetry` Supabase Edge Function.
3. Payload: `user_id` (anon for readers), `event_type`, `story_id`, `additional_data`.
4. **Storage**
5. Table `telemetry_events (id, user_id, event_type, story_id, data jsonb, created_at)`.
6. Row-level security: reader events insert-only.
7. **Dashboards**
8. Supabase “Logs & Analytics” → saved charts for WAU/MAU, funnel drop-off, error rates.
9. Weekly emailed report via Supabase scheduled function.

\=== Success Criteria (v1)

- ≥ 50 **active authors** in first 30 days.
- ≥ 20 published stories.
- ≥ 75 % reader retention Day-1 → Day-2.
- ≤ 2 % export validation failures.
- ≤ 1 % image fetch errors after caching patch.

\=== Feedback Loops

- **In-editor survey** (thumbs-up / thumbs-down + free-text) surfaces after export.
- **Crash & error logging** via Supabase Edge (plus Sentry for stack traces).
- Monthly roadmap review driven by telemetry and user feedback.