# **A — Bắt đầu từ đâu (ưu tiên ngay)**

1. **Mục tiêu MVP rõ ràng**
   * Web app PC-first, text-only, support: start run → show event → choose → apply result → save.
   * Không vẽ art/animation ở giai đoạn này.
2. **Tạo repository & skeleton**
   * Tạo monorepo (ví dụ: whatif/) chứa frontend/ + backend/ + shared/ (types/schemas).
   * Khởi tạo CI (GitHub Actions) chỉ build/test.
3. **Implement minimal loop ASAP**
   * Backend endpoint: POST /api/new-run (create profile + seed)
   * Backend endpoint: GET /api/next-event?runId=... (select next event from canonical pool by seed/state)
   * Backend endpoint: POST /api/choose (apply choice, return result + updated state)
   * Frontend: show event + 2–3 choices, stat panel, timeline log.
4. **Build a small canonical event pool** (30 events JSON) để test game flow — no AI at first.
5. **Quick playtest** with friends, collect feedback, then iterate.

# **B — Stack & kiến trúc (lựa chọn + lý do)**

Mình đưa ra 2 đường khả thi — **Đề xuất A (ưu tiên)** cho tốc độ & ecosystem, và **Đề xuất B** nếu bạn muốn tận dụng kinh nghiệm .NET của bạn.

### **Đề xuất A — TypeScript fullstack (mình khuyến nghị)**

* Frontend: React + Vite (TypeScript)
* Backend: Node.js + Express hoặc Fastify (TypeScript)
* DB: PostgreSQL (jsonb cho event templates)
* Cache / Queue: Redis (cache AI/pregen + rate limit)
* Shared types: shared/ package (TypeScript) để tránh mismatch schema
* Lý do: Type safety, chia sẻ type giữa frontend/backend dễ dàng, cộng đồng nhiều libs (seeded RNG, validator), nhanh triển khai.

# **C — Kiến trúc hệ thống & cấu trúc mã (clean, modular, testable)**

## **1) Kiến trúc tổng quát (monolith modular)**

graphql

CopyEdit

whatif/

├─ backend/

│ ├─ src/

│ │ ├─ controllers/ # API endpoints

│ │ ├─ services/ # EventService, RunService, AIService, CacheService

│ │ ├─ models/ # DB models / interfaces

│ │ ├─ validators/ # JSON schema validators (ajv)

│ │ └─ workers/ # background pregen / AI jobs

│ └─ tests/

├─ frontend/

│ ├─ src/

│ │ ├─ components/

│ │ ├─ pages/

│ │ ├─ lib/ # API clients, utils

│ │ └─ types/ # import from shared or generated client

├─ shared/ # types, JSON schemas, example events

└─ infra/ # docker-compose, k8s manifest, CI

## **2) Module chính ở backend**

* **RunService**: createRun(seed, profile), getRunState(runId), advanceTime.
* **EventService**: selectNextEvent(runState) — picks from pools (scripted → ai pool → fallback). Implements branch weighting, prerequisites check, seed-based RNG determinism.
* **ResolveService**: resolveChoice(runId, eventId, choiceId) — applies effects, enqueues chain events.
* **AIService**: prompt templates, call LLM (local or API), validate JSON output, cache result.
* **MetaService**: compute run score, grant points, apply purchases (perks/talents).
* **PersistService**: abstraction over DB (player, run, timeline, event\_cache).
* **AnalyticsService**: log choices, heatmap events.

## **3) Deterministic RNG (quan trọng)**

* Use seeded PRNG so same seed + same choices ⇒ same results (reproducible life).
* In Node.js: seedrandom or implement e.g. Mulberry32. In .NET: Random(seed) or deterministic library.
* Always pass the PRNG instance through selection pipeline, never call global random.

## **4) Event selection algorithm (pseudo)**

1. Build candidate list = events.filter(prereqs satisfied & min\_age/max\_age).
2. For each candidate compute score = base\_weight \* rarity\_factor \* modifiers(runState).
3. Normalize weights → pick with weighted random (using seeded PRNG).
4. If no candidate → fallback to generic random event.

## **5) Data model (short)**

* events (id, type, min\_age, max\_age, prerequisites JSON, choices JSON, tags, branch\_weight, source: script|ai)
* runs (id, player\_id, seed, started\_at, current\_age, state JSON)
* timeline (run\_id, age, event\_id, choice\_id, result)
* ai\_cache (signature\_hash, generated\_event\_json)
* talents / perks / meta\_points tables

## **6) JSON schema example (event) — keep strict**

(You already have a sample in the doc. Add JSON Schema & use AJV (TS) or Newtonsoft in .NET to validate AI outputs.)

## **7) Frontend component structure**

* NarrativePanel — shows event title, description, optional image.
* ChoiceButtons — shows 2–4 choices, disabled when resolving.
* StatPanel — shows visible stats + energy + age.
* Timeline — list of past events (collapsible).
* MetaShop — buy talents/perks between runs.

## **8) AI integration pattern (safe)**

* **Job queue**: background worker pre-generates batches of AI events and stores them in ai\_cache.
* **Runtime**: EventService tries scripted first → cached AI event next → on-the-fly AI call only when necessary.
* **Validation**: validate JSON schema; if fail -> retry n times -> fallback to scripted event.
* **Prompting**: pass only necessary context, enforce seed and tone.

# **D — Dev workflow, testing, deploy & ops (practical)**

* **Git flow**: trunk-based or feature-branch + PR. Use issues/milestones matching roadmap.
* **CI**: run lint, unit tests, schema validation on PR.
* **Docker**: containerize backend + db + redis for dev.
* **Staging**: deploy to cheap provider (Railway/Render/Heroku) for beta. Frontend to Vercel. DB on Supabase or Railway Postgres.
* **Monitoring**: Sentry for errors + custom analytics events to DB for decision heatmap.
* **Backups**: daily DB backups.

# **E — Security & safety**

* Sanitize any AI-generated text before storing/display.
* Rate-limit AI endpoints & user actions.
* Validate all incoming JSON on backend.
* Be careful storing PII.

# **F — 2–4 tuần đầu — checklist (concrete tasks)**

Week 0 (setup):

1. Create monorepo + init frontend (React + Vite + TypeScript).
2. Init backend (TS + Express or ASP.NET template) + Docker compose (postgres + redis).
3. Create shared types folder + basic Event JSON schema.

Week 1:  
 4. Implement RunService + in-memory event pool loader.  
 5. Implement GET /api/next-event and POST /api/choose (logic: apply effects, update state).  
 6. Frontend: minimal UI to call /api/next-event and render choices; implement stat panel.

Week 2:  
 7. Add deterministic seeded PRNG to selection & resolution.  
 8. Create 30 canonical events and test several runs, save timeline.  
 9. Add basic save/load and 3-slot saves.  
 10. Manual playtests and simple logging of choices.

Week 3–4:  
 11. Implement meta scoring and simple meta shop (1–2 talents/perks).  
 12. Add background worker stub for AI pregen (no calls yet).  
 13. Prepare for closed beta — deploy staging & invite 5–10 testers.

# **G — Extras & tips**

* **Keep shared/ types authoritative**: it eliminates mismatches between frontend and backend.
* **Use feature flags** for AI features so you can turn them on/off in prod.
* **Instrument everything**: store event\_id, choice\_id, run\_id for analysis.
* **Start small**: 30 good events + a tight flow beats 300 low-quality events.