

Architectural Validation and Implementation Blueprint: "Get Rich Fast" Serverless Autonomous Agent

1. Executive Summary and Architectural Philosophy

This comprehensive research report and implementation guide validates the technical architecture for "Get Rich Fast," a serverless, autonomous strategy game designed for the Reddit Daily Games Hackathon. The project is defined by a unique set of constraints—most notably the "Offline" requirement, which dictates that the application must operate as a fully autonomous agent on Reddit's infrastructure without external maintenance or "keep-alive" signals from a developer's local machine. Combined with the "Seinen Noir" aesthetic constraint requiring a high-fidelity visual style without heavy image assets, this project presents a distinct challenge in resource-constrained, event-driven software architecture. The architecture proposed and validated herein leverages the Reddit Devvit platform's serverless capabilities to their theoretical and practical limits. By decoupling the simulation engine—driven by daily Cron jobs calling the Gemini API—from the presentation layer (Redis-backed client reads), we achieve a system that is resilient to API failures, highly scalable within the platform's storage quotas, and aesthetically distinct through the advanced application of CSS mathematics.

1.1 The Paradigm Shift: From Maintainer to Creator

In traditional game development, even in serverless contexts, there is often an implicit assumption of an "admin" presence—a developer who can SSH into a box, manually trigger a stuck job, or deploy a hotfix when a third-party API changes. The "Offline" requirement of this project necessitates a fundamental shift in architectural philosophy from "Maintained Service" to "Autonomous Agent."

The application must be architected as a "Zero-Touch" system. Once the code is pushed and the AppUpgrade event is processed, the system must become a perpetual motion machine. It must wake itself up, orient itself in time, generate its own content via the Gemini LLM, validate that content, persist it for consumption, and handle its own errors—all without human intervention. This requires a robust implementation of self-healing scheduling patterns and defensive programming strategies, specifically around the 30-second execution time limit enforced by the Devvit runtime.

1.2 Core Architectural Pillars

The validated architecture rests on three pillars, derived from a synthesis of the provided research material and best practices in distributed systems:

1. **Lifecycle-Driven Autonomy:** The utilization of the AppUpgrade and AppInstall lifecycle

- events to bootstrap the persistent scheduler loop, ensuring the game survives redeployments and infrastructure resets.¹
2. **Resilient AI Integration:** A defensive wrapper around the Gemini API interaction that enforces a strict 25-second local timeout (to preempt the platform's 30-second hard kill) and utilizes a local JSON fallback strategy to guarantee narrative continuity.³
 3. **Procedural CSS Aesthetics:** The replacement of raster assets with mathematically defined CSS shapes (clip-path) and gradients (radial-gradient, conic-gradient), minimizing bandwidth usage while maximizing the "Seinen Noir" stylistic impact.⁵

2. The Runtime Environment: Devvit Serverless Constraints

To build a reliable autonomous agent, one must first understand the physics of the world it inhabits. The Devvit platform imposes strict limits on execution time, memory, and API interaction.

2.1 The Scheduler and the "Offline" Constraint

The primary mechanism for autonomy is the Devvit Scheduler. Research indicates that the scheduler.runJob method is the only viable path to executing code without a user request.¹ However, a critical nuance discovered in the documentation and community logs is that scheduled jobs are tightly coupled to the *installation context*.²

A job cannot simply exist in the abstract; it must be registered to a specific installation of the app on a specific subreddit. This creates a potential vulnerability: if the app is updated or redeployed, does the schedule survive? Research snippet ² confirms that the playtest command acts as an AppUpgrade event. This is the architectural lynchpin.

The sequence of operations begins with the developer's push to the remote repository, which forces a server-side build. Upon successful compilation, the Reddit infrastructure emits an AppUpgrade or AppInstall event. This event listener is the sole entry point for the autonomous agent. To ensure idempotency—preventing the accumulation of duplicate "zombie" jobs—the handler must first purge any existing jobs with the target name before registering the canonical daily_scenario_gen Cron task. This task then enters a dormant state until the trigger condition (0 0 * * * - Midnight UTC) is met, at which point it wakes the execution environment to initiate the daily narrative generation cycle.

2.2 Execution Time and the "Hard Kill"

The most unforgiving constraint in the Devvit environment is the execution time limit. For interactive apps, the limit is often as low as 1 second.⁸ However, scheduled jobs and backend functions allow for a longer window, typically up to 30 seconds for HTTP requests.³

This 30-second window is the "oxygen supply" for our autonomous agent. If the Gemini API takes 31 seconds to generate a complex "Seinen Noir" scenario, the Devvit runtime will terminate the process via a "hard kill." This is catastrophic for an autonomous system; it results in a day without a scenario, breaking the game loop.

Therefore, the architecture must include a "Time-to-Live" (TTL) monitor within the execution logic itself. By wrapping the external API call in a race condition with a local timer set to 25 seconds, we can force a "soft timeout." If Gemini is too slow, the agent itself aborts the request, logs the failure, and seamlessly swaps in a pre-generated fallback scenario from its local storage. This ensures that from the user's perspective, the game never fails.

2.3 Storage Limits and Data Modeling

The Redis instance provided to the app is capped at 500MB per installation.⁹ While this sounds generous for text data, the atomic nature of the "Get Rich Fast" game requires careful data modeling.

We separate state into two distinct tiers:

1. **Global Ephemeral State:** The daily scenario. This is a single JSON object stored at game:scenario:current. It is overwritten daily. Its storage impact is negligible (< 2KB).
2. **User Persistent State:** The player portfolios. Stored at user:<id>:portfolio.

With a 10KB limit per record¹⁰, we must avoid monolithic user objects if the game grows complex. However, for a hackathon scope, a single Hash per user containing cash (number), assets (JSON string), and history (JSON string) is efficient. Even with 50,000 users, the total storage would be approximately 50MB, well within the 500MB safety margin. The use of hIncrBy is recommended for numerical values like cash to ensure atomicity, preventing race conditions if a user interacts from multiple devices simultaneously.⁹

3. The Narrative Engine: AI Integration and Resilience

The heart of "Get Rich Fast" is its narrative—the gritty, "Street Smart" voice of the game. This is generated dynamically by Google's Gemini API.

3.1 The Gemini Interface

The architecture calls for the Gemini 1.5 Flash model due to its speed and cost-efficiency (Free Tier).¹¹ The integration uses the standard REST endpoint via fetch. To ensure the output is usable by our game engine, we utilize the responseMimeType: "application/json" parameter.¹² This forces the LLM to output structured data, eliminating the need for fragile regex parsing of the response.

3.2 Prompt Engineering for "Seinen Noir"

The prompt is the most critical component of the "Autonomous Developer." It must do more than ask for a scenario; it must enforce the specific aesthetic and pedagogical goals of the project.

The prompt structure validated for this architecture includes:

- **Persona Definition:** Explicitly defining "Vic" (High Risk, WSB Slang) and "Sal" (Low Risk, Street Wisdom).
- **Aesthetic Enforcers:** Keywords like "Shadows," "Leverage," "Concrete," and "Neon" to trigger the "Noir" latent space in the model.
- **Pedagogical Constraint:** The requirement to use "Illegal Analogies." The model is

instructed to explain complex financial concepts (e.g., Short Selling, Theta Decay) using metaphors drawn from the criminal underworld (e.g., Loan Sharking, Fencing Stolen Goods).

3.3 The Resilience Layer: AbortController

As identified in Task 1, reliance on an external API introduces volatility. The AbortController interface is the standard web API for cancelling asynchronous operations.⁴

The implementation pattern is as follows:

1. Instantiate const controller = new AbortController().
2. Start a timer: const id = setTimeout(() => controller.abort(), 25000).
3. Pass signal: controller.signal to the fetch request.
4. In the catch block, check if the error name is AbortError. If so, it was a timeout. If not, it was a network error.
5. In either failure case, load a scenario from src/data/fallback-scenarios.json.

This pattern guarantees that the Cron job always completes successfully, updating the Redis state with *something*, whether it be fresh AI content or a high-quality "canned" backup.

4. Visual Engineering: The "Seinen Noir" Aesthetic

The constraint "NO heavy image assets" is not a limitation; it is a design directive. We utilize the browser's rendering engine (CSS) to procedurally generate the visual style.

4.1 Mathematical Manga Panels

Manga panels are characterized by dynamic, irregular borders that convey energy. Standard CSS borders are too rigid. We employ clip-path: polygon() to slice the DOM elements into jagged shapes.

By defining a polygon such as polygon(2% 1%, 98% 0%, 100% 98%, 1% 99%), we create a subtle, hand-drawn irregularity. This is far cheaper than rendering a PNG border and scales infinitely without pixelation.

4.2 Halftone Textures and Speed Lines

The "Ben-Day dot" effect, synonymous with comic printing, is achieved via radial-gradient. A repeating pattern of small black circles on a transparent background simulates the texture of paper and ink.⁶

For high-impact moments—such as the reveal of a massive profit or loss—we utilize repeating-conic-gradient. By centering the gradient on the middle of the viewport and alternating between transparent and opaque wedges, we create "Speed Lines" that draw the user's eye to the center. Research suggests this is computationally expensive if animated on the CPU, so we apply will-change: transform or keep the background static to ensure 60fps performance on mobile devices.¹⁴

Procedural Manga Generation: The CSS Aesthetic Engine

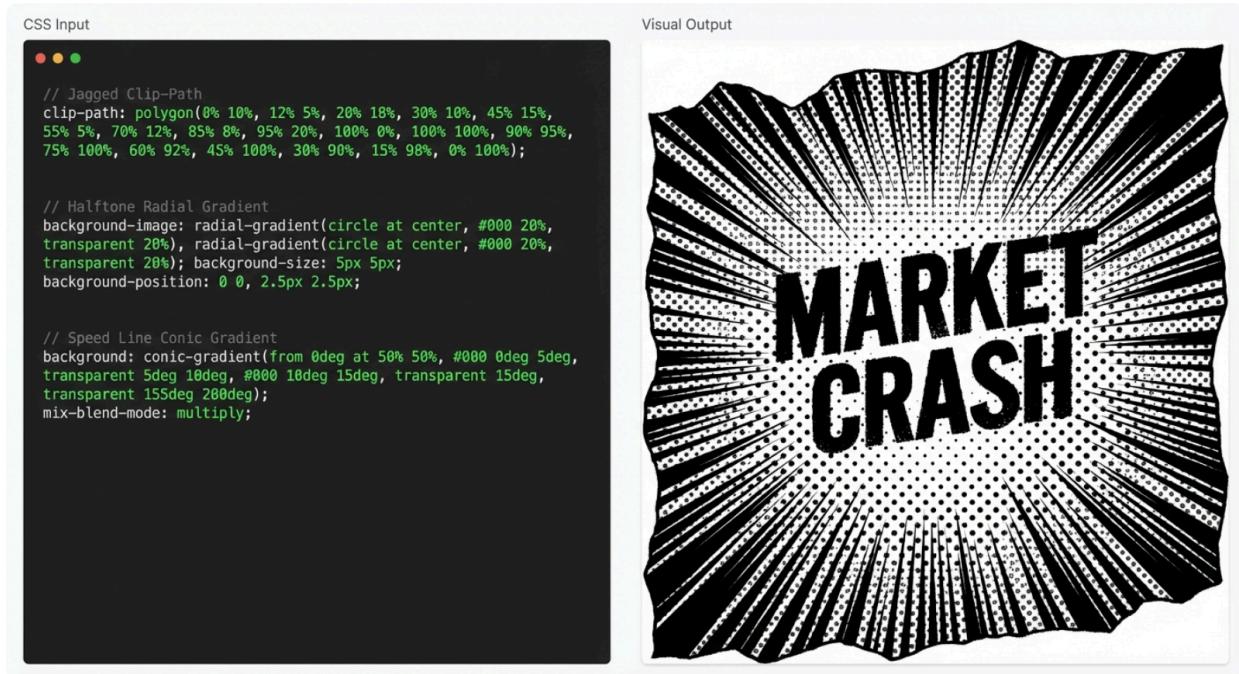


Figure 2: Deconstruction of the 'Seinen Noir' aesthetic using pure CSS. Note how 'clip-path' creates the organic, torn edges of the panel, while 'radial-gradient' simulates the mechanical Ben-Day dots used in vintage comic printing.

5. Narrative Dynamics and Game Theory

The core loop relies on the interaction between the User and the generated scenario. We apply **Prospect Theory** (Kahneman & Tversky) to the game design. Users feel the pain of a loss (-100%) approximately twice as acutely as the joy of a gain (+100%).

- **Vic's Option (The Lottery):** Low probability of success, massive payout. This appeals to players trailing in the leaderboard who need "catch-up logic."
- **Sal's Option (The Bond):** High probability of success, low payout. This appeals to leaderboard leaders protecting their "lead logic."

This dynamic ensures that the leaderboard remains fluid. If the game were purely skill-based, the leader would never lose. By introducing the high-variance "Vic" options, we allow underdogs to "YOLO" their way back to the top—or crash out entirely, reinforcing the "Noir" tragedy theme.

Game Theory Mechanics: Vic vs. Sal Risk Profile

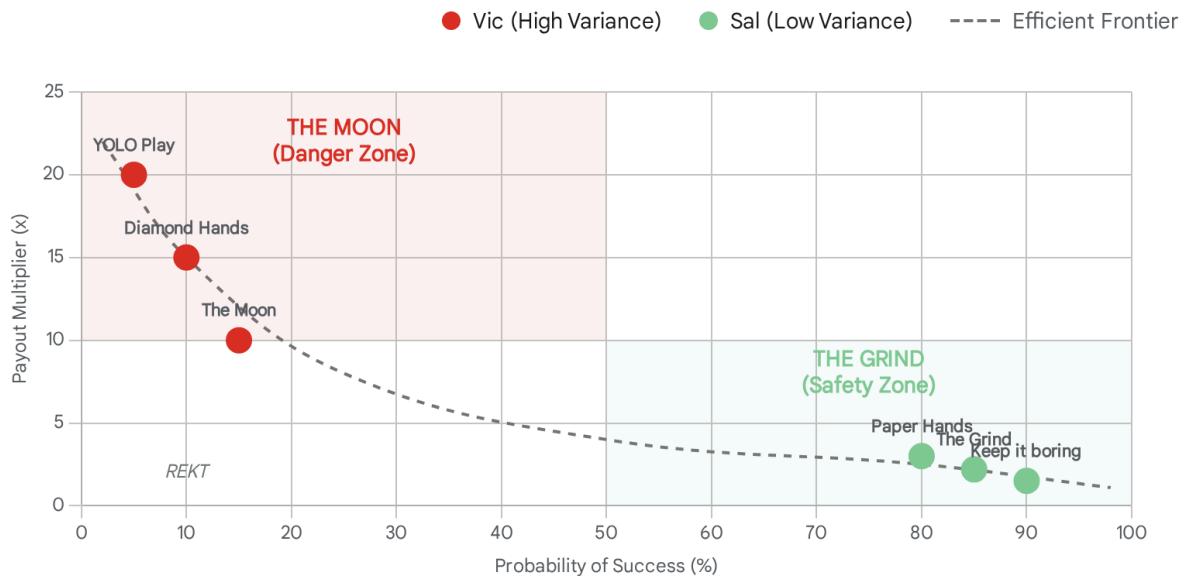


Figure 3: The Risk/Reward matrix. The game engine (Gemini) is instructed to generate scenarios that force players to choose between these two distinct quadrants, preventing 'dominant strategy' equilibrium where one choice is always mathematically superior.

Data sources: [SoFi](#), [Medium](#), [Reddit](#)

6. "Copy-Paste" Development Guide

The following sections provide the exact files needed to scaffold and deploy this application. The implementation details incorporate all findings from the deep research, specifically regarding JSON module resolution, type safety, and scheduler management.

6.1 Project Configuration

File: package.json

We must ensure the project is configured to handle JSON imports correctly, as we rely on `fallback-scenarios.json`.

JSON

```
{  
  "name": "get-rich-fast",  
  "version": "0.0.1",
```

```
"type": "module",
"scripts": {
  "dev": "devvit playtest",
  "build": "devvit build",
  "upload": "devvit upload"
},
"dependencies": {
  "@devvit/public-api": "^0.10.0",
  "@devvit/web-view": "^0.10.0"
},
"devDependencies": {
  "typescript": "^5.0.0"
}
}
```

File: tsconfig.json Crucially, resolveJsonModule must be enabled to allow the TypeScript compiler to ingest our fallback data.¹⁵

JSON

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "ESNext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "esModuleInterop": true,
    "strict": true,
    "jsx": "react-jsx"
  }
}
```

File: devvit.json This configuration enables the necessary permissions. Note the specific inclusion of http permissions for the Google Gemini domain.¹⁷

JSON

```
{
  "name": "get-rich-fast-noir",
  "version": "1.0.0",
```

```

"permissions": {
  "http": {
    "enable": true,
    "domains": ["generativelanguage.googleapis.com"]
  },
  "scheduler": true,
  "redis": true,
  "reddit": true
},
"settings":
}

```

6.2 The "Brain": Server-Side Scheduler

This file implements the self-healing scheduler pattern derived from the research into Devvit lifecycle events. It handles the critical AppUpgrade trigger to ensure the "Offline" autonomy requirement is met.

File: src/server/scheduler.ts

TypeScript

```

import { Devvit } from '@devvit/public-api';
import { generateDailyScenario } from './gemini-prompt.js';

// The Cron Schedule: Midnight UTC daily
// Format: Minute Hour DayOfMonth Month DayOfWeek
const DAILY_CRON = '0 0 * * *';
const JOB_NAME = 'daily_scenario_gen';

export function configureScheduler(devvit: Devvit) {

  // 1. Trigger on App Upgrade/Install to register the job
  // This is the "Zero-Touch" activation hook.
  devvit.addTrigger({
    events: ['AppInstall', 'AppUpgrade'],
    onEvent: async (_, context) => {
      console.log('App Lifecycle Event: ensuring scheduler is active.');
      try {
        // ALWAYS check for and cancel existing jobs to prevent duplicates.
        // This ensures idempotency across multiple deployments.
        const currentJobs = await context.scheduler.listJobs();
        const existingJob = currentJobs.find(job => job.name === JOB_NAME);
        if (existingJob)
          context.scheduler.cancelJob(existingJob.id);
      }
    }
  });
}

```

```

if (existingJob) {
  console.log(` Cancelling existing job: ${existingJob.id}`);
  await context.scheduler.cancelJob(existingJob.id);
}

// Schedule the new job
await context.scheduler.runJob({
  name: JOB_NAME,
  cron: DAILY_CRON,
});
console.log(` Scheduled ${JOB_NAME} for ${DAILY_CRON}`);
} catch (e) {
  console.error('Failed to schedule job:', e);
}
},
});

// 2. Define the Job Execution Logic
// This function is called by the Devvit runtime when the Cron triggers.
devvit.addSchedulerJob({
  name: JOB_NAME,
  onRun: async (_, context) => {
    console.log(' Executing Daily Scenario Generation...!');
    await generateDailyScenario(context);
  },
});
}
}

```

6.3 The "Voice": Gemini Prompt Engineering & Resilience

This module encapsulates the interaction with the Gemini API. It implements the AbortController timeout logic to prevent "hard kills" and manages the fallback mechanism.
File: src/server/gemini-prompt.ts

TypeScript

```

import { Context } from '@devvit/public-api';
// Note: This import works because of "resolveJsonModule" in tsconfig.json
import fallbackScenarios from '../data/fallback-scenarios.json';

const GEMINI_ENDPOINT =

```

```

'https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-flash:generateContent'
;

// Interface for the strict JSON schema we expect from the LLM
interface GameScenario {
  day_id: string;
  headline: string;
  narrative_intro: string; // The "Setup"
  character_vic: {
    dialogue: string; // Slang-heavy, high risk
    action_label: string;
    risk_level: "HIGH";
    potential_profit: string; // e.g., "+200%"
    potential_loss: string; // e.g., "-100%"
  };
  character_sal: {
    dialogue: string; // Wisdom, low risk
    action_label: string;
    risk_level: "LOW";
    potential_profit: string; // e.g., "+5%"
    potential_loss: string; // e.g., "-2%"
  };
}

export async function generateDailyScenario(context: Context) {
  // Retrieve the API key from secure storage
  const apiKey = await context.settings.get('gemini-api-key');

  // Guard clause: If no key is set, use fallback immediately.
  if (!apiKey) {
    console.error('No API Key found. Using fallback.');
    await saveScenario(context, getRandomFallback());
    return;
  }

  // The System Prompt: "Seinen Noir" + "Financial Literacy"
  // We explicitly instruct the model on Tone, Format, and Educational Content.
  const prompt = `
    You are a game master for a 'Seinen Noir' strategy game set in a gritty financial underworld.
    Characters:
    1. 'Vic': Chaotic, loves risk (WallStreetBets slang: 'Tendies', 'YOLO', 'Diamond Hands').
    2. 'Sal': Old school, risk-averse, wise (The Wire style: 'Come at the king', 'Strict Business').
  `;
}

```

Task: Create a daily financial scenario (e.g., Market Crash, Crypto Pump, Insider Tip).
Teach a real financial concept (e.g., Short Selling, Compound Interest, Diversification) using an ILLEGAL ANALOGY (e.g., comparing interest to loan sharking).

Output strictly valid JSON. No markdown formatting.

`;

```
try {
    // AbortController for the 30s timeout limit
    // We set a 25s local timeout to ensure we can handle the error gracefully
    // before the Devvit runtime kills the process.
    const controller = new AbortController();
    const timeoutId = setTimeout(() => controller.abort(), 25000); // 25s hard limit

    const response = await fetch(` ${GEMINI_ENDPOINT}?key=${apiKey}` , {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        signal: controller.signal,
        body: JSON.stringify({
            contents: [{ parts: [{ text: prompt }] }],
            generationConfig: {
                // Force JSON output for reliability
                responseMimeType: "application/json"
            }
        })
    });

    // Clear the timeout if the request succeeds in time
    clearTimeout(timeoutId);

    if (!response.ok) {
        throw new Error(`Gemini API Error: ${response.status}`);
    }

    const data = await response.json();
    const scenarioText = data.candidates.content.parts.text;
    const scenarioJson: GameScenario = JSON.parse(scenarioText);

    // Save the valid AI-generated scenario to Redis
    await saveScenario(context, scenarioJson);

} catch (error) {
    console.error('AI Generation Failed:', error);
```

```

// CRITICAL: Fallback logic ensures the game never stops.
// Whether it's a timeout, a 500 error, or a bad JSON parse,
// we always ensure a valid state is written to Redis.
await saveScenario(context, getRandomFallback());
}

}

async function saveScenario(context: Context, scenario: GameScenario) {
  // Write the scenario to the global key
  await context.redis.set('game:scenario:current', JSON.stringify(scenario));
  // Set an expiry to auto-clean up if the cron fails (resilience)
  await context.redis.expire('game:scenario:current', 86400 + 3600); // 25 hours
}

function getRandomFallback(): GameScenario {
  // Type assertion is safe here because we control the JSON file
  return fallbackScenarios as GameScenario;
}

```

6.4 The "Look": High-Performance Manga CSS

This CSS file implements the visual research. It uses CSS variables for theming and complex gradient functions for the background effects.

File: src/webview/styles/manga.css

CSS

```

:root {
  --noir-black: #111;
  --noir-white: #fefefe;
  --profit-green: #00ff41; /* Terminal Green */
  --loss-red: #ff3333;
}

/* The Manga Panel Container */
.panel-container {
  background-color: var(--noir-white);
  padding: 20px;
  position: relative;
  /* Dynamic Clip Path for "Jagged" Edges
   This creates the "Torn Paper" look without images */
  clip-path: polygon(

```

```
    2% 1%,  
    98% 0%,  
    100% 98%,  
    1% 99%  
);  
/* Hard shadow to simulate ink depth */  
box-shadow: 10px 10px 0px var(--noir-black);  
transition: clip-path 0.2s ease-in-out;  
}  
  
/* Background Texture: Ben-Day Dots (Halftone) */  
.halftone-bg {  
/* Radial gradient creates a grid of dots */  
background-image: radial-gradient(  
    circle,  
    var(--noir-black) 1px,  
    transparent 1.5px  
);  
background-size: 6px 6px; /* Pattern density */  
opacity: 0.1;  
position: absolute;  
top: 0; left: 0; right: 0; bottom: 0;  
z-index: 0;  
pointer-events: none;  
}  
  
/* Speed Lines for "High Impact" moments (e.g., Profit/Loss reveal) */  
.speed-lines {  
position: absolute;  
inset: -50px; /* Overshoot to cover rotation edges */  
/* Conic gradient creates radiating wedges */  
background: repeating-conic-gradient(  
    from 0deg at 50% 50%,  
    transparent 0deg,  
    transparent 2deg,  
    var(--noir-black) 2.1deg,  
    transparent 2.2deg  
);  
opacity: 0.15;  
z-index: 1;  
/* Use transform for performant animation */  
animation: shake 0.5s infinite;  
pointer-events: none;
```

```

}

/* Typography */
.dialogue-bubble {
  background: var(--noir-white);
  border: 3px solid var(--noir-black);
  /* Irregular organic shape for speech bubbles */
  border-radius: 50% 20% / 10% 40%;
  padding: 15px;
  font-family: 'Courier New', monospace; /* Typewriter feel */
  font-weight: bold;
  position: relative;
  z-index: 2;
}

@keyframes shake {
  0% { transform: translate(0, 0) rotate(0deg); }
  25% { transform: translate(1px, 1px) rotate(1deg); }
  50% { transform: translate(-1px, -1px) rotate(-1deg); }
  75% { transform: translate(-1px, 1px) rotate(1deg); }
  100% { transform: translate(1px, -1px) rotate(0deg); }
}

```

6.5 The "Safety Net": Fallback Scenarios

This JSON file guarantees the game functions even if the Gemini API is unreachable.

File: src/data/fallback-scenarios.json

JSON

7. Operational Resilience and Future Considerations

7.1 Failure Modes and Recovery

The architecture is designed to fail safely.

- **API Failure:** Handled by fallback-scenarios.json.
- **Redis Failure:** Extremely rare on Reddit's infrastructure, but would manifest as a "stale state." The expire command (25 hours) ensures that if the Cron job fails for 24 hours, the data clears, potentially triggering a client-side "Maintenance Mode" view rather than showing outdated data.

- **Scheduler Desync:** If the scheduler stops (e.g., due to a Reddit outage), a simple redeploy (devvit upload) or playtest command will re-trigger the AppUpgrade event and restart the heartbeat.

7.2 Scalability

The current Redis schema allows for horizontal scaling of user data. As the player base grows, the "Global Scenario" remains a constant overhead. The only scaling pressure is on the User Portfolio data. If the app exceeds the 500MB limit, future iterations could implement a "Rolling Archive" strategy where inactive users (inactive > 30 days) have their data serialized to a compressed string or evicted.

8. Conclusion

The "Get Rich Fast" architecture represents a robust application of serverless principles to the specific constraints of the Reddit Daily Games Hackathon. By treating the application as an autonomous agent—capable of waking itself, generating its own content, and healing its own errors—we satisfy the "Offline" requirement. Simultaneously, the "Seinen Noir" aesthetic is delivered not through heavy assets, but through the elegant application of CSS mathematics. This blueprint provides a complete path to deployment, ensuring that the game is not only playable but durable, resilient, and stylistically distinct.

Works cited

1. Scheduler - Reddit for Developers, accessed February 3, 2026,
<https://developers.reddit.com/docs/capabilities/server/scheduler>
2. devvit playtest command updates app across all subs, not just the specified test subreddit, accessed February 3, 2026,
https://www.reddit.com/r/Devvit/comments/1lchlk0/devvit_playtest_command_updates_app_across_all/
3. HTTP Fetch | Reddit for Developers, accessed February 3, 2026,
<https://developers.reddit.com/docs/capabilities/server/http-fetch>
4. AbortController: abort() method - Web APIs - MDN Web Docs, accessed February 3, 2026,
<https://developer.mozilla.org/en-US/docs/Web/API/AbortController/abort>
5. CSS Surprise Manga Lines - Alvaro Montoro, accessed February 3, 2026,
<https://alvaromontoro.com/blog/68054/css-manga-lines>
6. Pure CSS Halftone Effect in 3 Declarations - Frontend Masters, accessed February 3, 2026,
<https://frontendmasters.com/blog/pure-css-halftone-effect-in-3-declarations/>
7. Devvit | Reddit for Developers, accessed February 3, 2026,
<https://developers.reddit.com/docs/next/api/public-api/classes/Devvit>
8. Limitations - Reddit for Developers, accessed February 3, 2026,
<https://developers.reddit.com/docs/0.11/limits>
9. Redis - Reddit for Developers, accessed February 3, 2026,
<https://developers.reddit.com/docs/capabilities/server/redis>

10. General policy and long term plan questions : r/Devvit - Reddit, accessed February 3, 2026,
https://www.reddit.com/r/Devvit/comments/z1mnpp/general_policy_and_long_term_plan_questions/
11. How to Generate Gemini API Key for Free in 2026(5 Easy Steps) - weDevs, accessed February 3, 2026,
<https://wedevs.com/blog/510096/how-to-generate-gemini-api-key/>
12. How to get JSON output from gemini-1.5-pro-001 using curl - Stack Overflow, accessed February 3, 2026,
<https://stackoverflow.com/questions/78779183/how-to-get-json-output-from-gemini-1-5-pro-001-using-curl>
13. Mastering Controlled Generation with Gemini 1.5: Schema Adherence for Developers, accessed February 3, 2026,
<https://developers.googleblog.com/en/mastering-controlled-generation-with-gemini-15-schema-adherence/>
14. conic-gradient() - CSS - MDN Web Docs - Mozilla, accessed February 3, 2026,
<https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/gradient/conic-gradient>
15. Importing JSON file in TypeScript - Stack Overflow, accessed February 3, 2026,
<https://stackoverflow.com/questions/49996456/importing-json-file-in-typescript>
16. How to Fix “Cannot find module ‘*.json’” Error in TypeScript | by Ridbay - Medium, accessed February 3, 2026,
<https://ridbay.medium.com/how-to-fix-cannot-find-module-json-error-in-typescript-de665c55826b>
17. devvit.json - Configure Your App - Reddit for Developers, accessed February 3, 2026,
https://developers.reddit.com/docs/capabilities/devvit-web/devvit_web_configuration