Architectural Blueprint: "Duel of Minds" — Reddit Daily Games 2026

1. Executive Summary and Strategic Mandate

1.1 The Daily Games 2026 Challenge

The 2026 Reddit Daily Games Hackathon represents a pivotal moment in the evolution of social gaming on the platform. The mandate is deceptive in its simplicity: construct a "Daily Game" using the Devvit platform. However, the constraints imposed—specifically the "Zero Local Infrastructure" requirement and the exclusive use of the Gemini 2.0 Flash API (Free Tier)—transform this from a creative design challenge into a rigorous exercise in distributed systems architecture and resource orchestration.

"Duel of Minds" is conceived as the answer to this challenge. It is a daily knowledge quiz that pits the collective intellect of a subreddit against a procedurally generated challenge, or individual users against the "Mind of the Day." The core loop involves a daily reset where new content is generated, validated, and distributed without human intervention.

This report serves as the definitive technical guide for the implementation of "Duel of Minds." It is not merely a set of instructions but a comprehensive analysis of the Devvit runtime environment, the intricacies of the Gemini 2.0 Flash generative model, and the strategies required to achieve high-fidelity gameplay within strict serverless limits. We will explore the theoretical boundaries of the platform, the specific rate-limiting behaviors of Google's latest APIs, and the "density-optimized" coding patterns necessary to deliver a AAA experience in a constrained environment.

1.2 The Philosophy of Zero-Infrastructure Architecture

The "Zero Local Infra" constraint forces a fundamental shift in engineering philosophy. In traditional web development, state is cheap, and persistence is taken for granted. We spin up containers, maintain open WebSocket connections, and cache data in abundant memory. In the Devvit serverless environment, these luxuries do not exist.

We are operating in a world of:

- **Ephemerality:** Compute resources exist only for the duration of a function call (typically milliseconds to seconds).
- **Statelessness:** No memory persists between executions unless explicitly committed to the database.
- **Atomicity:** Every action that modifies shared state must be protected against race conditions, a non-trivial task without the aid of complex SQL locking mechanisms or custom server daemons.

The architecture for "Duel of Minds" therefore relies on a **Singleton Generation Pattern** coupled with a **Backend-for-Frontend (BFF)** caching layer. This approach decouples the

unpredictable latency and rate limits of the generative AI provider (Gemini) from the high-frequency, low-latency demands of the user base.

1.3 Scope of Analysis

This report is structured into two primary phases, reflecting the user's mandate:

- **Phase 1: Deep Research & Constraint Analysis:** A rigorous examination of the operational boundaries. We verify HTTP allowlists, dissect the specific token bucket algorithms of Gemini 2.0 Flash as of February 2026, and map the execution limits of the Devvit Scheduler.
- **Phase 2: Architecture & Implementation Guide:** A density-optimized technical specification. This includes the full database schema, the devvit.json configuration (replacing the requested YAML to align with platform realities), and the TypeScript logic for the LLM service and main game loop.

---

2. Phase 1: Deep Research and Constraint Analysis

2.1 Network Security and the HTTP Allowlist

The primary bridge between our closed Devvit environment and the intelligence of Gemini 2.0 is the HTTP fetch capability. Understanding the security model here is critical to preventing runtime failures.

2.1.1 The Global Allowlist vs. Explicit Permissions

Research indicates that Reddit maintains a **Global Fetch Allowlist** for widely used public APIs. As of 2026, generativelanguage.googleapis.com—the endpoint for Gemini—is explicitly listed as a globally allowed domain.[1] This is a significant strategic advantage, as it obviates the need for a manual domain approval process, which could otherwise jeopardize the hackathon timeline.

However, a nuance in the platform's security model often traps unwary developers: **Global availability does not imply automatic permission.** The application's manifest (the devvit.json file) acts as a local firewall. Even if Reddit allows a domain globally, the individual app must explicitly request access to it in its configuration. Failing to declare generativelanguage.googleapis.com in the http.domains array will result in immediate runtime errors, regardless of the global status.[1]

2.1.2 The Client-Side Sandbox (The "Webview Wall")

A critical architectural constraint verified during research is the strict bifurcation of network capabilities between the Client (Webview) and the Server.

- **Server-Side:** Can make HTTP requests to allow-listed external domains (e.g., Google, OpenAI, ESPN).[1]
- **Client-Side:** Is strictly sandboxed. It can **only** make requests to the app's own internal

API endpoints.[1]

This finding necessitates a **Backend-for-Frontend (BFF)** architecture. The React application running in the user's browser cannot communicate directly with Gemini. It must instead dispatch a request to a Devvit server endpoint (e.g., /api/get-quiz), which then proxies the request to the external service or, more likely in our optimized design, retrieves the data from Redis. This limitation is actually a security feature in disguise: it prevents the exposure of the Gemini API Key to the client-side browser, ensuring that our quota cannot be hijacked by malicious users.

2.2 Execution Constraints: The Time and Memory Walls

The Devvit runtime is a serverless environment, likely built on top of containerized ephemeral functions (similar to AWS Lambda or Google Cloud Run). This imposes hard limits on execution time and memory.

2.2.1 The 30-Second HTTP Timeout

While generic cloud functions might allow for execution times of up to 10 minutes [4], the Devvit documentation specifically flags a **30-second timeout** for HTTP fetch operations.[1] This is the "hard wall" for our AI interactions.

When we trigger a generation job:

1. **Cold Start:** The container initializes (~100-500ms).
2. **Logic Execution:** Pre-computation of prompts (~100ms).
3. **External Request:** The call to Gemini (~2s to 20s).
4. **Processing:** Parsing JSON and saving to Redis (~100ms).

The variability lies entirely in Step 3. Large language models, especially in their "Flash" or "Preview" tiers, can exhibit variable latency (Time to First Token and Total Generation Time) depending on global load. A complex "Chain of Thought" prompt that asks the model to reflect, critique, and revise its own work could easily exceed 30 seconds, causing the job to terminate abruptly.

Implication for "Duel of Minds": We must utilize **Single-Shot Prompting**. We cannot afford the latency of multi-turn conversations. The prompt must be engineered to deliver the Topic, Questions, Options, and Explanations in a single, strictly formatted JSON payload. This maximizes the probability of success within the 30-second window.

2.2.2 Scheduler Drift and Reliability

The Scheduler allows for cron-based execution (e.g., 0 0 * * * for daily at midnight).[5] However, serverless schedulers are rarely precise to the millisecond. "Drift"—the delay between the scheduled time and actual execution—is inevitable. Research suggests that while jobs are guaranteed to run, they may be queued if the platform is under heavy load.[7]

This impacts the "Midnight Release" feature. If the job runs at 00:00:05, there is a 5-second window where the day has changed but the content has not. Our architecture must handle this "staleness gap" gracefully, likely by serving the previous day's content or a generic fallback until the atomic switchover occurs.

2.3 Gemini 2.0 Flash: The Rate Limit Reality

The "Free Tier" of Gemini 2.0 Flash is the economic engine of this project, but it is also the tightest bottleneck. Misunderstanding these limits leads to the "Viral Death" scenario, where a popular app stops functioning immediately upon gaining traction.

2.3.1 The February 2026 Limits

Based on the provided research, the limits for Gemini 2.0 Flash (Free Tier) in early 2026 are as follows [8]:

- **RPM (Requests Per Minute):** 10 requests.
- **TPM (Tokens Per Minute):** 1,000,000 tokens.
- **RPD (Requests Per Day):** Approximately 500 to 1,500 (estimates vary, so we adopt the conservative 500 limit).

Analysis of the "Viral Death" Scenario:
Consider an architecture where the app generates a quiz *on demand* for each user who opens the post.

- **User 1 opens app:** 1 request. (OK)
- **User 5 opens app:** 5 requests. (OK)
- **User 11 opens app (within same minute):** 11 requests. **FAILURE.** The API returns a 429 Too Many Requests error. The app crashes for the 11th user.
- **User 501 opens app (within same day):** 501 requests. **CATASTROPHIC FAILURE.** The Daily Quota is exhausted. The app is dead for *everyone* until the next day.

This data irrefutably proves that a direct-to-LLM architecture is non-viable. The **Singleton Generation Pattern** is not just an optimization; it is a requirement. By decoupling generation from consumption, we convert "N Users" of load into "1 Request" of load.

2.3.2 The Dynamic Shared Quota

Google's documentation mentions "Dynamic Shared Quotas" for free tier models.[10] This means that during periods of extreme global demand, the 10 RPM limit might be dynamically lowered. This adds another layer of risk. Our "Backoff and Retry" logic must be robust enough to handle not just 429 errors, but also 500 or 503 errors that indicate upstream congestion.

2.4 Devvit Redis: The Atomicity Challenge

In a serverless environment, Redis is the only source of truth. However, the Devvit implementation of Redis has specific limitations compared to a standard Redis instance.

2.4.1 Storage Quotas

The limit is **500MB per subreddit installation**.[11]

- For a simple app, this is infinite.
- For a game tracking stats for 100,000 unique users, it is tight.
    - 100,000 Users * 1KB per user profile = 100MB (20% of capacity).
    - 365 Days of Quizzes * 10KB per quiz = 3.6MB (Negligible).

    - Leaderboards (ZSET) overhead = Moderate.

This confirms that **User Data Density** is the primary optimization target. We cannot store bloated JSON objects for every user. We must use compact data structures.

2.4.2 The Lua Limitation

Crucially, Devvit **does not support Lua scripts** (EVAL).[11] In standard Redis, Lua is used to execute complex transactional logic (read + logic + write) atomically. Without Lua, we must rely on the WATCH / MULTI / EXEC pattern (Optimistic Concurrency Control).[12]

The Concurrency Risk:

If we want to increment a user's "Win Streak" only if their score is above 80:

1. GET user:stats
2. (Server Logic: Check if score > 80)
3. SET user:stats

If the user plays two games simultaneously (e.g., across two devices), a race condition could occur between Step 1 and 3. WATCH detects if user:stats changed during Step 2. If it did, the EXEC fails, and we must retry the whole operation. This complicates the code but guarantees data integrity.

---

3. Architectural Blueprint (Phase 2 Design)

3.1 High-Level Architecture: The "Duel" Engine

The system architecture is defined by the separation of the **Control Plane** (automated generation) and the **Data Plane** (user interaction). This separation is enforced by the asynchronous nature of the Scheduler and the caching role of Redis.

3.1.1 System Components and Data Flow

1. **The Scheduler (The Heartbeat):** A server-side cron job running at 00:00 UTC. It is the only component that speaks to Gemini.
2. **The Generator (The Brain):** A TypeScript service that constructs the prompt, handles the API call with exponential backoff, validates the JSON schema, and commits the result to Redis.
3. **Redis (The State Layer):** Acts as the synchronization bridge. It holds:
    - quiz:{date}: The immutable quiz payload.
    - config:current_date: The pointer to the active quiz.
    - user:{id}: Compact user statistics.
    - leaderboard:{date}: The daily ZSET for ranking.
4. **The API Layer (BFF):** Devvit server endpoints (router.get, router.post) that sanitize data and handle client requests.
5. **The Client (The View):** A React-based webview that renders the game state.

Data Flow Narrative:

At midnight, the **Scheduler** wakes up. It checks if quiz:{today} exists (Idempotency Check). If not, it invokes the **Generator**. The Generator asks Gemini for a "History Quiz". Gemini returns a JSON blob. The Generator validates the schema. If valid, it writes the blob to quiz:{today} and updates config:current_date in a single atomic transaction.
Simultaneously, **Users** are polling the API. Their requests read config:current_date. Before 00:00, they get "Yesterday's Quiz". The moment the atomic transaction completes, the next poll returns "Today's Quiz". The transition is instant for all users globally.

3.2 The Singleton Generation Pattern

This is the central pillar of our "Efficiency Strategy."
Comparison of Resource Load:

Metric	On-Demand Generation (Naive)	Singleton Generation (Optimized)	Impact
Trigger	User Action (Page Load)	System Clock (Cron)	**Decoupled from Traffic**
Requests (10 Users)	10 API Calls	0 API Calls (Cached)	100% Savings
Requests (10k Users)	10,000 API Calls	1 API Call (Cached)	**Prevents Quota Death**
Latency	2s - 15s (Wait for AI)	< 50ms (Read from Redis)	**Instant UX**
Cost	High (Potential Overage/Blocks)	Zero (Free Tier)	Sustainable

The Singleton pattern transforms the variable cost of AI generation into a fixed, negligible cost, regardless of how viral the game becomes.

3.3 Data Schema Design (Density Optimization)

To fit 100,000+ users into 500MB, we employ a strict schema.

3.3.1 Quiz Storage (Strings)

- **Key:** quiz:YYYY-MM-DD
- **Value:** JSON String (Compressed)
- **Size:** ~2KB per quiz.
- **TTL:** 30 Days.
- **Auto-Cleanup:** Redis EXPIRE command is set on creation. This ensures old data vanishes automatically, keeping storage usage strictly bounded.

3.3.2 User Statistics (Hashes)

Using a Redis Hash allows us to group fields for a single user, reducing the overhead of key names.

- **Key:** u:{user_id} (Shortened key name to save bytes)
- **Type:** Hash

- **Fields:**
  - s (Score - Total): Integer
  - w (Wins): Integer
  - k (Streak): Integer
  - ld (Last Date Played): String (YYMMDD)
- **Optimization:** We use short field names (s instead of score) because Redis stores the field strings for *every* user in some encoding configurations (though ziplist optimization helps). Every byte counts at scale.

3.3.3 Leaderboards (Sorted Sets)

- **Key:** lb:{date}
- **Type:** ZSET
- **Score:** The user's score for that day.
- **Member:** {user_id}
- **Atomicity:** We use ZADD to upsert scores. This handles ranking automatically with O(log(N)) complexity.

---

4. Implementation Specification

4.1 Configuration: devvit.json

The user request referenced "YAML config," but Devvit strictly uses devvit.json. We correct this here. This file is the "permissions manifest" and is the single most common point of failure for external fetch requests.

JSON

```
{
  "name": "duel-of-minds-2026",
  "version": "1.0.0",
  "permissions": {
    "http": {
      "enable": true,
      "domains": [
        "generativelanguage.googleapis.com"
      ]
    },
    "scheduler": true,
    "redis": true,
    "reddit": {
      "enable": true,
```

```
      "asUser":
    }
  },
  "scheduler": {
    "tasks": {
      "generate_daily_quiz": {
        "endpoint": "/internal/cron/generate-quiz",
        "cron": "0 0 * * *"
      },
      "cleanup_old_data": {
        "endpoint": "/internal/cron/cleanup",
        "cron": "0 2 * * 0"
      }
    }
  }
}
```

Critical Details:

- **generativelanguage.googleapis.com**: Must be listed verbatim. No wildcards (*.googleapis.com) are allowed.[3]
- **Cron Schedule**: 0 0 * * * targets midnight UTC. The cleanup job runs weekly (* * 0 - Sunday) at 2 AM to avoid conflict with the generation job.

4.2 The LLM Service (TypeScript)

This service manages the "30-second wall" and "Rate Limit" risks.

Key Features:

1. **Strict JSON Enforcement:** Using responseMimeType: "application/json".[13]
2. **Exponential Backoff:** Handling 429 (Rate Limit) and 500 (Server Error).
3. **Type Safety:** Validating the output against a TypeScript interface.

TypeScript

```
import { Devvit } from '@devvit/public-api';

// Define the shape of our data to ensure type safety throughout the app
interface QuizQuestion {
  id: number;
  text: string;
  options: string;
  correctIndex: number;
  explanation: string;
```

```
}

interface DailyQuiz {
  date: string;
  theme: string;
  questions: QuizQuestion;
}

const GEMINI_API_URL =
'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent
';

export class LLMService {
  constructor(private context: Devvit.Context) {}

  /**
   * Generates a quiz for a specific date using Gemini 2.0 Flash.
   * Implements retry logic with exponential backoff to handle rate limits.
   */
  async generateDailyQuiz(dateStr: string): Promise<DailyQuiz | null> {
    const apiKey = await this.context.settings.get('GEMINI_API_KEY');
    if (!apiKey) {
        console.error("Critical: GEMINI_API_KEY is missing in settings.");
        return null;
    }

    // Prompt Engineering: Single-Shot, Strict Schema
    const prompt = `
      You are a game engine for "Duel of Minds".
      Task: Generate a daily knowledge quiz for ${dateStr}.
      Constraints:
      1. Theme: Randomly select from History, Science, Literature, or Pop Culture.
      2. Difficulty: 5 questions, starting easy, ending hard.
      3. Format: Return ONLY raw JSON. No markdown formatting.
      4. Content Safety: No political, religious, or NSFW topics.

      JSON Schema:
      {
        "date": "${dateStr}",
        "theme": "string",
        "questions": [
          { "id": number, "text": "string", "options": ["string", "string", "string", "string"],
"correctIndex": number (0-3), "explanation": "string" }
```

```
      ]
    }
  `;

  const maxRetries = 3;
  let attempt = 0;

  while (attempt < maxRetries) {
    try {
      console.log(` Attempt ${attempt + 1} for ${dateStr}`);

      const response = await fetch(`${GEMINI_API_URL}?key=${apiKey}`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          contents: [{ parts: [{ text: prompt }] }],
          generationConfig: {
            responseMimeType: "application/json", // Enforce JSON mode
            temperature: 0.7 // Balanced creativity/determinism
          }
        })
      });

      // Handle Non-200 Responses
      if (!response.ok) {
        const status = response.status;
        console.warn(` Gemini API returned status ${status}`);

        // If Rate Limited (429) or Server Error (5xx), we throw to trigger retry
        if (status === 429 |

| status >= 500) {
          throw new Error(`Transient Error ${status}`);
        }
        // If 400/401/403, it's a configuration error. Do not retry.
        return null;
      }

      const data = await response.json();
      const rawText = data.candidates?.?.content?.parts?.?.text;

      if (!rawText) throw new Error("Empty response from model");
```

```
      // Sanitization: Remove accidental markdown blocks if the model hallucinates them
      const cleanJson = rawText.replace(/```json/g, '').replace(/```/g, '').trim();

      const quiz = JSON.parse(cleanJson) as DailyQuiz;

      // Basic Validation: Ensure we actually got 5 questions
      if (!quiz.questions |

| quiz.questions.length!== 5) {
        throw new Error("Invalid quiz structure generated");
      }

      return quiz;

    } catch (e) {
      attempt++;
      if (attempt >= maxRetries) {
        console.error(` Failed after ${maxRetries} attempts: ${e}`);
        return null;
      }

      // Exponential Backoff: 1s, 2s, 4s
      // We cap wait time to ensure we don't hit the 30s Devvit execution limit
      const backoffTime = Math.min(Math.pow(2, attempt) * 1000, 5000);
      await new Promise(resolve => setTimeout(resolve, backoffTime));
    }
  }
  return null;
 }
}
```

4.3 The Scheduler Job (The Main Loop)

This script acts as the conductor. It orchestrates the "Midnight Handover" process detailed in our design phase.

Key Logic:

1. **Idempotency:** Checks if the quiz already exists to prevent wasted API calls.
2. **Fallback Strategy:** If generation fails (Gemini outage), it loads a "Generic Backup" to ensure the game continues.
3. **Atomic Commit:** Updates the quiz data and the current date pointer simultaneously.

TypeScript

```
// server/scheduler-job.ts
import { Devvit } from '@devvit/public-api';
import { LLMService } from './llm-service';

export async function generateQuizJob(event: any, context: Devvit.Context) {
  const redis = context.redis;
  const llm = new LLMService(context);

  const today = new Date().toISOString().split('T'); // "2026-02-04"
  const quizKey = `quiz:${today}`;

  // 1. Idempotency Check
  const exists = await redis.get(quizKey);
  if (exists) {
    console.log(` Quiz for ${today} already exists. Terminating.`);
    return;
  }

  // 2. Generation Phase
  console.log(` Starting generation for ${today}...`);
  let quiz = await llm.generateDailyQuiz(today);

  // 3. Fallback Mechanism
  if (!quiz) {
    console.error(` Generation failed. Engaging fallback protocol.`);
    // Ideally, you have a pool of backups: "backup:1", "backup:2"
    // Here we load a generic one.
    const backupStr = await redis.get('quiz:backup:generic');
    if (backupStr) {
       quiz = JSON.parse(backupStr);
       quiz.date = today; // Patch the date to match today
    } else {
       console.error(" FATAL: No backup quiz available.");
       return; // Nothing we can do.
    }
  }

  // 4. Atomic Commit Phase
  // We use WATCH/MULTI/EXEC to ensure consistency
  const configKey = 'config:current_date';

  try {
```

```
    const tx = await redis.watch(configKey);
    await tx.multi();

    // Store the quiz
    await tx.set(quizKey, JSON.stringify(quiz));

    // Update the pointer so clients see the new quiz
    await tx.set(configKey, today);

    // Set TTL for auto-cleanup (30 days)
    await tx.expire(quizKey, 60 * 60 * 24 * 30);

    await tx.exec();
    console.log(` Successfully committed quiz for ${today}.`);

  } catch (err) {
    console.error(` Transaction failed: ${err}`);
    // In a real app, you might schedule a retry job here
  }
}
```

4.4 The API Layer (BFF Router)

This component serves the data to the client. It handles the "Sanitization" of the quiz data to prevent cheating.

TypeScript

```
// server/main.ts (Fragment)

// Endpoint: Client fetches the daily quiz
context.router.get('/api/daily-quiz', async (req, res) => {
  const redis = context.redis;

  // Determine which quiz is "Live"
  const liveDate = await redis.get('config:current_date');
  if (!liveDate) {
    return res.status(503).send("System initializing...");
  }

  const quizDataStr = await redis.get(`quiz:${liveDate}`);
  if (!quizDataStr) {
```

```
    return res.status(404).send("Quiz data missing.");
  }

  const fullQuiz = JSON.parse(quizDataStr);

  // SECURITY: Sanitize the payload
  // We strip out the 'correctIndex' and 'explanation' fields
  // so the user cannot see the answers in their Network tab.
  const clientSafeQuiz = {
    date: fullQuiz.date,
    theme: fullQuiz.theme,
    questions: fullQuiz.questions.map((q: any) => ({
      id: q.id,
      text: q.text,
      options: q.options
      // correctIndex removed
      // explanation removed
    }))
  };

  return res.status(200).json(clientSafeQuiz);
});

// Endpoint: Client submits their answers
context.router.post('/api/submit-score', async (req, res) => {
  // Logic:
  // 1. Fetch full quiz from Redis (including answers).
  // 2. Compare user answers.
  // 3. Calculate score.
  // 4. Update User Stats (HSET) and Leaderboard (ZADD).
  // 5. Return result + rank.
});
```

---

5. Efficiency and Optimization Strategy

5.1 The "Midnight Handover" Sequence

One of the most complex aspects of a daily game is the transition at 00:00 UTC. If a user loads the game at 23:59:59 and submits at 00:00:05, what happens?

- **Problem:** The user played Quiz A, but the global pointer now points to Quiz B. Submitting answers for A against the key for B would result in a score of 0 and a

frustrated user.

- **Solution:** The client must send the date ID of the quiz they played (2026-02-03) along with their answers. The server endpoint /api/submit-score uses this ID to fetch the specific quiz key quiz:2026-02-03 for validation, rather than blindly using the "current" quiz.
- **Visualizing the Handover:**
    - **T-minus 10s:** Scheduler wakes up.
    - **T-minus 5s:** Gemini generates content.
    - **T-0s:** Atomic Redis transaction updates config:current_date.
    - **T+1s:** Next user fetch gets the new quiz. Old users can still submit old quiz scores because quiz:yesterday still exists (due to 30-day TTL).

5.2 Failure Mode Analysis: The "Circuit Breaker"

Despite our exponential backoff, Gemini 2.0 Flash might experience a prolonged outage.

- **Detection:** If the LLMService fails 5 times consecutively, we can flag a system health key in Redis: system:gemini_health = DOWN.
- **Action:** Subsequent scheduler runs check this key. If DOWN, they skip the HTTP call entirely and immediately load a pre-cached generic quiz.
- **Recovery:** A separate "Janitor" job or manual admin action can reset the health key to UP once the service is stable.

5.3 Scalability Projections

Can this architecture handle 1 million users?

- **Gemini Load:** 0 increase. The Singleton pattern keeps this flat.
- **Redis Reads:** 1 million users fetching the quiz = 1 million Redis GET operations. Redis can handle ~100k ops/sec. Spread over a day, this is trivial. The bottleneck would be the "Midnight Spike" (everyone checking at once).
- **Redis Writes:** 1 million score submissions. The atomic ZADD is fast, but heavy concurrency might cause WATCH collisions.
    - **Mitigation:** For extreme scale (Tier-1 Enterprise), we would shard the leaderboards (e.g., leaderboard:2026-02-04:shard1, shard2) and aggregate them, but for a Hackathon, the standard Optimistic Locking is sufficient for ~10k concurrent users.

---

6. Conclusion and Future Outlook

The "Duel of Minds" architecture demonstrates that the constraints of the Devvit platform—its statelessness, execution timeouts, and storage limits—are not barriers to innovation but guiderails for better engineering. By strictly adhering to the **Singleton Generation Pattern**, we neutralize the risk of the Gemini Free Tier limits. By implementing a **BFF Sanitization Layer**, we secure the integrity of the game against client-side tampering. And by utilizing

Redis Primitives effectively, we achieve a data density that allows for massive scalability within a modest memory footprint.
This report fulfills the mandate of the 2026 Hackathon: it presents a solution that is zero-infra, operationally resilient, and architecturally sound. It transforms the ephemeral "serverless" function into a persistent, living world for the players.

Key Takeaways for the Implementation Team

1. **Strictly Verify devvit.json:** Ensure generativelanguage.googleapis.com is present.
2. **Trust the Singleton:** Do not give in to the temptation of generating custom quizzes per user; it is the path to rate-limit death.
3. **Atomic Everything:** Never write to Redis without WATCH if the data is shared.
4. **Prompt Once:** Single-shot JSON prompts are the only way to survive the 30-second timeout.

Works cited

1. HTTP Fetch - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/capabilities/server/http-fetch
2. Global fetch allowlist - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/0.11/capabilities/http-fetch-allowlist
3. Overview - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/0.11/capabilities/http-fetch
4. Set task timeout for jobs | Cloud Run, accessed February 3, 2026, https://docs.cloud.google.com/run/docs/configuring/task-timeout
5. Configure Your App - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/capabilities/devvit-web/devvit_web_configuration
6. Scheduler - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/capabilities/server/scheduler
7. Job scheduler | dbt Developer Hub, accessed February 3, 2026, https://docs.getdbt.com/docs/deploy/job-scheduler
8. Gemini API Free Tier Limits 2025: Complete Guide to Rate Limits, 429 Errors & Solutions, accessed February 3, 2026, https://www.aifreeapi.com/en/posts/gemini-api-free-tier-limit
9. Gemini API Rate Limits 2026: Complete Per-Tier Guide with All Models, accessed February 3, 2026, https://www.aifreeapi.com/en/posts/gemini-api-rate-limits-per-tier
10. Bypassing Gemini API Rate Limits with Smart Key Rotation in Next.js | by Entekume jeffrey, accessed February 3, 2026, https://medium.com/@entekumejeffrey/bypassing-gemini-api-rate-limits-with-smart-key-rotation-in-next-js-8acdee9f9550
11. Redis - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/capabilities/server/redis
12. Redis - Reddit for Developers, accessed February 3, 2026,

https://developers.reddit.com/docs/0.11/capabilities/redis
13. Structured outputs | Gemini API - Google AI for Developers, accessed February 3, 2026, https://ai.google.dev/gemini-api/docs/structured-output