PRODUCT REQUIREMENTS DOCUMENT (PRD)

Project Name: Campus Veritas

Version: 3.0 (Final Hackathon Build)

---

1. Executive Summary

Campus Veritas is a decentralized, anonymous rumor verification platform. It solves the "Fake News" problem using Game Theory rather than central moderation.

Core Philosophy:

1. **Identity Air-Gap:** Users verify as students but act as anonymous, unlinkable tokens.
2. **Weighted Truth:** One "Senior" vote is worth more than five "Freshman" votes.
3. **Bot Resistance:** UI elements physically evade automated clicks; logical propagation delays prevent coordination attacks.

---

2. Technical Architecture (The "Air Gap")

2.1 Technology Stack

- **Frontend:** Next.js 14 (App Router), Tailwind CSS, **Framer Motion** (Required for Randomization).
- **Backend:** Node.js, Express.js.
- **Database:** SQLite (File: veritas.db), managed via **Prisma ORM**.
- **Auth:** Custom "Blind Token" Protocol.

2.2 System Services

The system is divided into two mutually exclusive logic flows:

1. **Identity Service (The Gatekeeper):**
    - Accepts email + OTP.
    - Checks if isClaimed.
    - Returns a **UUID**.
    - *Constraint:* MUST NOT store the UUID alongside the email.
2. **Activity Service (The Ledger):**
    - Accepts UUID for actions (Vote/Post).
    - Calculates TrustScore based on the UUID's reputation.
    - Has **Zero Knowledge** of the user's real name/email.

---

3. Database Schema (Prisma)

CRITICAL RULE: The User table must NEVER contain an email field.

```
Code snippet
// 1. IDENTITY LEDGER (The Gatekeeper)
model VerificationLog {
  email       String   @id
  otpCode     String
  isClaimed   Boolean  @default(false) // Prevents double-claiming
  verifiedAt  DateTime?
}

// 2. ACTIVITY LEDGER (The Anonymous World)
model User {
  id          String   @id @default(uuid()) // The "Blind Token"
  reputation  Float    @default(45.0)       // STARTING REP (0-100 Range) [cite: 1]
  joinedAt    DateTime @default(now())

  votes       Vote[]
  rumors      Rumor[]
}

model Rumor {
  id             String   @id @default(uuid())
  content        String
  authorId       String
  trustScore     Float    @default(0.0)

  // Propagation Logic
  status         String   @default("TRIBUNAL") // TRIBUNAL -> PUBLIC -> SETTLED/REJECTED

  // Timing
  createdAt      DateTime @default(now())
  visibleAt      DateTime // Jitter Delay (Now + Random 1-60 mins) [cite: 3]
  settlesAt      DateTime // Hard Cap (Now + 7 Days) [cite: 3]

  author         User     @relation(fields: [authorId], references: [id])
  votes          Vote[]
}

model Vote {
  id       String   @id @default(uuid())
  userId   String
```

```
  rumorId   String
  type      Int     // 1 (Verify) or -1 (Dispute)
  weight    Float   // The power of this vote (Rep * 0.02) [cite: 1]

  user      User    @relation(fields: [userId], references: [id])
  rumor     Rumor   @relation(fields: [rumorId], references: [id])

  @@unique([userId, rumorId])
}
```

---

4. Functional Logic (The "Veritas Protocol")

4.1 The "Blind Token" Handoff

- **Endpoint:** POST /api/auth/verify
- **Logic:**
    1. Receive email and otp.
    2. Check VerificationLog: If email exists & OTP matches & !isClaimed.
    3. Set isClaimed = true.
    4. Ask for previous UUID, if new user then he can skip.
    5. **GENERATE** a new UUID (user_session_token).
    6. If new user **CREATE** a new User entry with that UUID. otherwise replace the previous UUID with new one to preserve previous info
    7. **RETURN** the new UUID to client.
    8. **FORGET** the connection. Do not log "Email X got UUID Y".

4.2 Game Theory: Scoring & Costs

- **Vote Power Formula:**
  $$Power = Reputation \times 0.02$$
    - *Example:* 50 Rep = 1.0 Power. 75 Rep = 1.5 Power.
- **Posting Costs:**
    - **High Trust (>60 Rep):** Deduct **5 Points**.
    - **Low Trust (<60 Rep):** Deduct **10 Points**.
- **Settlement Rewards:**
    - **Consensus:** +5 Points (If you voted with the winner).
    - **Slashing:** -15 Points (If you voted against the winner).

4.3 Propagation: The "Tribunal" Pipeline

- **Step 1: Jitter Delay (Anti-Coordination)**
    - visibleAt = Now + Random(60,000ms to 3,600,000ms) (1 min to 60 mins).
    - Rumors are hidden from *everyone* until visibleAt.
- **Step 2: Tribunal Mode (Seniors Only)**

    - **Duration:** First **2 Hours** after visibleAt.
    - **Visibility:** Only accessible to users with Reputation > 75.
    - **The Kill Switch:** If **40%** of viewing Seniors downvote it, the rumor is **REJECTED** immediately.
- **Step 3: Public Wave**
    - After 2 hours, it opens to all users if not rejected.

4.4 Settlement Rules

A rumor is finalized (points awarded/slashed) when EITHER:

1. **Time Limit:** 7 Days have passed.
2. **Vote Cap:** The rumor hits the Kill Switch Votes in Tribunal Mode.

4.5 UI Bot Defense

- **Requirement:** "Verify" and "Dispute" buttons must move on every render.

Implementation:
JavaScript
// Random offset between -25px and +25px
const x = Math.random() * 50 - 25;
const y = Math.random() * 50 - 25;
<motion.button animate={{ x, y }} transition={{ duration: 2 }} />
-

---

5. API Endpoints Specification

Method	Endpoint	Description	Constraints
POST	/auth/verify	Exchanges Email+OTP for UUID.	**No Logging Link!**
GET	/feed	Fetches active rumors.	Filter WHERE visibleAt < NOW. If user Rep < 80, filter WHERE createdAt < NOW - 2h.
POST	/rumor	Creates a new rumor.	Deduct Cost (-5 or -10). Set visibleAt (1-60m jitter).

POST	/vote	Casts a weighted vote.	Weight = User.rep * 0.02.
CRON	(Internal)	Settlement Worker.	Runs every minute. Checks for votes.length >= 50 OR settlesAt < NOW.

---

6. Constants (Hardcoded Values)

JavaScript

```
// Copy this into server/constants.js
export const REP_INITIAL = 50.0;
export const REP_MAX = 100.0;
export const REP_SENIOR = 80.0; // Threshold for Tribunal Access

export const VOTE_MULTIPLIER = 0.02; // 50 Rep * 0.02 = 1.0 Power [cite: 1]

export const COST_POST_HIGH = 5;
export const COST_POST_LOW = 10; // [cite: 1]
export const REWARD_CONSENSUS = 5; // [cite: 2]
export const PENALTY_SLASH = 15; // [cite: 2]

export const JITTER_MIN_MS = 60 * 1000;       // 1 Minute [cite: 3]
export const JITTER_MAX_MS = 60 * 60 * 1000;  // 60 Minutes [cite: 3]

export const SETTLEMENT_TIME_MS = 7 * 24 * 60 * 60 * 1000; // 7 Days [cite: 3]
export const SETTLEMENT_VOTE_CAP = 50; // [cite: 5]

export const TRIBUNAL_DURATION_MS = 2 * 60 * 60 * 1000; // 2 Hours [cite: 5]
export const TRIBUNAL_REJECTION_RATE = 0.4; // 40% Dislikes kills the rumor [cite: 5]
```

7. Implementation Phases (For Agent Execution)

Phase 1: The Core (Backend)

1. Initialize Express + Prisma.
2. Implement the schema.prisma exactly as defined above.
3. Create the "Blind Token" Auth Controller.
4. Write the "Jitter" logic in the Rumor creation endpoint.

Phase 2: The Interface (Frontend)

1. Scaffold Next.js.
2. Create the "Login Screen" (Email input).
3. Create the "Feed" (Cards with hidden scores).
4. Implement Framer Motion for the wiggling buttons.

Phase 3: The Brain (Game Theory)

1. Implement the recalculateReputation() function.
2. Implement the "Tribunal" middleware (Filter feed based on req.user.reputation).