PRD — Indian History Quiz (1900–1947) — Streamlit MVP

Version: 1.0
Author: Product Manager / You
Date: 2025-09-14

---

1. Executive summary

Build a lightweight Streamlit quiz app focused on **Indian history from 1900 to 1947**. The MVP supports both **single-player** and **multiplayer (competition + leaderboards)** experiences, multiple question formats (MCQ, True/False, Fill-in-the-blank, Matching, Timed/Speed rounds), a layered scoring system (base points + time bonus + streaks + difficulty multiplier), and light gamification (progress bar, badges, avatars). Quiz content is **static, preloaded** for MVP.

The app is anonymous (temporary usernames) to reduce friction. Leaderboards (global + friends) are supported via a simple persistent store (SQLite or JSON for MVP), with an option to wire up an external DB for production.

---

2. Goals & success metrics

Primary goals

- Deliver an engaging, educational quiz experience about Indian history (1900–1947).
- Enable easy, repeatable play with social competition via leaderboards.
- Provide meaningful feedback (explanations) after questions.

Success metrics (MVP)

- **# of unique players** (first 30 days)
- **Average session length** (questions answered per session)
- **Completion rate** (sessions started → sessions completed)
- **Leaderboard submissions** (number of scores posted to global leaderboard)
- **Average score improvement** (compare first vs. third session)

Concrete numeric targets can be set later once analytics are enabled.

---

3. Scope

In-scope (MVP)

- Streamlit web app (single-file or small module structure).
- Static, preloaded question bank (JSON or Python list), covering 1900–1947.
- Single-player mode: fixed quizzes, timed sessions, and custom quizzes.
- Multiplayer/competition via **challenge codes** (asynchronous) + global leaderboard.
- Anonymous temporary usernames (user chooses a display name each session).
- Scoring: base points + time bonus + streak bonus + difficulty multiplier.
- Post-question explanation + correct/incorrect display.
- Gamified elements: progress bar, badges (unlockable), avatar selection (emoji-based).
- Simple persistent leaderboard storage (SQLite or JSON file).
- Step-by-step deployment instructions (local + Streamlit Cloud + Hugging Face Spaces).

Out of scope (MVP)

- User authentication (OAuth / email verification).
- Real-time multiplayer (sockets).
- Admin UI for question editing (no dynamic backend for Q/A).
- Advanced animations, sounds, or heavy graphics.
- Multi-language/localization.

---

4. User personas

1. **High-school student (Age 15–19)** — studying history, uses app to revise and test knowledge before exams.
2. **History enthusiast (Age 25–45)** — casual player who loves historical trivia and competition.
3. **Teacher/quizmaster (Age 30–55)** — uses app to craft small in-class quizzes and share challenge codes.

---

5. User stories

- As a casual player, I want to start a 10-question quiz so I can quickly test my knowledge.
- As a time-challenged user, I want a 5-minute timed session to see how many I can answer.
- As a user, I want explanations after each answer so I can learn from mistakes.
- As a player, I want to post my score to a global leaderboard to compete.
- As a group, I want to generate a challenge code to share with friends and compare scores in a friends leaderboard.
- As a player, I want badges for achievements to feel rewarded.

---

6. Functional requirements

6.1 Core quiz engine

- Support question types: `mcq`, `true_false`, `fill_blank`, `matching`.
- Randomize question order per session.
- Randomize options for MCQs.
- Support question-level `difficulty` (easy/medium/hard).
- Show a per-question timer (configurable); for timed sessions, apply session timer.
- Store user session info in `st.session_state` for continuity while session active.

6.2 Session types

- **Fixed quiz**: user chooses N questions (default: 10).
- **Timed session**: user chooses time limit (default: 5 minutes), answer as many as possible.
- **Custom**: user selects number of questions + difficulty mix.

6.3 Scoring

- Compute per-question points using: base points (by question type) × difficulty multiplier + time bonus + streak bonus.
- Only award time bonus for correct answers.
- Store final session score and relevant metadata (mode, time taken, correct_count).

(See scoring algorithm section for precise formulas.)

6.4 Feedback & learning

- After answering: show Correct / Incorrect; show short explanation (1–3 sentences) drawn from question data.
- Provide end-of-quiz summary: total score, breakdown by difficulty, accuracy %, time per question.

6.5 Leaderboards & multiplayer

- **Global leaderboard** (all submissions stored and ranked by score): show top 50.
- **Friends leaderboard** implemented via **challenge codes**: creator selects quiz settings → app generates short alphanumeric `challenge_code` (e.g., `AF3Z9`) → others enter code to take exact same quiz → scores tagged with challenge → friends leaderboard shows top scores for that challenge.
- Players can choose to **publish** or **skip** leaderboard submission at the end (privacy by default).

6.6 Gamification

- **Progress bar** during quiz (questions completed / total).
- **Badges**: first quiz, high scorer, streak master, quick thinker. Badges are unlocked and shown on end-of-quiz screen.
- **Avatar selection**: small set of emoji-based avatars to personalize temporary usernames.

6.7 Persistence & Data

- Store leaderboards and challenge metadata in a lightweight DB (SQLite recommended) or `data/leaderboard.json` for simplest MVP.
- Questions stored in `data/questions.json` (static file) and loaded on app start.

6.8 Admin & content

- For MVP: editing is manual (edit JSON or Python). Add instructions in README for updating question bank.

---

7. Non-functional requirements

- **Performance**: < 500ms question load time on typical broadband.
- **Availability**: app should be accessible (Streamlit Cloud or HF Spaces) and responsive.
- **Security**: do not store PII; no authentication in MVP. Avoid storing client IPs or other personal metadata.
- **Scalability**: MVP assumes small traffic. If high traffic expected, move DB to managed service (Supabase/Postgres).
- **Accessibility**: keyboard navigable, readable font sizes, color contrast.

---

8. Data & question bank schema

questions.json (array of objects). Example schema:

```
{
  "id": "q0001",
  "year": 1919,
  "topic": "Jallianwala Bagh",
  "type": "mcq",
  "difficulty": "medium",
  "question": "On which date did the Jallianwala Bagh massacre take place?",
  "options": ["13 April 1919", "15 August 1919", "30 January 1919", "26 January 1920"],
  "answer": "13 April 1919",
  "explanation": "The Jallianwala Bagh massacre occurred on 13 April 1919, when British troops fired on a peaceful gathering in Amritsar."
}
```

Supported `type` values and expected fields: - `mcq`: `options` (array), `answer` (string) - `true_false`: `answer` (`true` or `false`) - `fill_blank`: `answer` (string or array of synonyms) - `matching`: `pairs` (array of `{left, right}`), app will shuffle left/right and present matching UI

Sample question excerpts (MVP sample bank)

- MCQ: *When did Mahatma Gandhi return to India from South Africa?* → `1915` (Explanation: Gandhi returned in 1915.)
- True/False: *The Simon Commission included any Indian members.* → `False` (Explanation: it had no Indian members.)
- Fill-in-the-blank: *The Non-Cooperation Movement began in ____.* → `1920` (Explanation: launched after the Khilafat resolutions and events of 1920.)
- Matching: *Match leader → movement* (e.g., Gandhi → Non-Cooperation Movement; Subhas Chandra Bose → INA involvement)

(Provide ~100–200 curated items for launch if possible; 30–50 is reasonable MVP.)

---

9. Scoring algorithm (precise)

Parameters

- `type_base` (base per question type): MCQ=10, TRUE_FALSE=5, FILL_BLANK=15, MATCHING=20.
- `difficulty_multiplier`: EASY=1.0, MEDIUM=1.5, HARD=2.0.
- `time_limit` = per-question time in seconds (configurable). For timed sessions, `time_limit` is derived from session. For fixed quizzes, default per-question limit = 30s.
- `time_bonus_cap` = 50% of base_score (configurable) — maximum time bonus possible.
- `streak_bonus_unit` = 2 points per consecutive correct answer (scaled by difficulty multiplier).

Formula (for a correct answer)

1. `base_score = type_base * difficulty_multiplier`
2. `time_bonus = round((max(0, time_limit - time_taken) / time_limit) * time_bonus_cap)`
3. `streak_bonus = round(streak_bonus_unit * (current_streak - 1) * difficulty_multiplier)` (0 if streak=1)
4. `question_score = round(base_score + time_bonus + streak_bonus)`

For incorrect answers: `question_score = 0` (optionally show small penalty in future versions).

Example: MCQ (type_base=10), difficulty=HARD (2.0) → base_score=20. If time_limit=30s, time_taken=10s → time_bonus = ((30-10)/30)*10 = 6.67 → 7. If streak is 3 -> streak_bonus = 2*(3-1)*2 = 8. Total ≈ 20+7+8 = 35 points.

---

10. Leaderboards & multiplayer design details

Global leaderboard

- Table columns: `id`, `username`, `avatar`, `mode`, `score`, `correct_count`, `questions_count`, `duration_seconds`, `quiz_settings_hash`, `created_at`.

- Display top 50 by `score` + filters by mode (fixed/timed/custom) and timeframe (today/this week/all time).

Friends leaderboard (Challenge code)

- Creating a challenge generates an entry in `challenges` table: `code`, `quiz_settings`, `created_by`, `expires_at`, `created_at`.
- Players enter `code` to take exact same quiz (deterministic question ordering via seed stored in `quiz_settings`), ensuring fairness.
- Scores submitted with `challenge_code` populate challenge-specific leaderboard.
- Challenge codes expire after configurable period (e.g., 7 days).

Anti-cheat & fairness

- Use server-side scoring; never trust client-side score.
- Fix random seed per challenge to guarantee identical question/option ordering for all players of that challenge.
- Limit rapid repeat submissions by same username (throttling) if needed.

---

11. UX / UI requirements (Streamlit mapping)

Global structure (pages)

- **Home**: choose mode (Single-player, Timed, Custom, Multiplayer → Create/Join Challenge), quick start.
- **Play**: question/answer UI, timer, progress bar, avatar & username at top.
- **Results**: per-question summary, score breakdown, badges unlocked, `Publish score` button.
- **Leaderboards**: global + friend challenges.
- **About / How to play**: instructions, scoring explanation, data/privacy note.

Key components

- Question card: `st.markdown()` for text, `st.radio()` / custom UI for options, `st.text_input()` for fill-in, 2-column layout for matching.
- Timer: visible countdown using `st.empty()` with periodic refresh; use `time.time()` and `st.session_state` to compute remaining time.
- Progress: `st.progress()` to show completion.
- Avatar selection: set of emoji buttons.
- Badges: display icons + short description.

---

12. Data model (MVP SQL)

Tables (SQLite)

- `questions` (static - optional persistence): `id TEXT PRIMARY KEY, payload JSON` (optional)

- `leaderboard`:
- `id INTEGER PRIMARY KEY AUTOINCREMENT`
- `username TEXT`
- `avatar TEXT`
- `mode TEXT`
- `score INTEGER`
- `correct_count INTEGER`
- `questions_count INTEGER`
- `duration_seconds INTEGER`
- `challenge_code TEXT`
- `created_at TEXT`

- `challenges`:

- `code TEXT PRIMARY KEY`
- `quiz_settings JSON`
- `seed INTEGER`
- `created_by TEXT`
- `created_at TEXT`
- `expires_at TEXT`

---

13. Testing & QA

- **Unit tests**: scoring logic (edge cases), question parsing, DB write/read.
- **Manual QA**: iterate through all modes (fixed/timed/custom/challenge), confirm scoring matches formula, check leaderboard entries.
- **Security QA**: ensure no PII, verify user input sanitized.

---

14. Acceptance criteria (examples)

- User can start a 10-question quiz and finish it; final score is computed and displayed.
- Correct/Incorrect + explanation shown after each answer.
- Leaderboard can accept a published score and display top entries.
- Challenge code flow works: creator produces code, second user enters code and takes same quiz, both scores show on challenge leaderboard.
- Badges appear when earned.

---

15. Sample question bank (MVP — 12 examples)

Below are a few representative questions that you can expand. All are within 1900–1947.

1. **MCQ** — *When did the Jallianwala Bagh massacre occur?*
   Options: `13 April 1919`, `15 August 1919`, `30 January 1919`, `26 January 1920`
   Answer: `13 April 1919`
   Explanation: `The Jallianwala Bagh massacre happened on 13 April 1919 when troops fired on a peaceful gathering in Amritsar.`

2. **MCQ** — *Who launched the Non-Cooperation Movement in 1920?*
   Options: `Mahatma Gandhi`, `Subhas Chandra Bose`, `Jawaharlal Nehru`, `Sardar Patel`
   Answer: `Mahatma Gandhi`
   Explanation: `Gandhi launched the Non-Cooperation Movement in 1920 in response to events after World War I.`

3. **TRUE_FALSE** — *The Simon Commission included Indian members.*
   Answer: `False`
   Explanation: `The Simon Commission of 1928 had no Indian members, provoking country-wide protests.`

4. **FILL_BLANK** — *The Salt March (Dandi March) began in the year ____.*
   Answer: `1930`
   Explanation: `Gandhi's Salt March to Dandi was in 1930 as civil disobedience against the salt tax.`

5. **MCQ** — *Which movement was launched by the Indian National Congress in 1942 demanding an end to British rule?*
   Options: `Quit India Movement`, `Civil Disobedience Movement`, `Non-Cooperation Movement`, `Swadeshi Movement`
   Answer: `Quit India Movement`
   Explanation: `The Quit India Movement was launched in August 1942 demanding immediate British withdrawal.`

6. **MATCHING** — *Match the leader to an associated event/movement.*
   Pairs: `Gandhi -> Salt March`, `Subhas Chandra Bose -> Indian National Army (INA)`, `Bhagat Singh -> Revolutionary activities`
   Explanation: `These pairings reflect prominent associations in the independence struggle.`

7. **MCQ** — *In which year did the Rowlatt Act result in widespread protests?*
   Options: `1919`, `1922`, `1908`, `1930`
   Answer: `1919`
   Explanation: `The Rowlatt Act provoked protests and ultimately led to harsh responses like the Jallianwala Bagh incident.`

8. **FILL_BLANK** — *India gained independence in the year ____.*
   Answer: `1947`
   Explanation: `India became independent on 15 August 1947 (independence year = 1947).`

(Expand to 50–200 questions for a richer MVP experience.)

---

16. Tech stack & project structure (MVP)

Core - Python 3.8+ - Streamlit - SQLite (builtin `sqlite3`) or `tinydb`/JSON for simplest approach - `pandas` (optional, for analytics)

Dev tooling - `pytest` for unit tests - `black` / `ruff` for formatting/linting

Repository layout (recommended)

```
streamlit-india-quiz/
├── app.py                  # Main Streamlit app
├── quiz/
│   ├── engine.py         # Core quiz logic: scoring, question selection
│   ├── data.py           # helpers to load question bank
│   └── ui.py             # UI helper components
├── data/
│   ├── questions.json    # static question bank
│   └── leaderboard.db    # (created at runtime) sqlite DB
├── requirements.txt
├── README.md
└── tests/
    └── test_scoring.py
```

---

17. Deployment & step-by-step instructions

17.1 Local run (simplest)

1. Install Python 3.8+.
2. Create virtual environment and activate:

```
python3 -m venv venv
# macOS/Linux
source venv/bin/activate
```

```
# Windows
venv\Scripts\activate
```

1. Install dependencies:

```
pip install --upgrade pip
pip install streamlit pandas
# If using tinydb or pytest:
# pip install tinydb pytest
```

1. Project files: ensure `app.py` and `data/questions.json` exist.
2. Initialize DB (optional): run a tiny script to create `data/leaderboard.db` or let app create file on first write.
3. Run the app:

```
streamlit run app.py
```

1. Open `http://localhost:8501` in your browser.

17.2 Deploy to Streamlit Cloud (recommended for easy public deploy)

1. Push project to a GitHub repository (include `requirements.txt`).
2. Create a Streamlit Cloud account and connect your GitHub.
3. Create a new app in Streamlit Cloud, point to your repo and the `app.py` file.
4. Configure secrets if you use an external DB (Supabase/Postgres) via Streamlit's secrets manager.
5. Deploy — Streamlit Cloud will install requirements and host the app.

Notes: For persistent leaderboards across deploys, use an external DB (Supabase/Postgres) and add connection secrets in Streamlit Cloud. Streamlit Cloud's ephemeral file system is not ideal for long-term file-based persistence.

17.3 Deploy to Hugging Face Spaces

1. Create a new Space on Hugging Face and choose **Streamlit** as the runtime.
2. Add your repo files (requirements.txt, app.py, data folder). Commit and push.
3. The Space will build and expose a public URL.

Notes: HF Spaces also runs your app but file persistence across builds is limited; use an external DB to persist leaderboards.

---

18. Implementation checklist (recommended development steps)

1. Create repo skeleton + sample questions (12–30 items).
2. Implement core quiz engine (question selection, randomization).
3. Implement UI for single-player fixed quiz.

4. Implement scoring logic & per-question feedback.
5. Implement result summary page + local leaderboard storage.
6. Add timed and custom quiz modes.
7. Implement challenge code flow (friends leaderboard).
8. Add badges & avatar selection.
9. Polish UI, add README + deployment docs.
10. Write tests for scoring and DB operations.

---

19. Future enhancements (post-MVP)

- Authentication (OAuth) + persistent user profiles.
- Real-time multiplayer (WebSockets) and match-making.
- Admin panel for adding/editing questions.
- Richer analytics dashboard for educators.
- Social sharing & integration (WhatsApp, Twitter/X, etc.).

---

20. Appendix

Example `requirements.txt`

```
streamlit>=1.20
pandas
pytest
# optional
tinydb
python-dotenv
```

Example minimal scoring unit test (pytest)

```
from quiz.engine import compute_question_score

def test_mcq_scoring():
    # MCQ, hard, quick answer, streak 3
    score = compute_question_score(type='mcq', difficulty='hard', time_limit=30,
time_taken=10, streak=3)
    assert score > 0
```

Ready to move to the code stage?

This PRD defines the MVP scope, data, UX, scoring, and deployment guidance so you can begin implementing the Streamlit app. If you'd like, I can now:

- Generate a **starter** `app.py` **Streamlit codebase** (complete with scoring, question loading, a sample UI, local leaderboard), or
- Produce the **question bank JSON** with X sample items (select the count), or
- Create the **DB init script** and helper modules.

Tell me which artifact you want next and I will generate the code (Streamlit `app.py` recommended as the next step).