ChatGPT

Veyoyee MVP – Functional Requirements & Action Plan

Overview and Scope

Veyoyee is an **incentivized survey platform** for academic and nonprofit researchers. It enables researchers to design and deploy surveys and reward respondents for participation. Rewards (e.g. cash or token payouts) are offered **"in return for taking your survey"**, which significantly boosts response rates and data quality. (For context, platforms like Prolific emphasize *fair compensation* – e.g. $12/hour minimum – to ensure quality responses.)

Scope (MVP): The initial Veyoyee release will include: basic user authentication and roles (researcher, respondent, admin); a web-based survey builder; respondent-facing survey-taking UI; token-based incentive accounting; and automated payout processing via Stripe. Non-essential features (e.g. advanced analytics, extensive question types, or multi-language support) are deferred to future versions. The goal is a functional prototype in 6–8 weeks that lets a researcher publish a survey, a respondent complete it, and both parties see the results and payments flow end-to-end.

User Roles

- **Researcher:** Registers and logs in, creates/edits surveys, sets target audience and reward per completion, monitors survey progress and results, and manages survey budgets (wallet of tokens/funds).
- **Respondent:** Registers (completes optional profile), browses or is invited to surveys, answers survey questions, and earns tokens/credits. Respondents can track their earned rewards and request payouts to their linked account.
- **Admin:** Oversees the platform. Admin can manage all users, moderate surveys or content (if needed), adjust system-wide settings (e.g. platform fees), and view overall platform metrics. The Admin role ensures compliance (e.g. enforcing payout rules, monitoring for fraud, or pausing problematic surveys).

Functional Requirements

- **User Authentication & KYC:** Users (researchers and respondents) must sign up with email/password or SSO. Researchers may need verified institutional email or Stripe identity verification for payouts. Stripe Connect's onboarding can handle identity (KYC) for both parties (it "minimize[s] compliance… building your own identity verification"). Respondents should verify their identity or payment account before first payout.
- **Survey Creation:** Researchers can create surveys via a form-based builder. Core features include: setting title/description, adding various question types (multiple-choice, text, numeric), defining logic or skip patterns (optional MVP), and specifying total reward per respondent or budget. Surveys start in "Draft" until the researcher publishes them. Researchers can preview surveys before launch.

- **Dashboard & Management (Researcher):** After login, researchers see a dashboard of their surveys (with status, responses received, budget remaining, etc.). They can view individual survey results: e.g. response counts, basic analytics (charts or tables), and export raw data. Budget tracking shows how many tokens have been distributed versus remaining. Researchers can deactivate or delete a survey if needed.
- **Survey Taking (Respondent):** Authenticated respondents see a list of available surveys (with brief descriptions, estimated time, and reward). They can start a survey, answer questions, and submit responses. The system should prevent multiple submissions (unless allowed) and handle partial saves if session drops. After submission, respondents immediately earn the specified token reward.
- **Token Logic & Reward Accounting:** Each completed survey credit is recorded in a **token ledger**. Functional requirements include: assigning tokens to respondents upon valid submission, updating the researcher's budget balance, and capping tokens per user if needed. The platform will likely use an internal "token" or credit system to track incentives, redeemable for real money. For example, 1 token = $1 by default (redeemable via Stripe). Fair pay guidelines (e.g. *at least $12/hour*) should be suggested when researchers set rewards. The system should prevent empty submissions from earning tokens (e.g. require all mandatory questions answered).
- **Payment & Payouts:** Veyoyee uses **Stripe** for all payments. Researchers must fund their surveys (e.g. via a Stripe payment or connected account) and the platform holds these funds. Respondents can cash out earned tokens through Stripe Connect to their bank or debit card. Key flows: onboarding users with Stripe (create connected accounts), charging researchers' cards to fund surveys, and triggering payouts to respondents when they request withdrawal. Stripe Connect is ideal here: it "is the best way to integrate Stripe as a… platform or marketplace… to orchestrate money movement across multiple parties". The system should enforce payout thresholds (e.g. minimum balance before payout) and handle Stripe fees automatically. An admin interface should list all transactions (charges, refunds, payouts) for audit.
- **Admin Functions:** The admin can view all surveys and user accounts. Admin can suspend surveys, ban abusive users, and intervene in disputes (e.g. if a respondent claims non-payment). The system logs admin actions. Optionally, admin can adjust global parameters (like default fees or token values).

Non-Functional Requirements

- **Performance:** The app should load pages quickly (target <2s) and handle expected concurrent use (e.g. dozens of active surveys and hundreds of respondents). Next.js will use server-side rendering (SSR) or static generation for public pages for speed. Supabase/Postgres queries should be efficient (indexed on survey/user IDs). Use caching (CDN or in-app) for static assets and API responses where feasible.
- **Security:** All data in transit is protected by HTTPS/TLS. User passwords are hashed, and sensitive data (like PII or payment info) is encrypted or tokenized. Use Supabase's row-level security to isolate data per user. Prevent common web vulnerabilities (SQL injection, XSS, CSRF) by sanitizing inputs and using Next.js/React best practices. Stripe handles PCI compliance for payment details. Administrative and financial endpoints require strong authentication (possibly 2FA for admin).
- **Privacy:** Comply with data protection norms: only collect necessary personal info, allow users to delete their data, and anonymize survey responses if requested. Researchers should see only aggregated respondent data, not personal identifiers (unless explicitly collected). The platform should support consenting process (e.g. an opt-in message on surveys) and meet any IRB requirements.

- **Scalability:** The stack is cloud-native: Next.js and APIs can run on serverless (Vercel or AWS), and Supabase scales with Postgres. Supabase's tagline "Build in a weekend, scale to millions" shows it's designed for rapid prototypes and high growth. Tailwind CSS keeps front-end lightweight. We should use pagination for large response sets and horizontally scale the database or use read replicas if needed. Stripe itself scales for payments globally.
- **Reliability & Maintainability:** The codebase (Node.js/Next.js) should be modular and documented. Use a version control repository with CI/CD pipelines. Backup the database regularly. Implement error monitoring and logging. Ensure uptime via cloud hosting (e.g. Vercel uptime SLA, Supabase availability).

Technical Architecture

- **Frontend:** Built with **Next.js** (React framework) and styled with **Tailwind CSS**. Next.js provides SSR and static generation for fast performance, and integrates easily with Tailwind ("Tailwind CSS… fully compatible with Next.js"). The UI will be responsive, mobile-first (using Tailwind utility classes), and accessible. Pages include: home/landing (for waitlist/signup), dashboard (React components with charts), survey builder (dynamic form editor), survey taker (React forms), and profile/settings pages.
- **Backend:** Leverage **Supabase** (Postgres) for most backend needs. Supabase provides user Authentication, a Postgres database, and instant RESTful APIs. We will define tables for users, surveys, questions, options, responses, tokens, transactions, etc. For custom logic (e.g. complex transactions or Stripe webhooks), we can use Next.js API routes (Node.js) or Supabase Edge Functions. The choice of Supabase means minimal custom backend code and easy realtime features (e.g. notifications when a survey response arrives).
- **Database Schema (Overview):** Rough tables might include:
- `users` (id, role, email, profile data, StripeAccountID)
- `surveys` (id, researcher_id→users, title, description, reward_per_response, status, created_at)
- `questions` (id, survey_id, type, prompt, options)
- `responses` (id, survey_id, respondent_id→users, submitted_at, answers JSON)
- `tokens` (id, user_id, survey_id, amount, earned_at, redeemed_flag)
- `transactions` (id, type:charge/payout/refund, amount, user_id, stripe_txn_id, timestamp)
- `admin_logs` (id, action, target_id, admin_id, timestamp)
  Foreign keys and indexes ensure fast joins (e.g. on survey_id). Supabase's realtime and auth features map directly to these tables.
- **Stripe Integration:** Use **Stripe Connect** for payments. Each user (researcher or respondent) has or is connected to a Stripe account (via OAuth or API tokens). When a respondent requests payout, the backend invokes Stripe's Payout API. When a researcher funds a survey, the app creates a Stripe charge. All Stripe events (payments, transfers, payouts) will be handled via webhooks to update our DB's `transactions` and token balances. Stripe Connect also handles KYC: it "onboard[s] users… in just three clicks" and supports 100+ countries.
- **Hosting & Deployment:** The Next.js app will be deployed on Vercel or a similar platform for automatic CI/CD. Supabase is managed hosting. Stripe requires webhooks on a public HTTPS endpoint. Domain, SSL, and email services (for notifications) are configured as needed.

UX Considerations

- **Mobile-Responsiveness:** Design UI with a mobile-first approach. Tailwind's utility classes make it easy to adjust layouts for different screen sizes. Key pages (survey browse, take, and dashboards) should be fully usable on phones and tablets.
- **Ease of Use:** The survey builder should be as simple as possible (form fields, drag to reorder if time permits). Use clear labels, inline validation, and progress indicators in surveys. Provide tooltips or help text for unfamiliar concepts (e.g. what is a "token"). The dashboard should present data visually (graphs or counters) for quick insights. Keep the number of steps low – e.g. one-click survey publication, one-click payout request.
- **Onboarding & KYC Flows:** Introduce KYC/verification only when necessary. For example, allow respondents to take surveys before payout, but require identity verification or linking a payment method before cashing out. Stripe Identity flows can verify IDs easily ("Stripe Identity… verify millions of global users… 100 countries"). Use Stripe-hosted verification forms to reduce friction.
- **Accessibility:** Follow WCAG guidelines: proper color contrast, support screen readers, keyboard navigation. Form fields should have labels. Use accessible date/time pickers and avoid CAPTCHAs if possible (or use reCAPTCHA v3) to reduce frustration.
- **Feedback & Errors:** Provide immediate feedback after actions. E.g. "Survey published successfully." Show loading indicators during network calls. Handle error cases gracefully (e.g. failed payments) with clear messages and steps to retry.

Action Plan

Sprint Schedule (6–8 Weeks)

We will use 1-week agile sprints. Each sprint has clear goals and a demoable deliverable.

1. **Week 1 – Setup & Foundation:**
2. Define data model in Supabase; set up authentication (email sign-up, roles).
3. Configure Stripe sandbox and Connect settings.
4. Scaffold Next.js project with Tailwind. Create basic layout and navigation.
5. *Milestone*: Repo initialized, basic auth and homepage working.
6. **Week 2 – User Management & Auth UI:**
7. Implement user registration/login flows, role selection (Researcher or Respondent).
8. Build user profile pages (including Stripe onboarding links).
9. Admin user seed or UI to assign admin role.
10. *Milestone*: Users can register, log in, and see a placeholder dashboard based on role.
11. **Week 3 – Survey Builder (Researcher UI):**
12. Develop survey creation/editing interface: form to add title, questions (MCQ, text), reward amount.
13. Save surveys to DB (Draft state). Allow preview and publish.
14. Design respondents listing: when published, active surveys are queryable.

15. *Milestone*: Researcher can create and publish a basic survey.

16. **Week 4 – Respondent Workflow:**

17. List available surveys for respondents.
18. Implement survey-taking UI: render questions, collect answers, submit to DB.
19. Upon submission, credit tokens to respondent and debit from researcher's budget.
20. Show confirmation and earned balance update.

21. *Milestone*: End-to-end flow: researcher publishes survey, respondent completes it, tokens are awarded.

22. **Week 5 – Payout Integration:**

23. Integrate Stripe Connect on backend: handle Stripe webhooks for charges/payouts.
24. Enable researcher to fund survey (simulate via Stripe charge in dev).
25. Allow respondent to connect payout account (Stripe Connect onboarding) and request payout when balance threshold is met.
26. Implement wallet/balance pages: track earned tokens and transaction history.

27. *Milestone*: Respondent can request payout; system initiates Stripe payout.

28. **Week 6 – Dashboards & Admin:**

29. Researcher dashboard: show survey stats, response count, budget remaining.
30. Admin dashboard: view all surveys, transactions; tools to suspend users/surveys.
31. Basic analytics: graphs of response rates or demographics (if collected).
32. Polishing UI/UX: mobile layout checks, basic styling.

33. *Milestone*: Full end-to-end with dashboards; begin internal QA testing.

34. **Week 7 – Testing & Refinement (Optional):**

35. Fix bugs identified in QA. Add features deferred earlier (e.g. email notifications, better validation).
36. Deploy a "waitlist" landing page or beta signup form for early adopters.
37. Prepare documentation: README, FRD delivered.

38. *Milestone*: MVP feature-complete, ready for beta release.

39. **Week 8 – Buffer & Launch:**

40. Final polishing, performance optimizations, and security review.
41. If scope allows, implement additional nice-to-haves (e.g. answer export CSV, multi-language text).
42. Officially launch MVP or invite closed beta.

Key Milestones

- **Sprint 1 Demo:** Basic auth and project setup.

- **Sprint 3 Demo:** Working survey builder (create/edit/publish).
- **Sprint 4 Demo:** First complete survey cycle (publish → take → reward tokens).
- **Sprint 5 Demo:** Stripe payment flow (funding and payout).
- **MVP Release:** Live waitlist or beta launch, marketing site ready.

Risks & Mitigations

- **Scope Creep:** With only 6–8 weeks and a solo builder, avoid over-engineering features.
  Mitigation: Stick to MVP scope. Use agile priorities (MoSCoW) and cut any non-critical feature.
- **Stripe Integration Complexity:** Payment bugs or delays (e.g. handling webhooks, international payouts).
  Mitigation: Start Stripe integration early (Week 1/2); use Stripe's test environment. Leverage Stripe's documentation and prebuilt UI where possible.
- **KYC/Compliance Issues:** Verifying user identity across countries can be tricky.
  Mitigation: Use Stripe Identity for global KYC (already built into Connect). Only require full KYC for payouts, not for taking surveys.
- **Single Developer Bottleneck:** A solo developer has limited time.
  Mitigation: Use BaaS solutions (Supabase, Stripe) to reduce coding. Reuse templates and libraries. Allocate time weekly to review priorities.
- **Security or Privacy Lapses:** Storing research data entails responsibility (especially with human subjects).
  Mitigation: Follow best practices (encrypt sensitive data, perform a security review). Possibly engage a consultant or use automated security checks. Educate admin on data handling policies.

Sources: The above requirements and plan draw on best practices in survey platforms and web development. Incentivized surveys have been shown to improve response rates. Veyoyee's tech stack (Next.js + Tailwind + Supabase + Stripe) allows rapid development and scalability, while Stripe Connect simplifies payments and KYC. Tailwind's utility-first CSS ensures responsive design. These choices and sprints follow lean/agile principles for an MVP launch [1].

---

[1] Fast-Track MVP Development | 6 Weeks Launch Guide by molfar.io — molfar.io
https://www.molfar.io/blog/startup-time-machine-rapid-development-guide