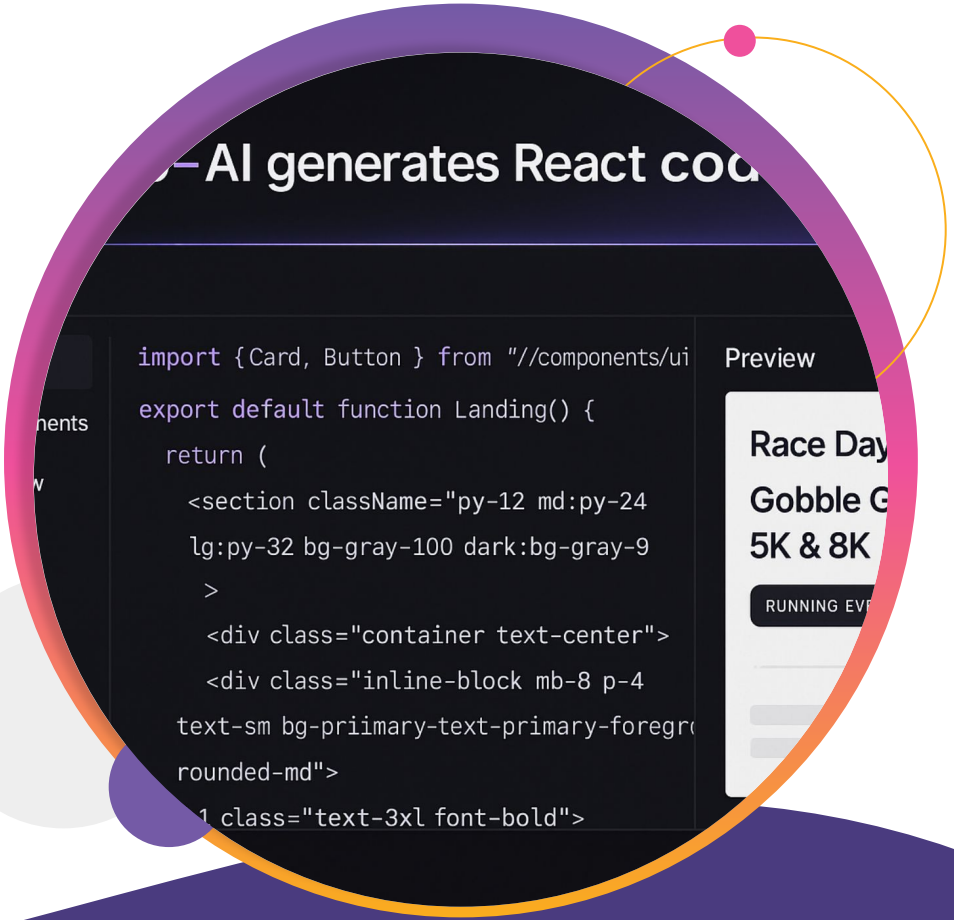
Vibe Coding & Security

RunSignup

Agenda

- Quick recap on vibe coding
- When does security become a concern?
- Secure RunSignup Access
- Q/A



What is Vibe Coding?

Coding by Intent, Not Syntax: A development style where the human provides high-level, natural language instructions ("vibes"), and an AI (LLM) writes the actual code.

The Workflow

- Input: "Make the button blue and bouncy when clicked."
- Process: AI generates HTML/CSS/JS instantly.
- Loop: User tests it -> "The bounce is too slow" -> AI rewrites code -> User approves.

Security Context

- The Black Box Risk: If you don't read the code, you don't see the vulnerabilities (e.g., hardcoded keys, insecure data handling).
- Blind Trust: Vibe coding prioritizes speed and functionality over security best practices.
- The "Vibe Check" Gap: It works ("good vibes"), but is it safe?

Top 5 Risks in the Era of Vibe Coding

1. The "Hallucinated Package" Attack

The Vibe: "I need a library to parse this specific old file format."

The Risk: AI models sometimes invent package names that sound plausible but don't exist. Attackers can register these "hallucinated" names (Dependency Confusion) on public repositories (npm, PyPI) and inject malicious code that your vibe-coded app automatically downloads.

2. Hardcoded Secrets & API Keys

The Vibe: "Connect to this database and show me the user list."

The Risk: To get the code working instantly, AI often hardcodes credentials (AWS keys, DB passwords) directly into the source code rather than using environment variables. If the user doesn't read the code before pushing to GitHub, those secrets are public.

3. Insecure Default Configurations

The Vibe: "Spin up a quick server to handle these requests."

The Risk: AI prioritizes functionality over hardening. It may generate a server that runs as root, has debugging modes enabled (exposing stack traces), or allows all Cross-Origin (CORS) requests (Access-Control-Allow-Origin: *), leaving the door wide open for attacks.

Top 5 Risks in the Era of Vibe Coding

4. Logic Flaws & Edge Case Blindness

The Vibe: "Make sure users can only see their own data."

The Risk: The AI might implement a simple check (e.g., client-side validation) that looks like it works but lacks robust server-side enforcement. Because the human isn't auditing the logic, subtle flaws (like Insecure Direct Object References - IDOR) slip through.

5. Training Data Regurgitation (Copy-Paste Vulnerabilities)

The Vibe: "Write a function to sanitize user input."

The Risk: The AI is trained on vast amounts of internet code—including old, insecure Stack Overflow answers. It might suggest outdated hashing algorithms (like MD5) or vulnerable regex patterns (ReDoS) because they were statistically common in its training data.

When Does Security Become a Concern?

The "Safe Zone" (Low Risk)

What it looks like: Local-only prototypes, static HTML/CSS, visual mockups, "Hello World" apps.

The Vibe: "I'm just playing with colors and layout."

Verdict: Minimal security concern. If it breaks, only your local machine sees it.

When Does Security Become a Concern?

The Tipping Point (The Danger Zone Begins)

External Data Ingestion: The moment your code accepts input from the outside world (e.g., a file upload, a user comment, or a URL).
Risk: Injection attacks (SQLi, XSS) become possible.

API Integration: Connecting to OpenAI, Stripe, or a weather service.
Risk: This requires keys. Where is the AI storing them? (Likely hardcoded).

Database Connections: Moving from "mock data" to a real SQL or NoSQL database.
Risk: The AI might write queries that allow unauthorized access to user records.

When Does Security Become a Concern?

The "Red Alert" Zone (High Risk)

Authentication & Identity: Asking the AI to "build a login page."

Risk: Rolling your own auth is notoriously dangerous; AI might use weak hashing or skip session validation.

Deployment: Moving code from localhost to a public URL (Vercel, Netlify, AWS).

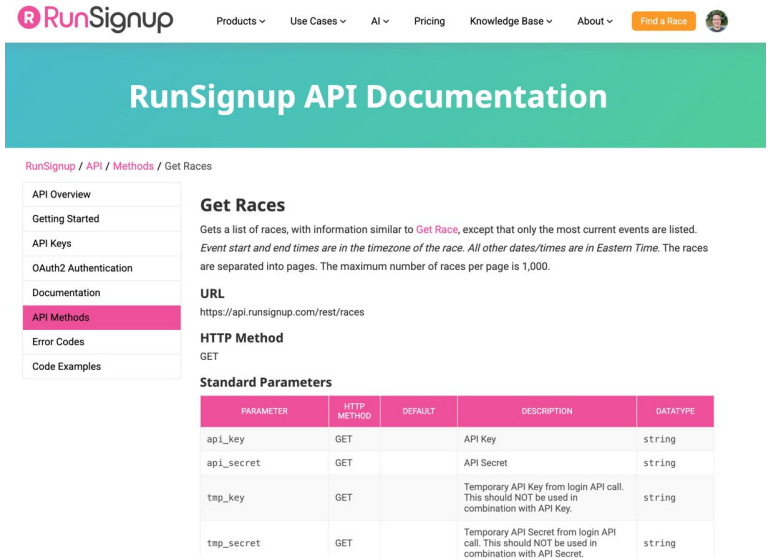
Risk: Your "vibe" is now discoverable by bots and scanners looking for exposed endpoints.

Best Practice: Principle of Least Privilege

No Credentials	API Keys	OAuth 2.0
Getting publicly available information such as race information.	Getting information about races you have access to either as an event director or partner.	Building applications that allow others to authenticate using their RunSignup account and authorizing access.

Work Without Credentials

https://runsignup.com/API/races/GET



“I’d like to build a race finding app that will display upcoming races within a certain distance of a zip code. Use the RunSignup API for getting races.

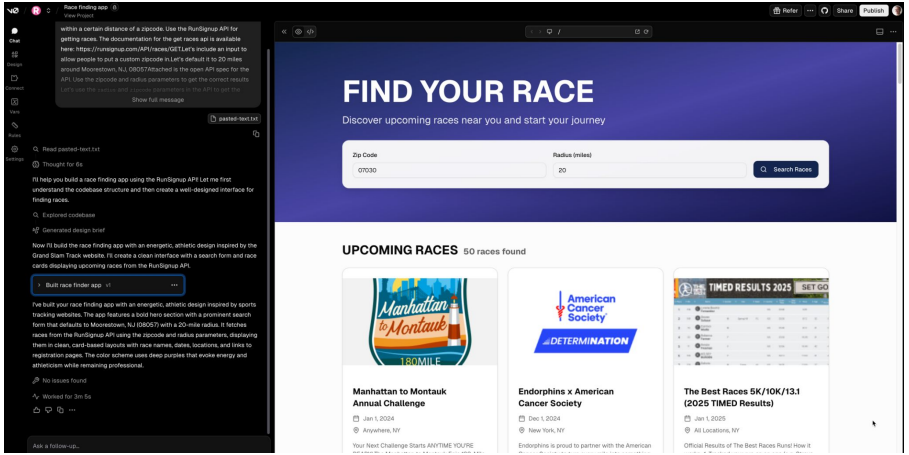
The documentation for the get races api is available here: https://runsignup.com/API/races/GET.

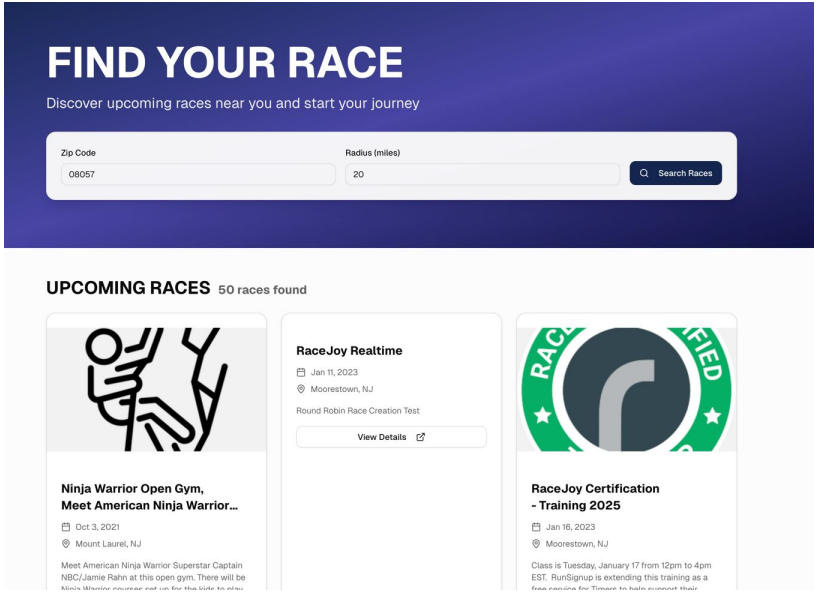
Include an input to allow people to put a custom zip code in.

Set the default to 20 miles around Moorestown, NJ, 08057

Attached is the open API spec for the API. Use the `radius` and `zipcode` parameters in the API to get the correct results”

Work Without Credentials

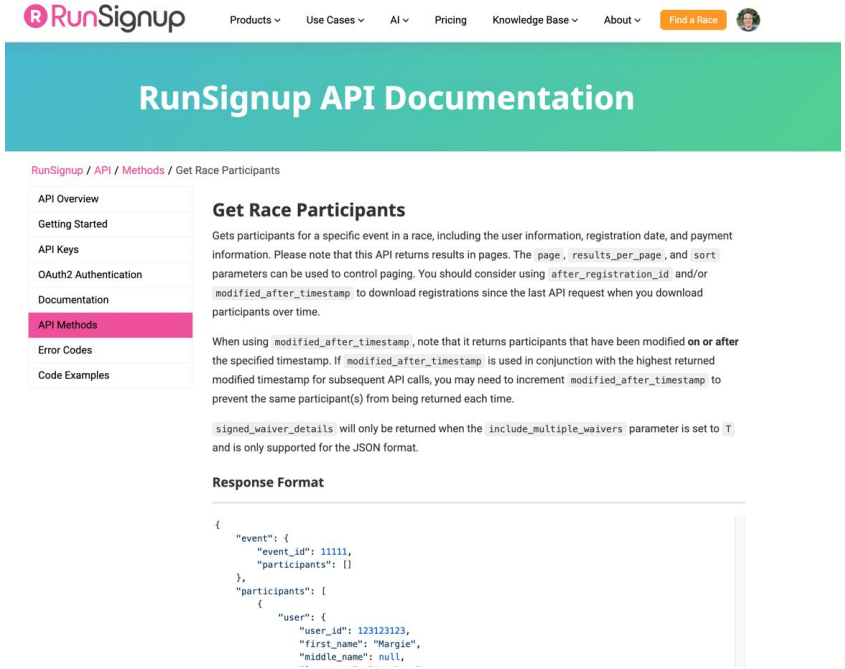




https://v0-race-finding-app.vercel.app/

When Credentials Are Required

https://test.runsignup.com/API/race/:race_id/participants/GET



“I have a race on test.runsignup.com with ID 7257. I’d like to create an app that lists the participants of the race for event ID 18220.

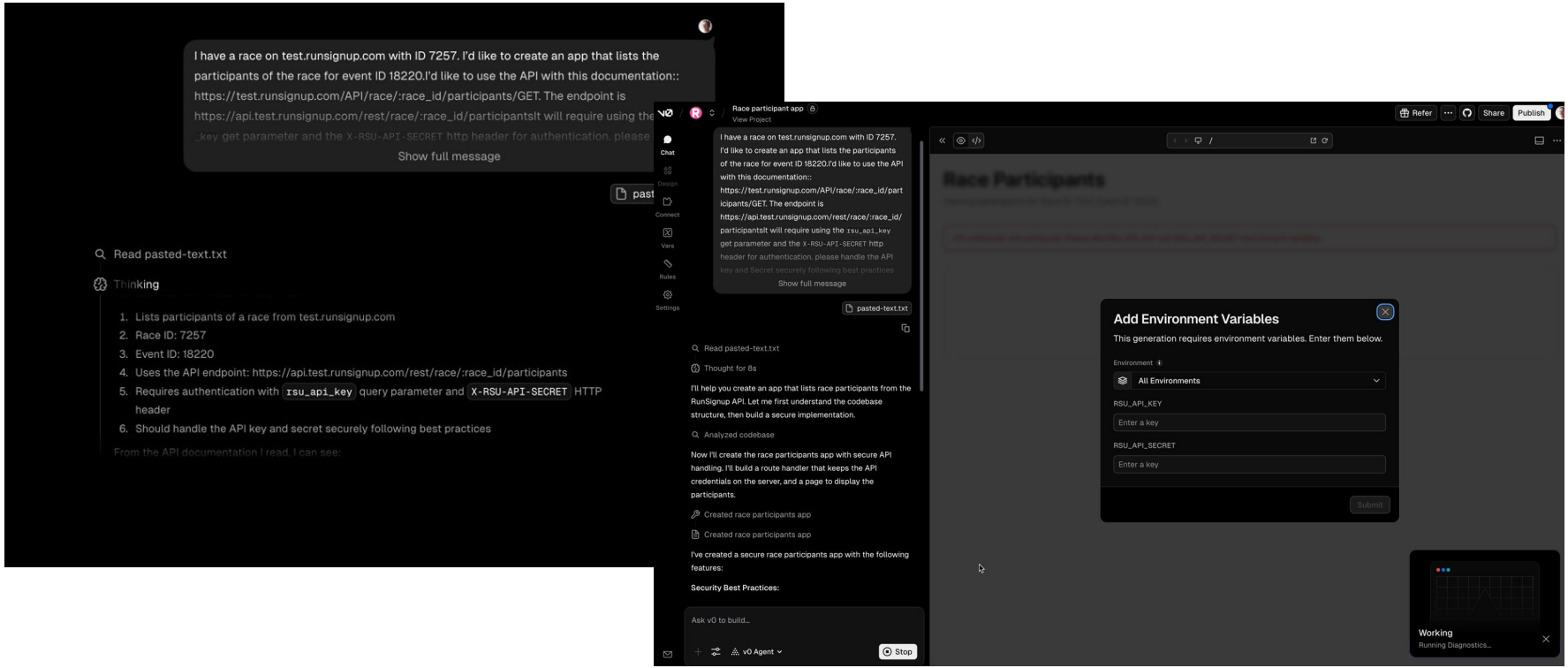
I’d like to use the API with this documentation:: https://test.runsignup.com/API/race/:race_id/participants/GET. The API endpoint is https://api.test.runsignup.com/rest/race/:race_id/participants

Attached is the OpenAPI Specification

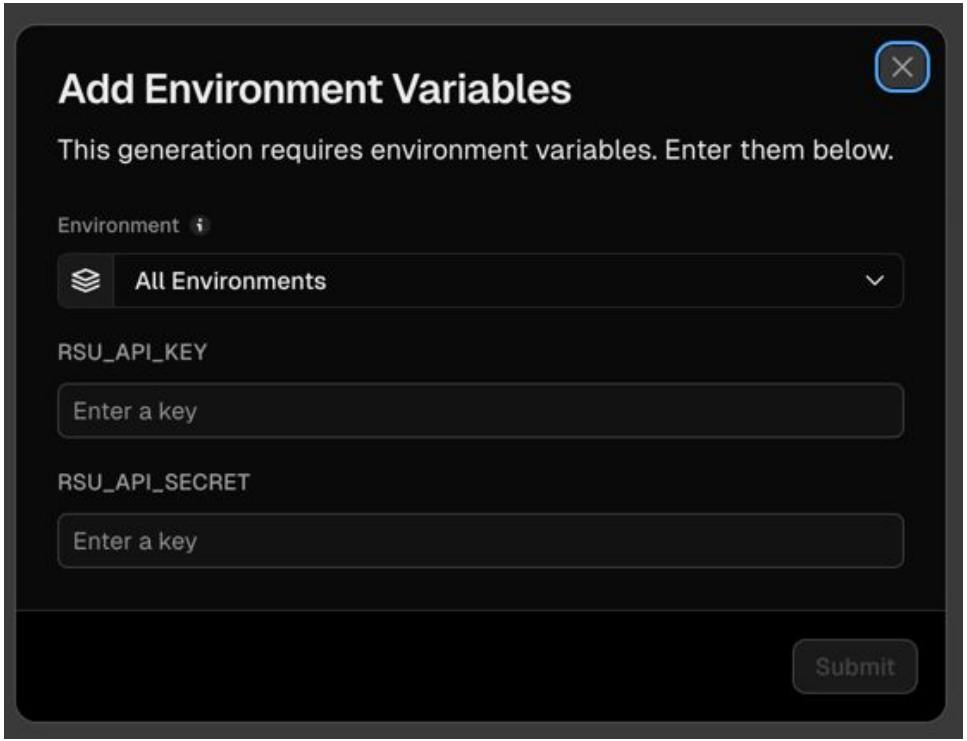
It will require using the `rsu_api_key` get parameter and the `X-RSU-API-SECRET` http header for authentication.

Please handle the API key and Secret securely following best practices from a security perspective.”

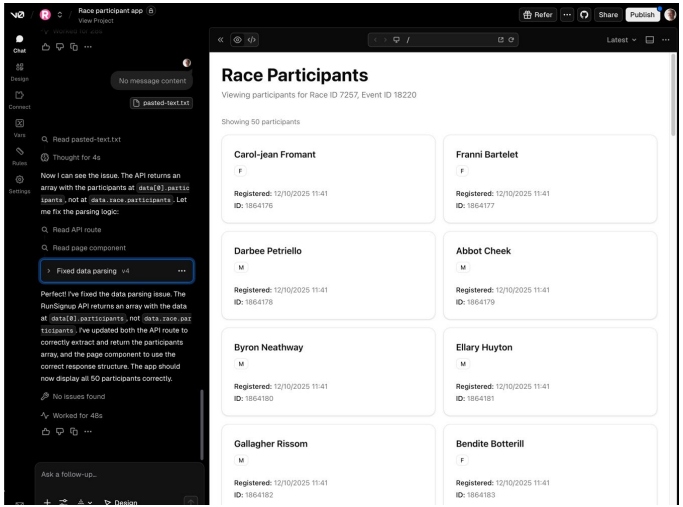
When Credentials Are Required



When Credentials Are Required



When Credentials Are Required



https://v0-race-participant-app.vercel.app/

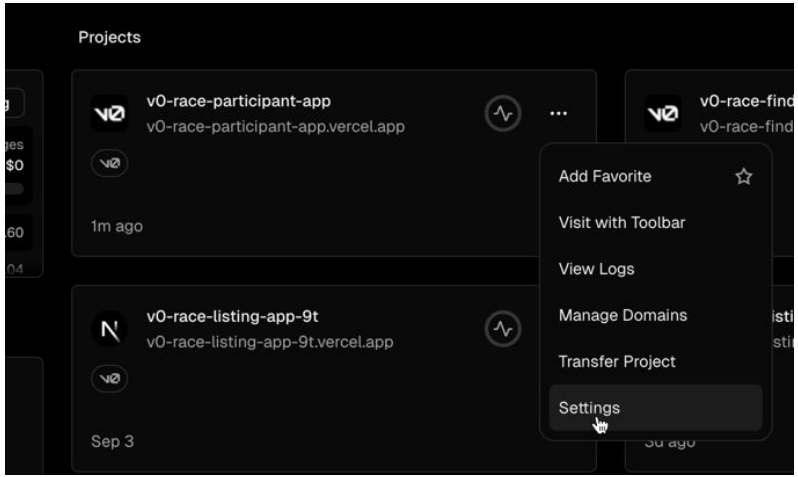
Race Participants

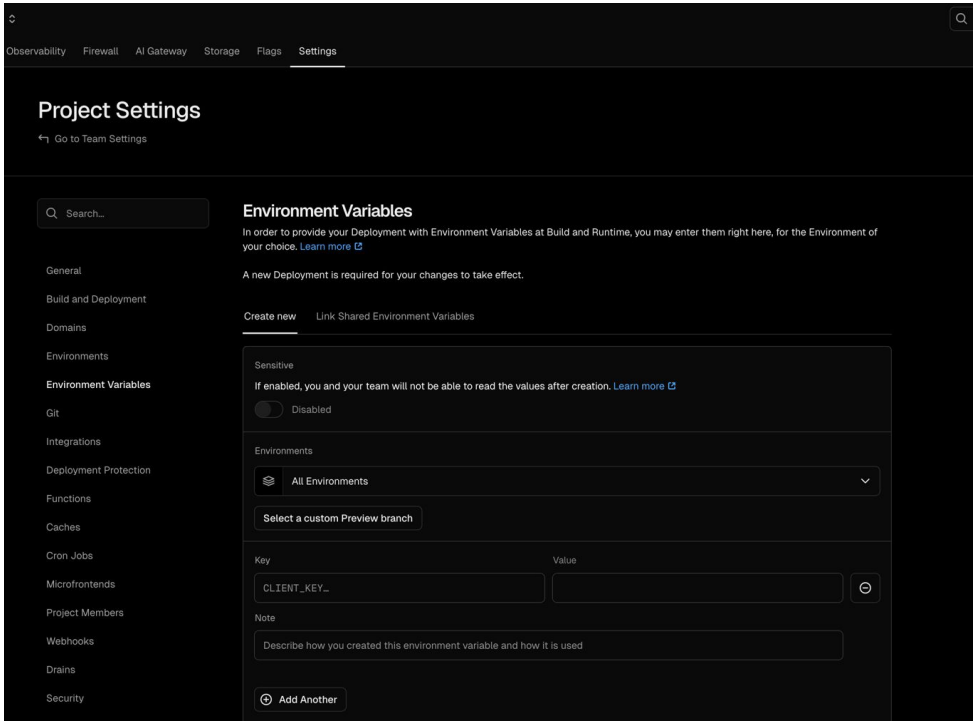
Viewing participants for Race ID 7257, Event ID 18220

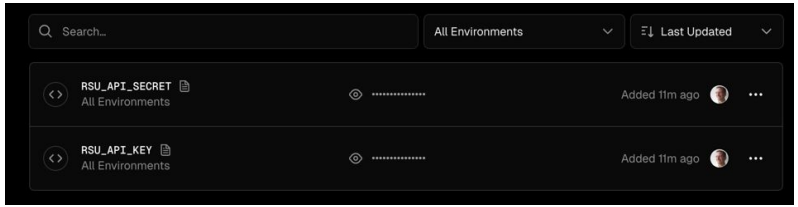
Showing 50 participants

Carol-jean Fromant F **Registered:** 12/10/2025 11:41 **ID:** 1864176	**Franni Bartelet** F **Registered:** 12/10/2025 11:41 **ID:** 1864177	**Darbee Petriello** M **Registered:** 12/10/2025 11:41 **ID:** 1864178
Abbot Cheek M **Registered:** 12/10/2025 11:41 **ID:** 1864179	**Byron Neathway** M **Registered:** 12/10/2025 11:41 **ID:** 1864180	**Ellary Huyton** M **Registered:** 12/10/2025 11:41 **ID:** 1864181
Gallagher Rissom M **Registered:** 12/10/2025 11:41 **ID:** 1864182	**Bendite Botterill** F **Registered:** 12/10/2025 11:41 **ID:** 1864183	**Shannan Priditt** M **Registered:** 12/10/2025 11:41 **ID:** 1864184
Keri Metson F **Registered:** 12/10/2025 11:41 **ID:** 1864185	**Nester Hanford** M **Registered:** 12/10/2025 11:41 **ID:** 1864186	**Glenna Oxherd** **Registered:** 12/10/2025 11:41 **ID:** 1864187

Setting Environment Variables in Vercel

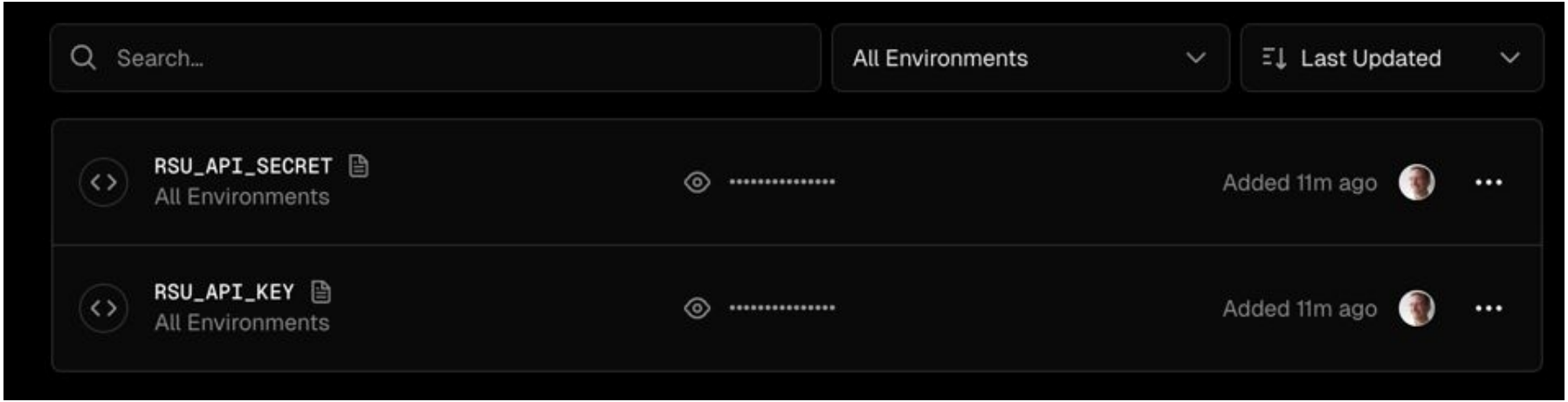






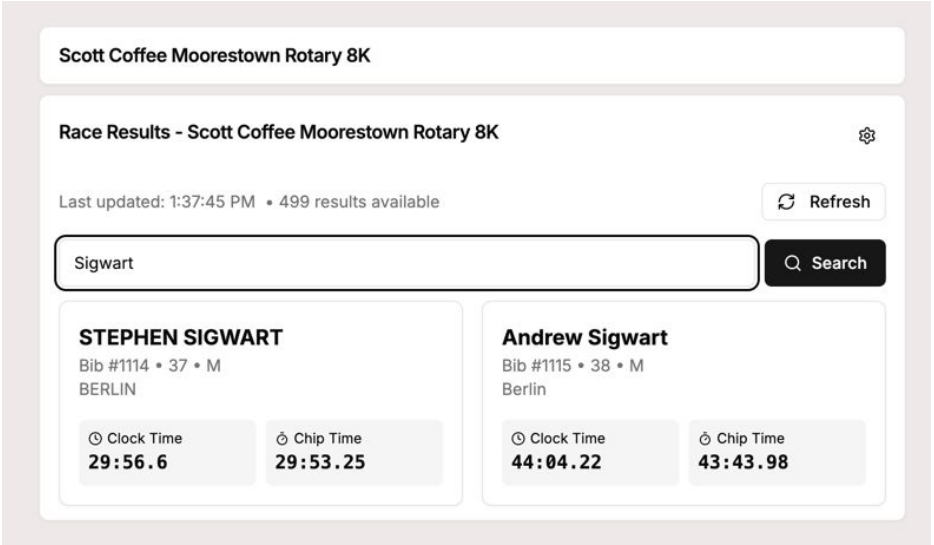
https://v0-race-participant-app.vercel.app/

Setting Environment Variables in Vercel



Does The Tool Already Exist?

https://runsignup.com/TimerUtilities



The "Vibe Check" – A Security Checklist

1. The "No Naked Keys" Rule

- **The Action:** Never let the AI hardcode API keys or passwords.
- **The Prompt Fix:** Explicitly instruct the AI: *"Use environment variables for all secrets. Do not hardcode credentials."*

2. The 60-Second Syntax Audit

- **The Action:** Don't just run the code; read the logic blocks.
- **The Focus:** Look specifically at **imports** (Are these real libraries?) and **data flow** (Is user input going directly into a SQL query?).

3. "Adversarial" Prompting

- **The Action:** Use the AI to audit itself.
- **The Prompt Fix:** After the code is written, open a new chat window and paste the code back in with the prompt: *"Act as a security researcher. Find three potential vulnerabilities in this code and show me how to fix them."*

Q/A

RunSignup