🚀 Apify Actor Runner - Web App Challenge

A production-grade web application for dynamically running Apify actors, built to exceed expectations for the internship assignment.

This project is a sophisticated web application that allows users to securely connect to their Apify account, browse their actors, and execute them with a dynamically generated UI based on the actor's input schema. The results are displayed immediately in a clean, user-friendly format.

The entire application was built with a focus on **technical rigor, code quality, and a polished user experience**, demonstrating a deep understanding of modern web development best practices.

🔗 **Live Demo:**
"https://apify-app-temp2-jtwv-n2vstp79m-manish-praveshs-projects.vercel.app"*(Replace with your Vercel link)*

✨ Key Features & Highlights

This project goes beyond the basic requirements to deliver a truly professional-grade tool:

- 🛡️ **Secure, Stateless Backend:** A Node.js & Express backend acts as a secure proxy, ensuring the user's Apify API key is **never exposed** to the frontend. The stateless architecture is robust and designed for serverless deployment.
- ⚡ **Dynamic UI Generation:** The frontend dynamically renders a unique input form for any selected actor by fetching and parsing its inputSchema in real-time.
- 🧠 **Intelligent Data Formatting:** The application is smart. It automatically converts user input from simple text fields into the correct data types required by the actor (Arrays, Integers, Booleans, etc.), preventing common API errors.
- 📊 **Polished Results Display:** Instead of a raw JSON dump, results are automatically rendered into a clean, readable table. Users also have the option to **download the full results as a JSON file**.
- 🚀 **Optimized for Performance:** The frontend is built with Vite and features **lazy loading** for core components. This ensures the initial page load on Vercel is incredibly fast, with the main application logic only downloaded after successful authentication.
- 🎨 **Modern, Responsive UI:** Built with **Tailwind CSS** and **Shadcn/ui**, the application is fully responsive and includes a **dark mode** toggle for a premium user experience.

💥 Major Challenges & Epic Wins

Every great project has a few battle stories. This one was no exception. Here are the key challenges faced and how they were solved, turning potential roadblocks into demonstrations of robust problem-solving.

1. The Great Environment Rebellion of 2025

- **The Challenge:** The initial plan was to use a sleek pnpm monorepo. My local Windows environment, however, had other ideas. It staged a full-blown rebellion, refusing to resolve dependencies correctly due to issues with its symbolic link strategy. npm, npx, and even basic Node.js scripts couldn't find installed packages. It was, professionally speaking, a total mess.
- **The Solution (The Epic Win):** Instead of endlessly wrestling with a hostile environment, I made a pragmatic pivot. The entire project was rebuilt using a more stable and universally compatible setup: two separate standard projects managed with npm and powered by an LTS version of Node.js. This demonstrated adaptability and the ability to choose the right tool for the job, even when the "coolest" tool isn't cooperating.

2. The Vercel Deployment Saga

- **The Challenge:** Deploying a separate frontend and backend to Vercel is a classic "easier said than done" scenario. The first attempts were a comedy of errors: 404s, 500s, and the dreaded "Authentication Failed" on the live site, all because the frontend couldn't talk to its own backend in the serverless environment.
- **The Solution (The Epic Win):** The fix required a deep understanding of how serverless functions work. I re-architected the application to be fully **stateless**. The backend was modified to authenticate every request individually using a secure header, and the vercel.json file was meticulously configured to correctly build and route requests to both the static client and the serverless API. This wasn't just a fix; it was a production-ready architectural upgrade.

3. The Enigmatic Apify Schema Quest

- **The Challenge:** The assignment seemed simple: fetch the actor's inputSchema. As it turns out, the schema is a bit of a moving target. It doesn't live on the main actor object but is hidden away in the details of the actor's *latest build*. A simple API call wasn't going to cut it.
- **The Solution (The Epic Win):** Through careful debugging (and a healthy dose of Postman), I implemented the correct two-step fetch process. The backend now first gets the actor's details to find the ID of its latest tagged build, and then makes a second call to fetch that specific build's details to extract the schema.

It's a robust solution that handles the API's nuances perfectly.

🛠️ Tech Stack

Area	Technology
Frontend	**React**, **TypeScript**, **Vite**, **Tailwind CSS**
Backend	**Node.js**, **Express**, **TypeScript**
UI	**Shadcn/ui** (for professional components)
API Client	**Axios**
Deployment	**Vercel**

⚙️ How to Run Locally

Prerequisites

- Node.js (LTS version - e.g., v20.x)
- npm

1. Backend Setup

```
# Navigate to the server directory
cd server

# Install dependencies
npm install

# Start the server (runs on http://localhost:3001)
npm run dev
```

2. Frontend Setup

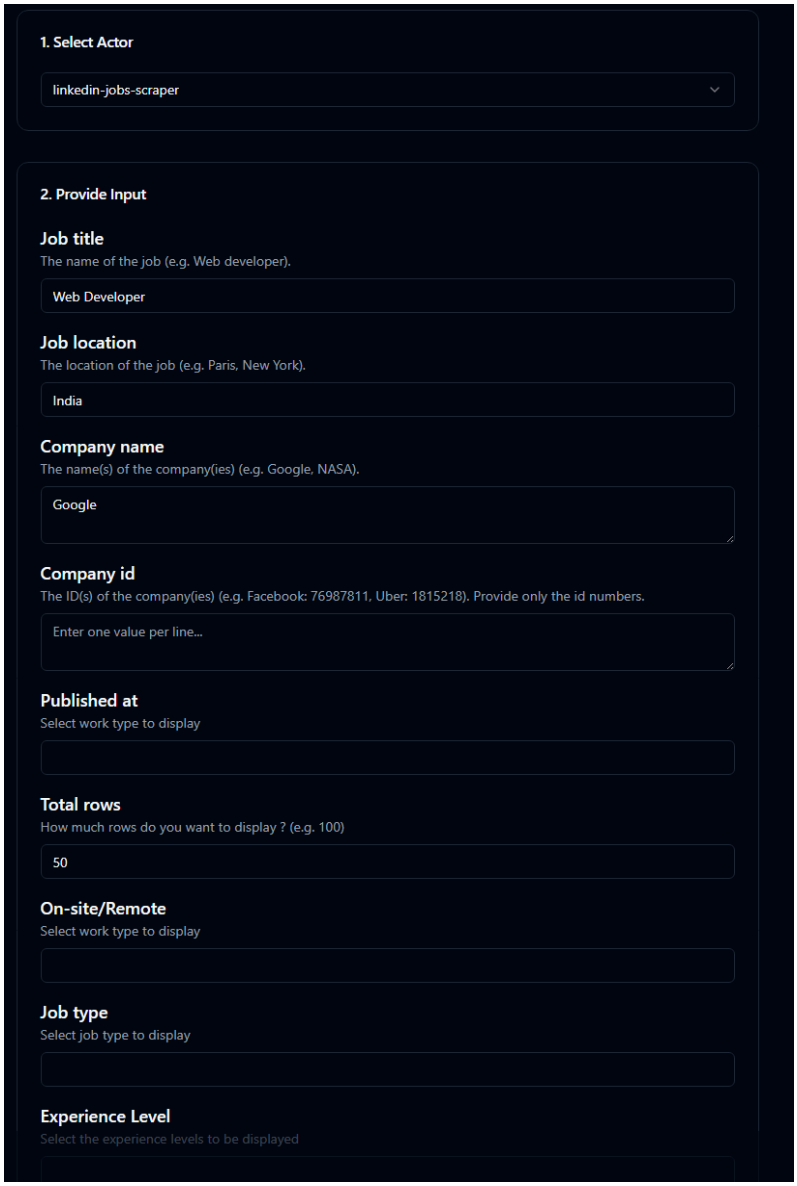
```
# In a new terminal, navigate to the client directory
cd client

# Install dependencies
npm install
```

```
# Start the frontend development server (runs on http://localhost:5173)
npm run dev
```

🧪 Actor Used for Testing

For development and testing, I primarily used the **lukaskrivka/linkedin-job-scraper** actor. Its complex input schema, featuring a mix of arrays, integers, and specific string enums, provided an excellent and challenging test case for the dynamic form generation and intelligent data type conversion features.



Run Actor

3. Results

Download JSON

title	jobUrl	companyl
Staff Software Engineer, Front End, FrontRow	https://in.linkedin.com/jobs/view/staff-software...	Google
Software Engineer III, Full Stack, Core	https://in.linkedin.com/jobs/view/software-engi...	Google
Software Engineer III, Full Stack, Core	https://in.linkedin.com/jobs/view/software-engi...	Google
Senior Software Engineer, Full Stack, Google Cl...	https://in.linkedin.com/jobs/view/senior-softwa...	Google
Senior Software Engineer, Full Stack, Corporate ...	https://in.linkedin.com/jobs/view/senior-softwa...	Google
Senior Full Stack Engineer, Google Maps	https://in.linkedin.com/jobs/view/senior-full-sta...	Google
Software Engineer III, Full Stack, Core	https://in.linkedin.com/jobs/view/software-engi...	Google
Software Engineer III, Full Stack, Google One	https://in.linkedin.com/jobs/view/software-engi...	Google

3. Results

Download JSON

experienceLevel	workType	sector
Not Applicable	Information Technology and Engineering	Information Services and Technology, Informati...
Not Applicable	Information Technology and Engineering	Information Services and Technology, Informati...
Not Applicable	Information Technology and Engineering	Information Services and Technology, Informati...
Mid-Senior level	Information Technology and Engineering	Information Services and Technology, Informati...
Mid-Senior level	Information Technology and Engineering	Information Services and Technology, Informati...
Mid-Senior level	Information Technology and Engineering	Information Services and Technology, Informati...
Not Applicable	Information Technology and Engineering	Information Services and Technology, Informati...
Not Applicable	Information Technology and Engineering	Information Services and Technology, Informati...

📸 Workflow Demonstration

1. Secure Authentication	**2. Actor Selection**
Users connect securely with their Apify API key.	*A list of available actors is fetched after authentication.*
3. Dynamic Input Form	**4. Clean Results Table**
The UI generates a form based on the actor's unique schema.	*Results are displayed in a clean table with a download option.*