# Project "Verify"

**Verify** is an AI-powered Chrome extension that acts as an "Information Nutrition Label" for online articles. Instead of giving a simple "true" or "false" verdict, its goal is to empower users to build their critical thinking skills by providing a transparent, multi-layered analysis of the content they are reading.

The project's design is guided by the core HCI principles of **Trust, Transparency, and Clarity.**

## What It Does

When a user is on a news article or web page, they can click the "Verify" icon to open a side panel. After clicking "Analyze," the extension provides a comprehensive report with four key sections:

1. **Site Reputation:** Analyzes the trustworthiness and political bias of the entire website domain (e.g., `wikipedia.org`).
2. **Article Tone (Sentiment):** Analyzes the language of the specific article to detect its sentiment (Positive, Negative, Neutral) and bias (Objective, Biased, Strongly Biased).
3. **AI Authorship:** Provides a probability score of whether the article was written by a human or generated by an AI.
4. **Fact-Check Results:** Extracts the key factual claims from the article and individually verifies them, labeling each as "Verified," "Disputed," or "Questionable."
5. **Interactive Highlighting:** Automatically highlights the verified claims on the original web page, directly connecting the analysis to the content.

## Technical Architecture

The project is a "monorepo" composed of two separate applications that run at the same time:

1. `verify-backend` **(The Engine):**
   - A **Node.js + Express.js** server that runs locally on your computer.
   - Its only job is to receive text and a URL from the frontend.
   - It then runs 5+ parallel and serial AI analysis tasks using the Google Gemini API.
   - It returns a single, complete JSON object to the frontend.
1. `verify-frontend` **(The UI):**
   - A **React + Vite** application built as a **Chrome Extension (MV3)**.
   - It uses the **Chrome Side Panel API** to display the user interface.
   - It's responsible for scraping the text from the active web page, sending it to the backend, and rendering the beautiful, animated UI based on the JSON response.

## Tech Stack

| Component | Technology | Purpose |
|---|---|---|
| **Backend** | Node.js | The server environment. |
| | Express.js | For creating the `/analyze` API endpoint. |
| | `@google/generative-ai` | The Google Gemini API, used for all 5 AI analysis tasks. |
| | `axios` | Used to call the Google Fact Check API. |
| **Frontend** | React | For building the side panel user interface. |
| | Vite | For fast, modern frontend building and bundling. |
| | TailwindCSS | For all UI styling and the "look and feel." |
| | `lucide-react` | For the icons in the UI. |
| | Chrome Extension API | For the side panel, scripting, and tabbing. |
| **APIs** | Google Gemini API | Powers all 5 core analysis features. |
| | Google Fact Check API | The primary database for checking known misinformation. |

## Getting Started: How to Run the Project

To run this project, you will need **two terminal windows** open at the same time.

## Prerequisites

- **Node.js** installed on your computer.
- **A Google Gemini API Key:** Get this from Google AI Studio.
- **A Google Cloud API Key:** Get this from Google Cloud Console. You must **enable the "Fact Check Tools API"** for this key.

## Step 1: Run the Backend

First, we need to get the "engine" running.

1. Open your first terminal window.
2. Navigate to the backend folder:
3. `cd path/to/verify-backend`
4. 
5. Create a secret `.env` file to hold your API keys:
6. `touch .env`
7. 
8. Open the `.env` file in a text editor and add your two keys. (It must be these exact names).
9. `GEMINI_API_KEY=YOUR_GEMINI_KEY_GOES_HERE`
10. `GOOGLE_API_KEY=YOUR_GOOGLE_CLOUD_KEY_GOES_HERE`
11. 
12. Install all the necessary packages:
13. `npm install`
14. 
15. Start the server:
16. `node server.js`
17. 
18. **Success!** The terminal should print:
19. `Server is running on http://localhost:3001`
20. `Gemini API Key loaded successfully!`
21. `Google Fact Check API Key loaded successfully!`
22. 
23. Leave this terminal running.

## Step 2: Build and Load the Frontend

Now, in a *new* terminal window, we'll build the UI.

1. Open your second terminal window.
2. Navigate to the frontend folder:
3. `cd path/to/verify-frontend`
4. 
5. Install all the necessary packages. (If you've had issues, run `rm -rf node_modules && rm package-lock.json` first).
6. `npm install`
7. 
8. Build the final, optimized application:
9. `npm run build`
10. 
11. This creates a new folder named **`dist`** inside `verify-frontend`. This `dist` folder is your entire Chrome extension.

## Step 3: Load the Extension in Chrome

1. Open the Chrome browser.
2. Go to this URL: `chrome://extensions`
3. In the top-right corner, turn on the **"Developer mode"** toggle.
4. Click the **"Load unpacked"** button that appears.
5. A file prompt will open. Navigate to your project and select the **`verify-frontend/dist`** folder.
6. The "Verify" extension will appear in your list. Click the puzzle-piece icon in your toolbar and "Pin" the extension.

## Step 4: Run Your First Analysis!

1. Go to any news article or Wikipedia page.
2. Click the "Verify" icon in your toolbar.

3. The side panel will open.
4. Click the **"Analyze Page"** button.
5. Watch as the full "Information Nutrition Label" appears and the claims are highlighted on the page.

## How It Works: The Data Flow

1. **User Clicks "Analyze Page"**.
2. The React `App.jsx` file gets the current `tab.url` and injects a script to get the `document.body.innerText`.
3. The frontend sends a `POST` request to `http://localhost:3001/analyze` with the `articleText` and `articleUrl`.
4. The Node.js server receives the request.
5. The server's `/analyze` endpoint runs **four** analysis tasks *in parallel* using `Promise.all`:
   - o `analyzeSite(articleUrl)`
   - o `analyzeSentiment(articleText)`
   - o `analyzeAuthorship(articleText)`
   - o `extractClaims(articleText)`

1. Once these are complete, it begins **Step 2**:
   - o It loops through the list of `rawClaims` one by one.
   - o For each claim, it calls `verifyClaim()`.
   - o `verifyClaim()` *first* checks the **Google Fact Check API**.
   - o If no match is found, it *falls back* to `corroborateWithGemini()` for a real-time web search.

1. The server bundles all the results into a single JSON object and sends it back to the frontend.
2. The React app receives the JSON, updates its state, and renders the `SiteAnalysis`, `SentimentAnalysis`, `AuthorshipAnalysis`, and `ClaimCard` components.
3. Finally, `App.jsx` calls `runHighlight()`, which injects the `highlighter.js` script into the web page to highlight the text.