# Full Stack Engineering Challenge: Brand Protection Monitor (PoC)

This assignment is designed to evaluate your technical, architectural, and communication skills in a time-boxed, real-world scenario. You are tasked with creating a Proof-of-Concept (PoC) web application that actively monitors a public Certificate Transparency (CT) log to detect potential brand phishing or domain abuse.

The suggested time budget for this task is **8 hours**, but there is no strict time limit. Please submit your code along with a description of any planned features that remain incomplete.

---

## 1. Core Objectives

Create a full-stack application that achieves the following:

1. **Monitor:** Connect to a specific CT Log and retrieve recent certificate entries.
2. **Process:** Check the certificate domains (Common Name and Subject Alternative Names) against a user-defined list of keywords.
3. **Display:** Provide a web interface to manage keywords, view processing status, and highlight matching certificates.
4. **(Bonus points) Export:** Allow users to export the results (matched certificates) to a CSV file.

## 2. Required Technology Stack

The solution **must** be implemented using the following stack. Adherence to this stack is a key evaluation criterion.

| Component | Technology | Primary Role |
|---|---|---|
| **Backend API/Service** | **Golang (Go)** | CT Log Consumer, Keyword Processor, REST API Server. |
| **Database** | **PostgreSQL** | Persistent storage for keywords, matched certificates, and monitor state. |
| **Frontend UI** | **React** | Interactive dashboard for configuration and display. |
| **Styling** | **Tailwind CSS** | Utility-first styling for responsive and modern UI. |
| **Type Safety** | **TypeScript** | **Required for the React frontend** and preferred for enhanced development. |
| **Communication** | **REST APIs** | Defines the interface between the Go backend and the React frontend. |

## 3. Implementation Details & Constraints

### 3.1. CT Log Source & Consumption

- **CT Log URL:** https://oak.ct.letsencrypt.org/2026h2
- **Performance Note (PoC Scope):** You are **not** required to process the entire volume of the CT log in real-time. The Go backend should retrieve a fixed, recent batch of certificates (e.g., the last **100 entries**) periodically (e.g., every minute) to demonstrate the core monitoring loop. There is no need to store all the certificates, just those that trigger on a keyword.
- **HINT:** Use the get-entries function defined in **RFC 6962** (Certificate Transparency) to retrieve a batch of certificates, leveraging the information from the latest Signed Tree Head (STH).

### 3.2. Feature Requirements

| Feature | Description |
|---|---|
| **Keyword Management** | The frontend must allow a user to **add/remove/view** the list of keywords being monitored. These keywords must be persisted in **PostgreSQL**. |
| **Certificate Matching** | The Go service must parse the certificate data and identify if the Common Name (CN) or any Subject Alternative Name (SAN) contains a monitored keyword. |
| **Monitoring Dashboard** | Display the list of **matched certificates** (domain, issuer, date, matched keyword). **Matched entries must be prominently highlighted** (e.g., color-coded). |
| **Status Feedback** | The UI must clearly show the user that the monitor is active and processing. Display metrics such as the **total number of certificates processed** in the last cycle. |
| **Export Functionality** | Implement a backend endpoint and a frontend button to download all currently stored **matched certificates** into a **CSV file**. |

### 3.3. Project Rules & Submission

- **Code Usage:** You **must** use AI where possible and are encouraged to use the internet and 3rd party code/libraries. We do expect the creator to understand and be able to explain the functionality and have verified any AI generated code.
- **Deliverables:**
    1. A **Zip file** containing all source code (including any 3rd party dependencies used).
    2. A detailed **README.md** file with the following sections:
        - **Setup/Running Instructions:** How to install dependencies and run the backend/frontend components (including database setup).

    - **Implemented Features:** A clear list of which features were completed.
    - **Design Decisions/Ambiguities:** Describe any architectural choices, interpretations of ambiguous requirements, or trade-offs made.
    - **Limitations & Known Bugs:** Document any outstanding issues or features that were not completed due to time constraints, along with an outline of your intended solution.

## 4. Evaluation Criteria

Your submission will be evaluated on the following:

- **Functionality:** Does the program work and deliver all required features, especially the core monitoring, matching, highlighting, and (bonus) CSV export?
- **Code Quality:** Idiomatic use of Go and TypeScript, clean, readable code, and effective use of comments.
- **Architecture:** Clear separation of concerns, effective use of REST APIs, and a sensible PostgreSQL schema design.
- **Communication:** Clarity and completeness of the **README.md** file.