招聘信息实时抓取与展示平台项目纲要

本项目旨在搭建一个定时抓取公司招聘信息、实时存储并按时间分类展示的网页应用。

阶段一：A环境与技术选型

模块	技术/工具	目的
编程语言	Python, JavaScript (或 TypeScript)	Python 用于爬虫脚本；JS 用于前端和数据库交互。
爬虫库	`requests`，`BeautifulSoup`（或 `Selenium`）	负责发送 HTTP 请求和解析 HTML 内容。
数据存储	Firebase Firestore	提供实时、可靠、可扩展的文档数据库，用于存储招聘数据。
前端框架	HTML/CSS/JavaScript (或 React/Angular)	用于搭建用户界面，展示招聘信息。
自动化/调度	Google Cloud Functions / Cloud Run (推荐)	用于定时执行爬虫脚本和数据清理任务。

阶段二：爬虫功能开发 (Python)

目标： 开发一个稳定、高效的爬虫脚本，能够抓取招聘岗位关键信息。

1. **目标分析：**
    - 确定公司招聘页面的 URL 和加载方式（静态 HTML 或动态 JS 加载）。
    - 分析招聘列表和详情页面的 HTML 结构，确定抓取路径。
2. **核心抓取逻辑（`crawler.py`）：**
    - 使用 `requests` 发送请求获取页面内容。
    - 使用 `BeautifulSoup` 或其他解析器定位岗位列表。
    - 对每个岗位，提取以下关键字段：
        - `job_title`（职位名称）
        - `job_link`（职位链接）
        - `posting_date`（发布日期或抓取日期 - **非常关键，用于分类和删除**）
        - `job_location`（工作地点，可选）
3. **数据清洗与结构化：**
    - 确保 `posting_date` 被统一格式化为 ISO 8601 字符串或 Unix 时间戳，以便 Firestore 存储。
    - 将所有抓取到的数据组织成 JSON 数组结构。
4. **数据库连接函数：**

- 编写函数，通过 Firebase Admin SDK (若在后端环境) 或 REST API，将抓取到的数据写入 Firestore。

阶段三：数据存储与管理 (Firebase Firestore)

目标： 建立持久化数据库，并处理数据的实时更新与删除。

1. **Firestore 初始化：**
   - 在前端代码中，使用提供的全局变量 `__firebase_config` 初始化 Firebase 应用。
   - 使用 `__initial_auth_token` 进行用户认证（匿名或自定义令牌）。
   - **数据路径规范：** 使用公共路径 `artifacts/{__app_id}/public/data/jobs` 存储招聘岗位文档。
2. **文档结构：**
   - 每个文档代表一个招聘岗位，包含以下字段：

```
{
  "job_title": "XXXX 工程师",
  "job_link": "http://...",
  "posting_timestamp": 1678886400000, // Unix 时间戳 (毫秒)
  "created_at": <Firestore Timestamp> // Firestore 自动创建的时间戳
}
```

3. **数据实时读取：**
   - 在前端使用 `onSnapshot` 监听 `jobs` 集合的变化，确保数据实时同步到网页。

阶段四：前端网页应用开发 (HTML/JS)

目标： 搭建一个单页应用，实时展示和分类招聘信息。

1. **基本页面结构 (`index.html`):**
   - 使用 Tailwind CSS (通过 CDN) 搭建响应式界面。
   - 设置三个主要筛选器/标签页：**当天** (Today)、**过去一周** (Past Week)、**过去一个月** (Past Month)。
   - 设置一个区域用于显示岗位列表。
2. **数据获取与分类 (`app.js`):**
   - 在 `onSnapshot` 回调函数中获取所有岗位数据。
   - 定义时间边界：
     - **当天：** `posting_timestamp` 在过去 24 小时内。
     - **过去一周：** `posting_timestamp` 在过去 7 天内。
     - **过去一个月：** `posting_timestamp` 在过去 30 天内。
   - 根据当前选中的标签页（当天/一周/一月），过滤数据并调用渲染函数。

3. **列表渲染：**
    - 编写函数将过滤后的数据动态渲染到页面上，显示岗位名称和链接。
    - 确保列表样式清晰、美观，响应式适配移动设备。

阶段五：自动化与维护（部署和清理）

目标： 实现爬虫的定时运行和数据的自动清理。

1. **定时爬虫调度：**
    - 将阶段二的爬虫脚本部署到 Google Cloud Functions 或 Cloud Run。
    - 使用 Cloud Scheduler 设置定时任务（例如，每 6 小时运行一次），触发爬虫函数执行。
2. **数据自动清理 (核心要求):**
    - 创建一个单独的 Cloud Function (`cleanup_jobs`)，用于数据库维护。
    - 该函数定时（例如，每天凌晨）执行以下逻辑：
        - 计算 30 天前的时间戳。
        - 查询 Firestore 中 `posting_timestamp` 小于该时间戳的所有文档。
        - 批量删除这些文档，实现"超过一个月的岗位自动删除"的需求。

项目完成标准：

1. Python 脚本能稳定抓取目标网站数据并写入 Firestore。
2. 网页能实时显示 Firestore 中的数据。
3. 网页上的分类（当天/一周/一月）准确无误。
4. 有机制（例如 Cloud Function）保证超过 30 天的数据自动删除。