## 🔍 What Is This Project About?

You're building a **web crawler and search engine system** — like a mini version of Google — but using **distributed cloud computing**. Instead of running on one computer, it runs on **many virtual machines in the cloud** (like AWS, GCP, Azure) that work together to:

- **Crawl websites** (download pages)
- **Index content** (organize it so it can be searched)
- **Handle failure** (if a part of the system breaks, it keeps working)

---

## ✅ What Needs to Be Implemented?

### 🖥️ 1. Client Interface

- A simple **web interface or CLI** (Command Line Interface) to:
  - Start a web crawl (by giving URLs)
  - Set options (like how deep to crawl, limit to domains)
  - Search the index for keywords
  - Monitor the progress

---

### 🧠 2. Master Node (Controller)

This is the brain of the system:

- **Splits URLs into tasks** and sends them to workers
- **Schedules tasks** for crawler/indexer VMs
- **Watches workers' health** (are they running or failed?)
- **Reassigns tasks** if something goes wrong
- **Manages the index** creation

---

### 🕷️ 3. Crawler Nodes (Worker VMs)

These download pages:

- **Fetch pages** from assigned URLs
- **Parse the page**: extract text, links, and metadata
- **Follow rules** like robots.txt
- **Send data** to indexer nodes
- **Report status** to master

---

## 🗂️ 4. Indexer Nodes

These build the searchable index:

- **Take content** from crawlers
- **Create index** (e.g., inverted index = word → list of pages)
- **Store index** in **cloud storage**
- **Answer search queries**

---

## 📬 5. Distributed Task Queue

- A **cloud-based message queue** (like AWS SQS or GCP Pub/Sub)
- Used for:
  - Sending tasks from master → crawlers/indexers
  - Receiving results/status from workers

---

## ☁️ 6. Cloud Storage

- Stores everything:
  - **Seed URLs**
  - **Downloaded HTML pages**
  - **Processed content**
  - **Index**
- Should be **durable and safe** (doesn't lose data)

---

## 💡 Extra Features to Implement

### ⚙️ Scalability

- You should be able to **add more VMs** when needed to crawl/index faster.

---

### 🛡️ Fault Tolerance

Make sure the system keeps working even if some parts fail:

- **Crawler failure**:
  - Master detects it (e.g., via heartbeat signals)
  - Reassigns the URLs to other crawlers
- **Indexer failure**:
  - Duplicate/store parts of the index
  - Rebuild lost data if needed
- **Reliable queues and storage**:
  - Use built-in reliability of cloud services

---

### 🧑‍💻 User Stories (What Users Can Do)

- Start a crawl with a list of URLs
- Set crawl settings (like how deep to go)
- Keep crawling even if some machines fail
- Search for words in indexed data
- Track system progress (crawl status)

---

### 🏗️ Summary of What You’ll Build

| Component | What it Does |
|---|---|
| **Client Interface** | Start crawl, search content, monitor progress |
| **Master Node** | Controls the whole system, assigns work, handles faults |
| **Crawler Nodes** | Download and parse web pages |
| **Indexer Nodes** | Build and store the search index |
| **Task Queue** | Handles task distribution and communication |
| **Cloud Storage** | Stores data and index safely |

Let’s say the website is:

🔗 **https://news.site.com**

This website might have many pages, such as:

- 📄 https://news.site.com/home
- 📄 https://news.site.com/sports
- 📄 https://news.site.com/tech/article123
- 📄 https://news.site.com/world/article456

Each of those is a **separate web page** under the same website.

**Here's how it works step by step:**

1. **You provide seed URLs**
   – For example:

   https://example.com

   https://news.site.com

2. **The system crawls those URLs**
   – It downloads those pages
   – Extracts links from them
   – Follows those links (up to a certain depth or within domain limits)

3. **The content from crawled pages is indexed**
   – The system builds a search index (like a map of words to pages)

4. **Later, you can search with keywords**
   – After crawling is done, you can search for terms like:

   "climate change"

   "machine learning"

   – And the system will return the relevant pages it crawled.