# Project Goals

Develop a web automation system that uses anti-anti-crawler technology to simulate user behavior and perform web browsing and interaction.

The system adopts a cloud service model. After the user initiates a web page access request through the API, the system will allocate host resources to create a browser instance, load the target web page, and return the rendered page content.

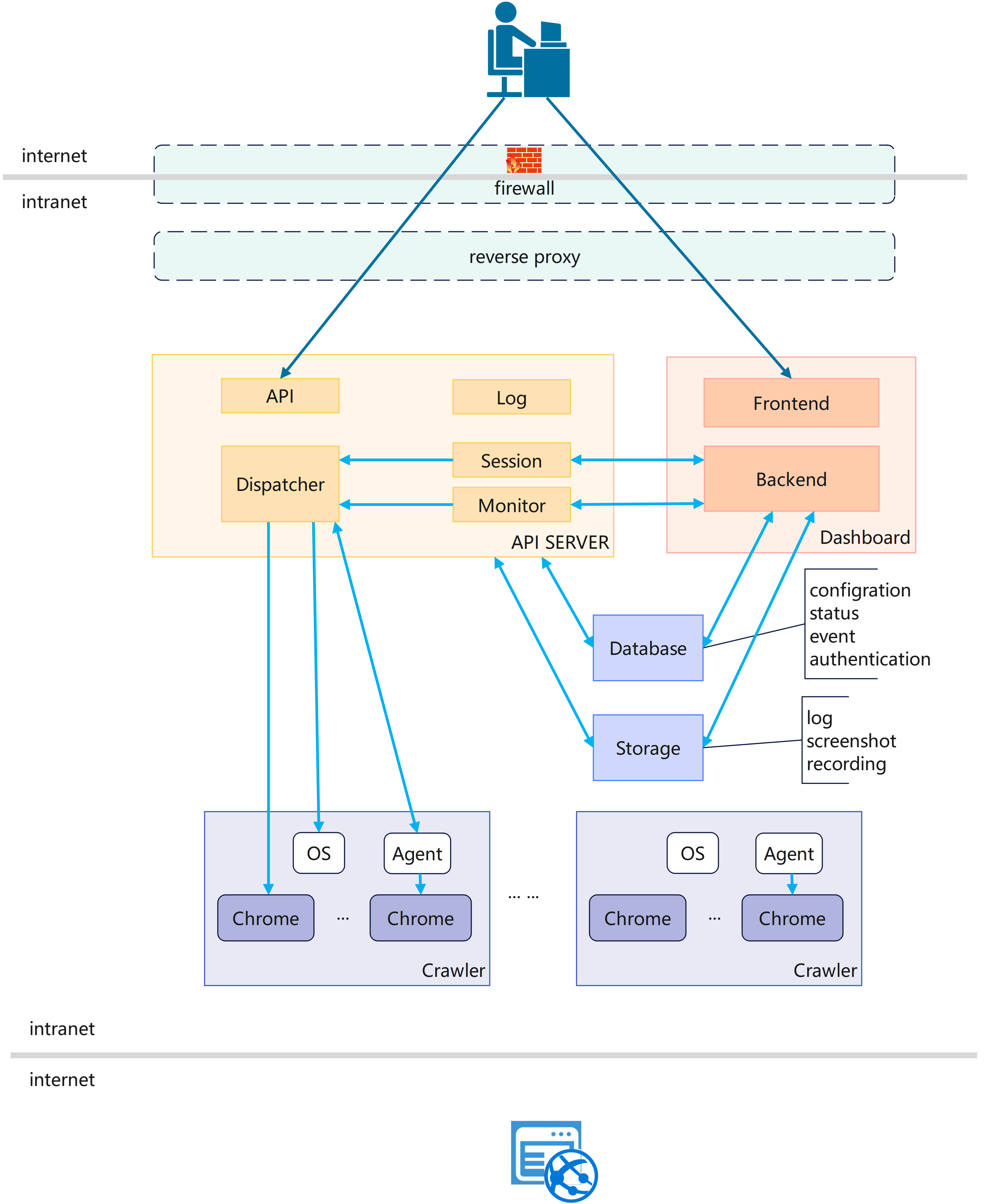
The system can maintain page state within a session and support continuous data reading or interactive operations. Such as:

* Content reading: returns formatted data so that text, links, and structured data (such as tables and lists) can be extracted
* Form filling: fill in the input box, select the drop-down option, submit the form
* Element operations: click elements , scroll pages, take screenshots

The system adopts modular design, which makes the functions independent, can be deployed flexibly, and supports distributed resource management.

# System Architecture

## Architecture diagram



## Module Description

### Crawler Server

Crawler Server is the host that performs crawler tasks. Each Crawler Server is installed with a specific browser and supports running multiple isolated browser instances at the same time .

The browser instance uses anti-anti-crawler technology (to avoid blocking) to access the target website based on user requests. The browser instance also supports specifying a proxy server.

### API Server

API The Server is responsible for accepting user requests and scheduling Crawler Servers. The main functions are:

* Manage Crawler Servers : Add, edit, and delete Crawler Servers
* Monitor Crawler Server: monitor and record the status of Crawler Server and provide abnormal alarms.
* Accept user API requests: As the access point for web page access, it provides a complete web page automation operation interface.   
  It provides parameter verification and identity authentication. Through synchronous calls, various requests can be accepted, including web page opening, content acquisition, element positioning, element operation, web page screenshots, etc. Distribute API requests to the Crawler Server according to session association or load balancing strategy .
* Manage sessions: Maintain context information for all active sessions. Supports allocation and release of session resources, supports session state management, and maintains the association between API requests and sessions.
* Logging: Record detailed request and response data to provide data support for system optimization and troubleshooting.

### Dashboard

A management system built on a B / S architecture. Provides system management functions, including configuring and monitoring distributed systems, viewing session status, viewing API call records, etc. Supports data query and analysis. Provides permission control functions.

### Data storage

This system needs to record a lot of data. Different types of data are classified and stored according to their characteristics:

* The structured data storage module is responsible for saving relational data such as system configuration, user information, and operation records.
* The unstructured data storage module is responsible for saving web page screenshots, logs and other content.

## Design principles

### Modularity

Each component is independent and communicates through well-defined interfaces.

### High Availability

The system supports cluster deployment and ensures service continuity through load balancing.

### Ease of use

The interface design takes into account security, ease of use, and expansibility. User-friendly interface.

# Technical Analysis

The core of this system is how to reliably control multiple browser instances in the operating system. This includes the following aspects.

## Crawler Server Management

Crawler Servers is a distributed system consisting of multiple Crawler Servers. Operating system is not restricted. We need to be able to add and delete Crawler Servers, monitor resources, manage processes, and manage files. There are two solutions to maintain and monitor node system resources:

* Agent mode: Develop an agent software that runs on the operating system. The agent actively connects to the API Server and provides two-way communication. It can realize node resource monitoring, operation control, heartbeat, and browser instance management.

Advantages: No username/password required, two-way communication, flexible operation control

Disadvantages: high development cost, long cycle, high technical complexity, and OS-specific

* OS mode: API Server actively connects to Crawler Server for management and control. Different protocols are used for connection and control according to different node operating systems, such as ssh\winrm\telnet, etc. It can realize node resource monitoring, operation control, and browser instance management.

Advantages: low development cost and short cycle

Disadvantages: Username/password required, one-way communication, weak controllability, OS-specific

You can choose one of the above options or adopt both at the same time.

## Browser Automatic

We use real browsers for automation. The system has no restrictions on browsers. However, different browsers use specific automation techniques. Taking Chrome browser as an example, we need to solve the following problems.

### Isolation

When a new Chrome instance is started, the user-data-dir parameter is used for data isolation. And data is cleaned up after the session ends to protect privacy.

### Automation

There are two anti-anti-crawler automation control solutions based on Chrome :

* UIAutomation: Chrome is controlled by an agent. It supports data reading and operation control. The page data structure is in a custom format , which makes the parsing or manipulation API more complicated. It is more difficult to develop. Parallel operation of multiple browser instances may be problematic.
* JS Controller: Develop an extension based on Chrome. Use the extension to connect to the API Server to accept control. Support data reading and operation control. The page data structure can be in HTML format. The page parsing is simple, and the API control is simple. Development is easier.

### Simulation

In order to improve the reliability of the crawler system, it is necessary to simulate the web page operations of real users. This includes:

* Provide more realistic hardware and software environment and geographic location
* Closer to human operating frequency(unfixed)
* Log in to the website using a valid user account
* Mixed with random meaningless clicks

## Session Management

A session is a continuous access and operation process, involving user API calls and browser instance scheduling.

### API Calls

User API calls and session lifecycle

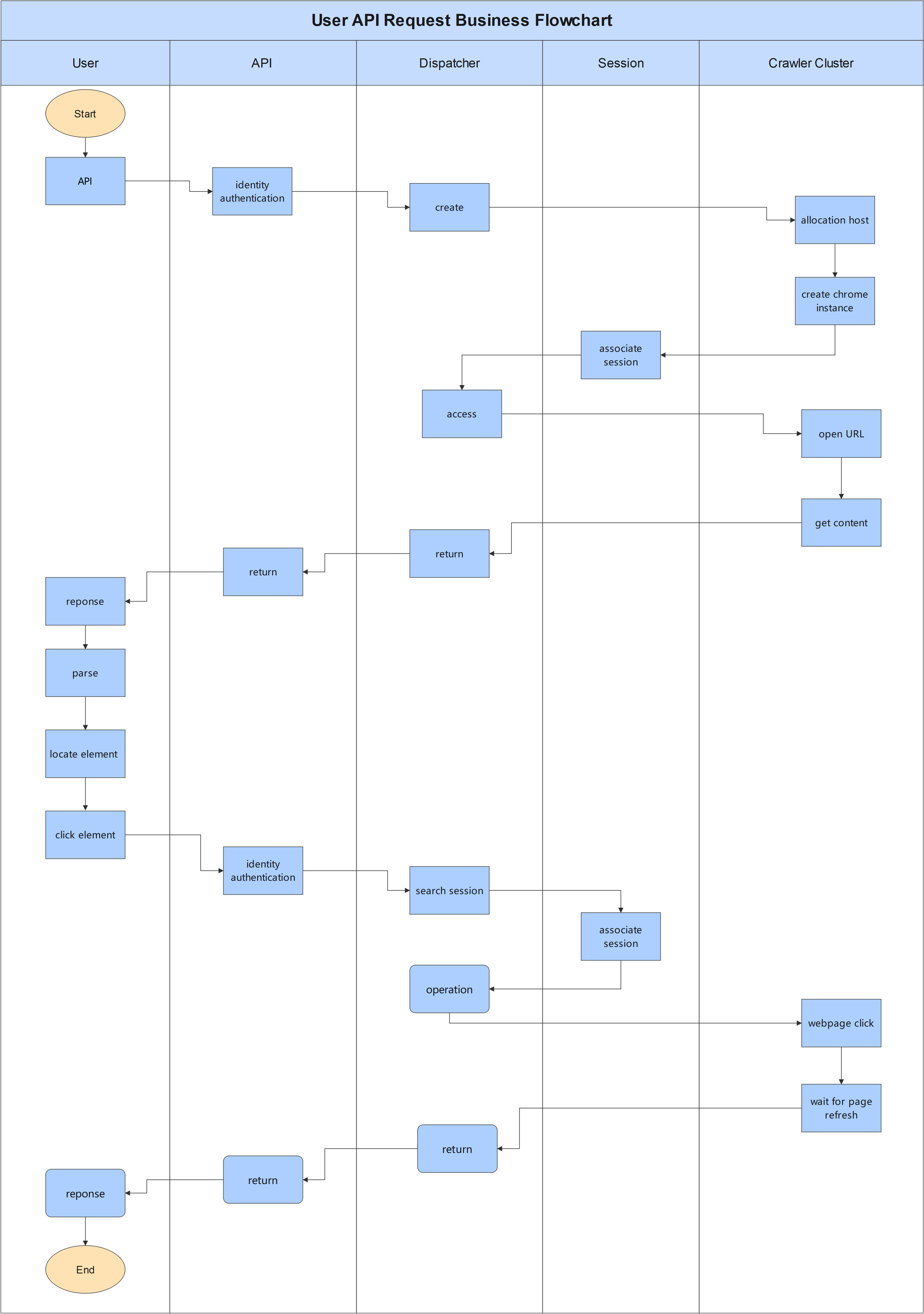
* Session creation: Depending on the API request parameters, choose to open a new session or associate with an existing browser instance.
* Session persistence: During a session, the system maintains the state of the page, and users can read data, click elements, fill in content, or jump to links on the current page.
* Session end: After the operation is completed, the user can actively end the session and the system will release related resources.

### Session Scheduling

Session and browser instance lifecycle

* Session creation: Create a browser instance based on the load of each Crawler Server node and strategy.
* Session association: Associated to an existing browser instance based on API request parameters
* Session destruction: If a session has no operations for a long time, it will be closed, resources will be released, and data will be cleaned up.

# API Call Process



# Technical limitations

## Hardware Login Credentials

For authentication methods that rely on hardware devices (such as USB Keys and smart cards), the cost of supporting them is relatively high.

## Data read restrictions

More complex technical solutions may be required for reading pictures, videos, and audios and downloading files.

## Page operation restrictions

Input to secure controls (ActiveX or NPAPI plug-ins) is somewhat difficult.

For content upload operations, additional technical solutions are required.

Direct manipulation of invisible elements may not be possible.

In headless mode, you may not be able to directly manipulate the page.

## Proxy Restrictions

Changing proxy during a session may cause access issues.

If the proxy fails during a session, it may cause subsequent access problems.

## VM Restrictions

If the host is a virtual machine, some websites may not be accessible.

# Technology Selection

The following lists the technologies involved in the technical architecture of this system for selection.

## frontend

**Vue.js**: A progressive framework with a gentle learning curve, suitable for small and medium-sized projects with rapid iteration.

**React**: It is more componentized, has a huge community ecosystem, and is suitable for complex enterprise-level applications.

## backend

**Java**: Enterprise-level development standards, mature high-concurrency scenarios, and a complete operation and maintenance tool chain.

**Node.js**: A full-stack JavaScript unified language suitable for I/O intensive applications (such as real-time applications and API gateways).

**Python**: High development efficiency and seamless integration of AI/data analysis scenarios.

## Agent​

**Go**: compiled into a single binary, easy to deploy across platforms, suitable for resource-constrained terminals.

**Python**: rapid development and controllable environment dependencies.

**C++**: high performance, flexible and controllable.

## database

* Relational

**PostgreSQL**: Comprehensive features (JSON support, GIS), the first choice for open source.

**MySQL**: easy to use, with rich operation and maintenance experience.

* NoSQL

**MongoDB**: Document-type requirements (such as flexible schema).

**Redis**: Cache/real-time scenarios.

## storage

**NAS**: Shared file storage requires a redundancy solution.

**Object Storage**: cloud storage solution with better scalability.

## WebServer

**Nginx**: High performance, lightweight, modular.

**Apache**: r Rich modules, easy to integrate, mature and stable.

## Considerations

1. Team capabilities: Avoid introducing technologies that the team is completely unfamiliar with.

2. Operation and maintenance costs: Compare the development efficiency, operation and maintenance complexity, and long-term maintenance difficulty of the technology.

3. Technology Outlook: Choose technologies that are in line with industry trends (such as cloud native, microservice architecture) and have a good evolution path.

4. Key point: Avoid abuse of technology and reduce technical complexity.

# Initial goals

## API Service

An API for crawling any URL with support of proxy and more parameters

## Dashboard

A web system for Configuring and monitoring Crawler Server with logging function

## Crawler Server

distributed crawling system via servers(Windows11) + browsers(Chrome)

# Future Features

Support asynchronous crawler task execution:

* Task script management
* Task scheduling
* Saving and retrieving data