Problem Statement

# Test task: Web application for analyzing web pages

## Objective

The objective is to build a web application that does an analysis of a web-page/URL.

The application should show a form with a text field in which users can type in the URL of the web page to be analyzed. Additionally, to the form, it should contain a button to send a request to the server.

After processing, the results should be shown to the user.

Results should contain next information:

- What HTML version has the document?
- What is the page title?
- How many headings of what level are in the document?
- How many internal and external links are in the document? Are there any inaccessible links and how many?

- Does the page contain a login form?

In case the URL given by the user is not reachable an error message should be presented to a user. The message should contain the HTTP status code and a useful error description.

## Restrictions

1. The application should be written in Golang
2. The application must be put under git control
3. You can use whatever libraries/tools you want.

## Submission

Please provide the result as a git repo bundled with:

- A short text document that lists the main steps of building/deploying your solution as well as all assumptions/decisions you made in case of unclear requirements or missing information
- Suggestions on possible improvements of the application

Solution : Web-Page-Analyzer

Before You Read

- This is an MVP version of app for the problem statement
- Uses GIN framework, with Golang 1.20.3 (this is due to my system limitation)
- Unit tests are added for the service layer and core engine only (due to time limitations)
- No DI framework been used (just everything in plain go for now)
- Api versioning is not done

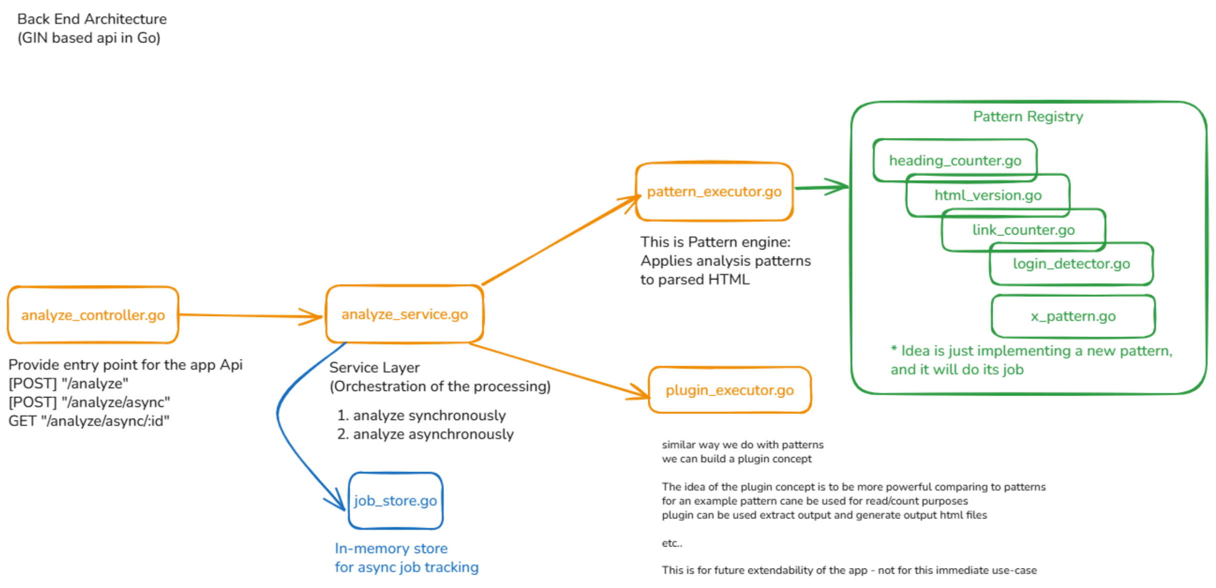
(It's not because I don't know about above but due to time constraints having limited time to evaluate everything)

Repo

https://github.com/Erandauh/web-page-analyzer

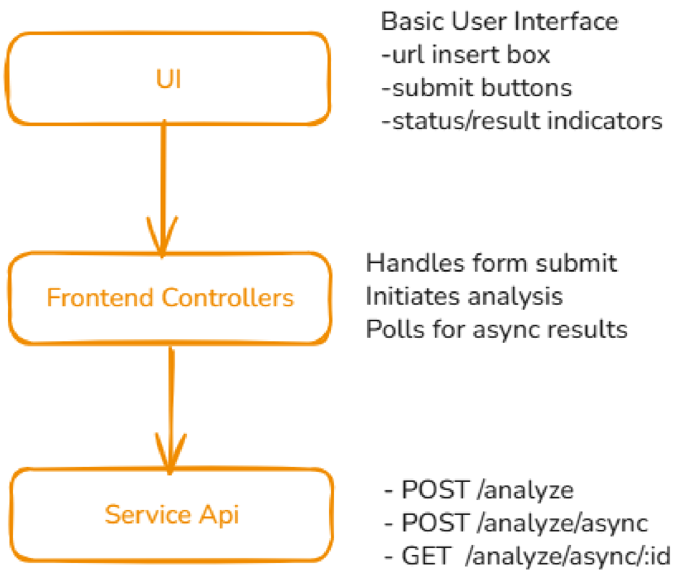
(It's public for now, please let me know once this evaluation is done, so I'll make it private)

Backend Architecture


Back End Architecture
(GIN based api in Go)
analyze_controller.go
Provide entry point for the app Api
[POST] "/analyze"
[POST] "/analyze/async"
GET "/analyze/async/:id"
analyze_service.go
Service Layer
(Orchestration of the processing)
1. analyze synchronously
2. analyze asynchronously
job_store.go
In-memory store
for async job tracking
pattern_executor.go
This is Pattern engine:
Applies analysis patterns
to parsed HTML
Pattern Registry
heading_counter.go
html_version.go
link_counter.go
login_detector.go
x_pattern.go
* Idea is just implementing a new pattern,
and it will do its job
plugin_executor.go
similar way we do with patterns
we can build a plugin concept
The idea of the plugin concept is to be more powerful comparing to patterns
for an example pattern cane be used for read/count purposes
plugin can be used extract output and generate output html files
etc..
This is for future extendability of the app - not for this immediate use-case

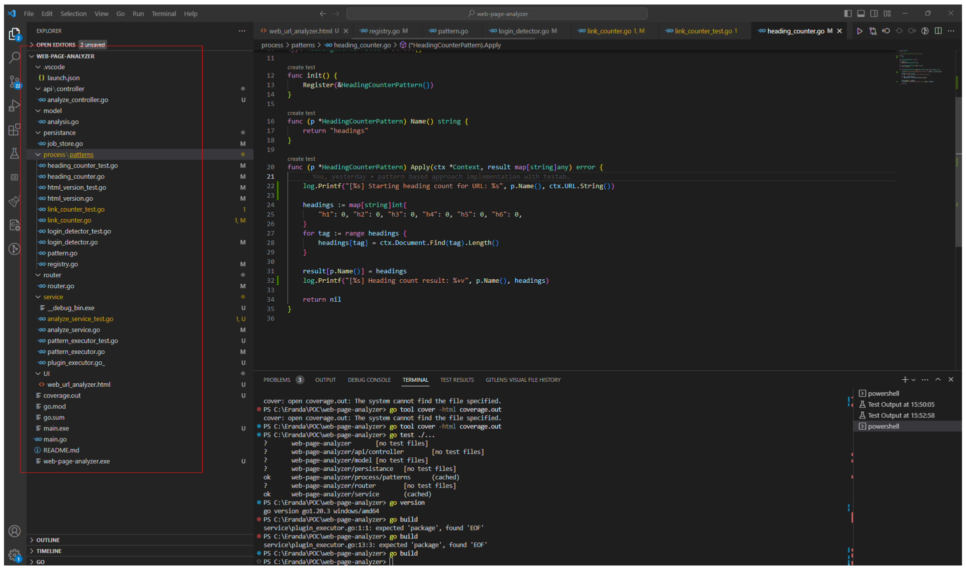
Frontend Architecture

(HTML5 Basic UI)



Development Environment

Windows Based PC
VSCode as IDE
Golong 1.20.3 (due to my system limitation)



```go
func init() {
	Register(&HeadingCounterPattern{})
}

func (p *HeadingCounterPattern) Name() string {
	return "headings"
}

func (p *HeadingCounterPattern) Apply(ctx *Context, result map[string]any) error {

	log.Printf("[%s] Starting heading count for URL: %s", p.Name(), ctx.URL.String())

	headings := map[string]int{
		"h1": 0, "h2": 0, "h3": 0, "h4": 0, "h5": 0, "h6": 0,
	}
	for tag := range headings {
		headings[tag] = ctx.Document.Find(tag).Length()
	}

	result[p.Name()] = headings
	log.Printf("[%s] Heading count result: %+v", p.Name(), headings)

	return nil
}
```

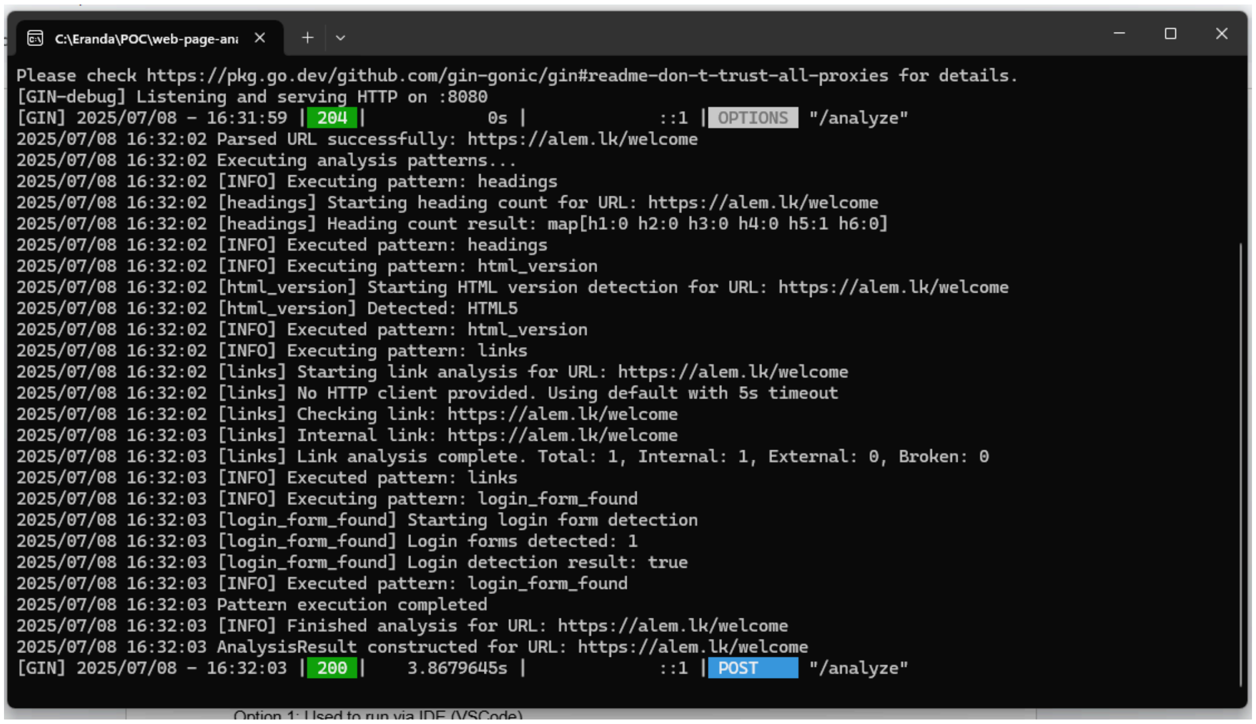
```
cover: open coverage.out: The system cannot find the file specified.
PS C:\Eranda\POC\web-page-analyzer> go tool cover -html coverage.out
cover: open coverage.out: The system cannot find the file specified.
PS C:\Eranda\POC\web-page-analyzer> go tool cover -html coverage.out
PS C:\Eranda\POC\web-page-analyzer> go test ./...
?       web-page-analyzer       [no test files]
?       web-page-analyzer/api/controller        [no test files]
?       web-page-analyzer/model [no test files]
?       web-page-analyzer/persistance   [no test files]
ok      web-page-analyzer/process/patterns      (cached)
?       web-page-analyzer/router        [no test files]
ok      web-page-analyzer/service       (cached)
PS C:\Eranda\POC\web-page-analyzer> go version
go version go1.20.3 windows/amd64
PS C:\Eranda\POC\web-page-analyzer> go build
service\plugin_executor.go:1:1: expected 'package', found 'EOF'
PS C:\Eranda\POC\web-page-analyzer> go build
service\plugin_executor.go:13:3: expected 'package', found 'EOF'
PS C:\Eranda\POC\web-page-analyzer> go build
PS C:\Eranda\POC\web-page-analyzer>
```

How to Run

<u>Running the BE Server</u>

Option 1: Used to run via IDE (VSCode)

Option 2: Executable exe "web-page-analyzer.exe" is also in the repo itself (if you are using a windows based PC)

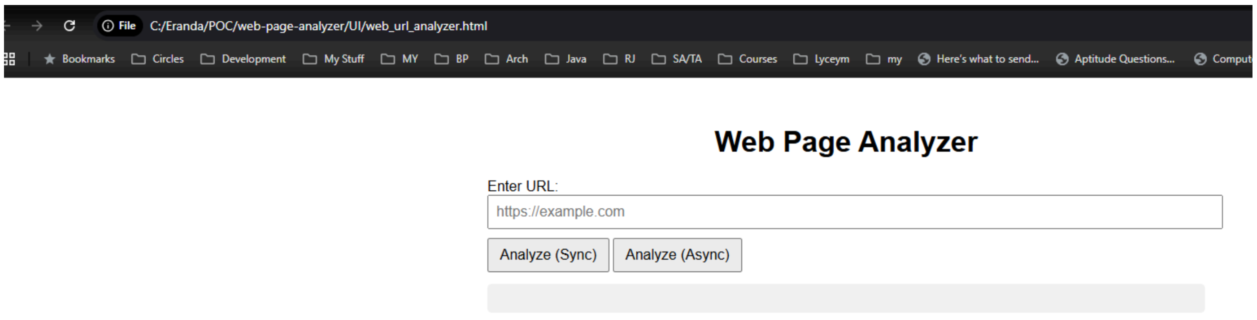


```
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on :8080
[GIN] 2025/07/08 - 16:31:59 | 204 |            0s |             ::1 | OPTIONS  "/analyze"
2025/07/08 16:32:02 Parsed URL successfully: https://alem.lk/welcome
2025/07/08 16:32:02 Executing analysis patterns...
2025/07/08 16:32:02 [INFO] Executing pattern: headings
2025/07/08 16:32:02 [headings] Starting heading count for URL: https://alem.lk/welcome
2025/07/08 16:32:02 [headings] Heading count result: map[h1:0 h2:0 h3:0 h4:0 h5:1 h6:0]
2025/07/08 16:32:02 [INFO] Executed pattern: headings
2025/07/08 16:32:02 [INFO] Executing pattern: html_version
2025/07/08 16:32:02 [html_version] Starting HTML version detection for URL: https://alem.lk/welcome
2025/07/08 16:32:02 [html_version] Detected: HTML5
2025/07/08 16:32:02 [INFO] Executed pattern: html_version
2025/07/08 16:32:02 [INFO] Executing pattern: links
2025/07/08 16:32:02 [links] Starting link analysis for URL: https://alem.lk/welcome
2025/07/08 16:32:02 [links] No HTTP client provided. Using default with 5s timeout
2025/07/08 16:32:02 [links] Checking link: https://alem.lk/welcome
2025/07/08 16:32:03 [links] Internal link: https://alem.lk/welcome
2025/07/08 16:32:03 [links] Link analysis complete. Total: 1, Internal: 1, External: 0, Broken: 0
2025/07/08 16:32:03 [INFO] Executed pattern: links
2025/07/08 16:32:03 [INFO] Executing pattern: login_form_found
2025/07/08 16:32:03 [login_form_found] Starting login form detection
2025/07/08 16:32:03 [login_form_found] Login forms detected: 1
2025/07/08 16:32:03 [login_form_found] Login detection result: true
2025/07/08 16:32:03 [INFO] Executed pattern: login_form_found
2025/07/08 16:32:03 Pattern execution completed
2025/07/08 16:32:03 [INFO] Finished analysis for URL: https://alem.lk/welcome
2025/07/08 16:32:03 AnalysisResult constructed for URL: https://alem.lk/welcome
[GIN] 2025/07/08 - 16:32:03 | 200 |     3.8679645s |             ::1 | POST     "/analyze"
```

<u>Running the FE</u>

Go to Dir
\web-page-analyzer\UI

Just double click and open the “web_url_analyzer.html” in any browser
(its basic JS and HTML, so it should work on any browser, out-of-the-box)


File C:/Eranda/POC/web-page-analyzer/UI/web_url_analyzer.html
Bookmarks Circles Development My Stuff MY BP Arch Java RJ SA/TA Courses Lyceym my Here's what to send... Aptitude Questions... Comput

Web Page Analyzer

Enter URL: https://example.com

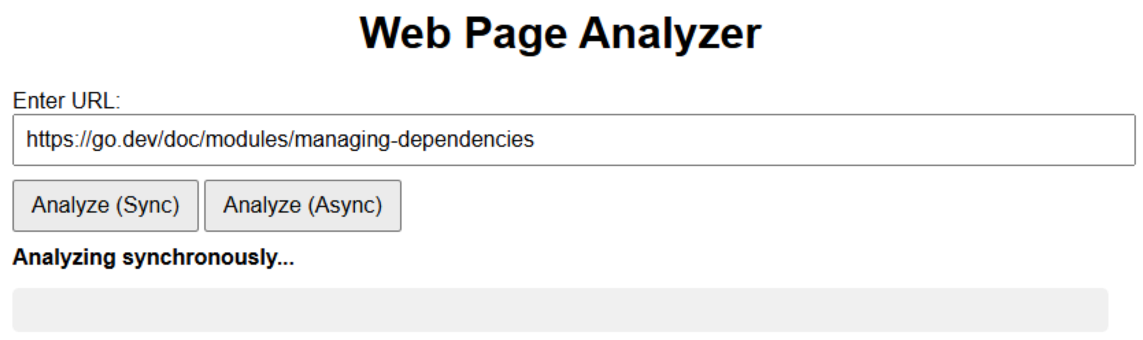
Analyze (Sync)
Analyze (Async)

Working App

Sync Analysis

Analyze synchronously (this will take a bit of a time, depending on the webpage complexity)

Start:

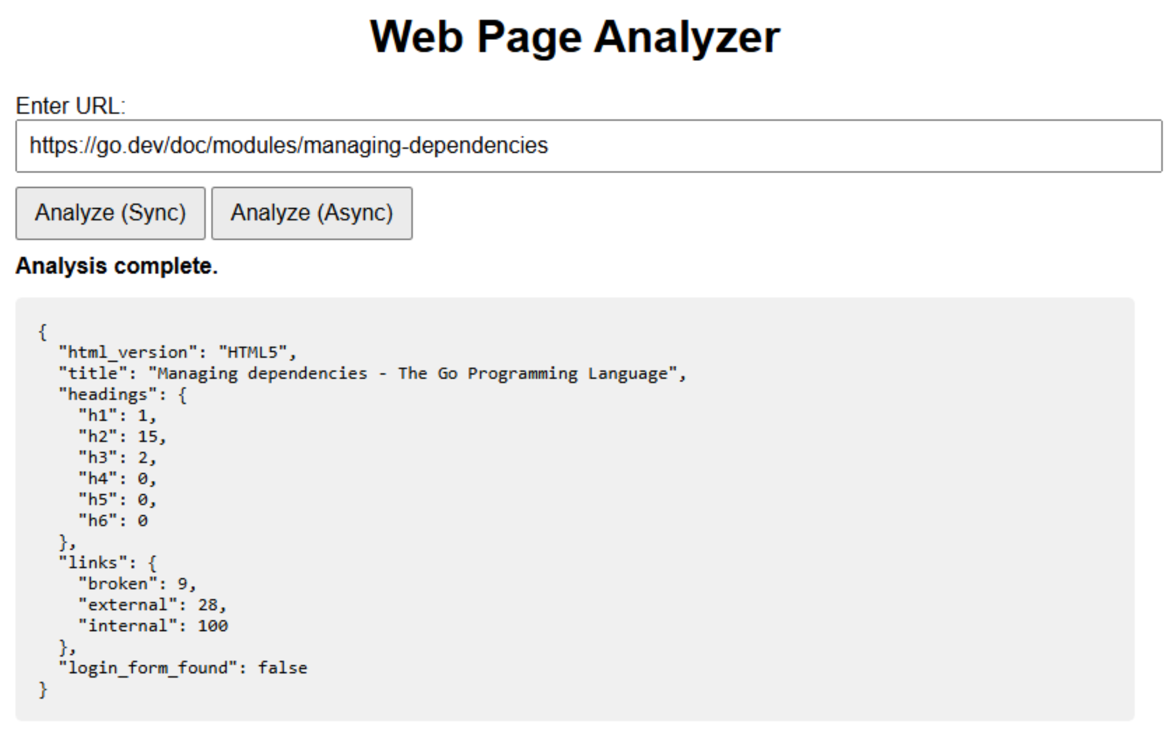


Web Page Analyzer

Enter URL: https://go.dev/doc/modules/managing-dependencies

Analyzing synchronously...

Results display:



Analysis complete.

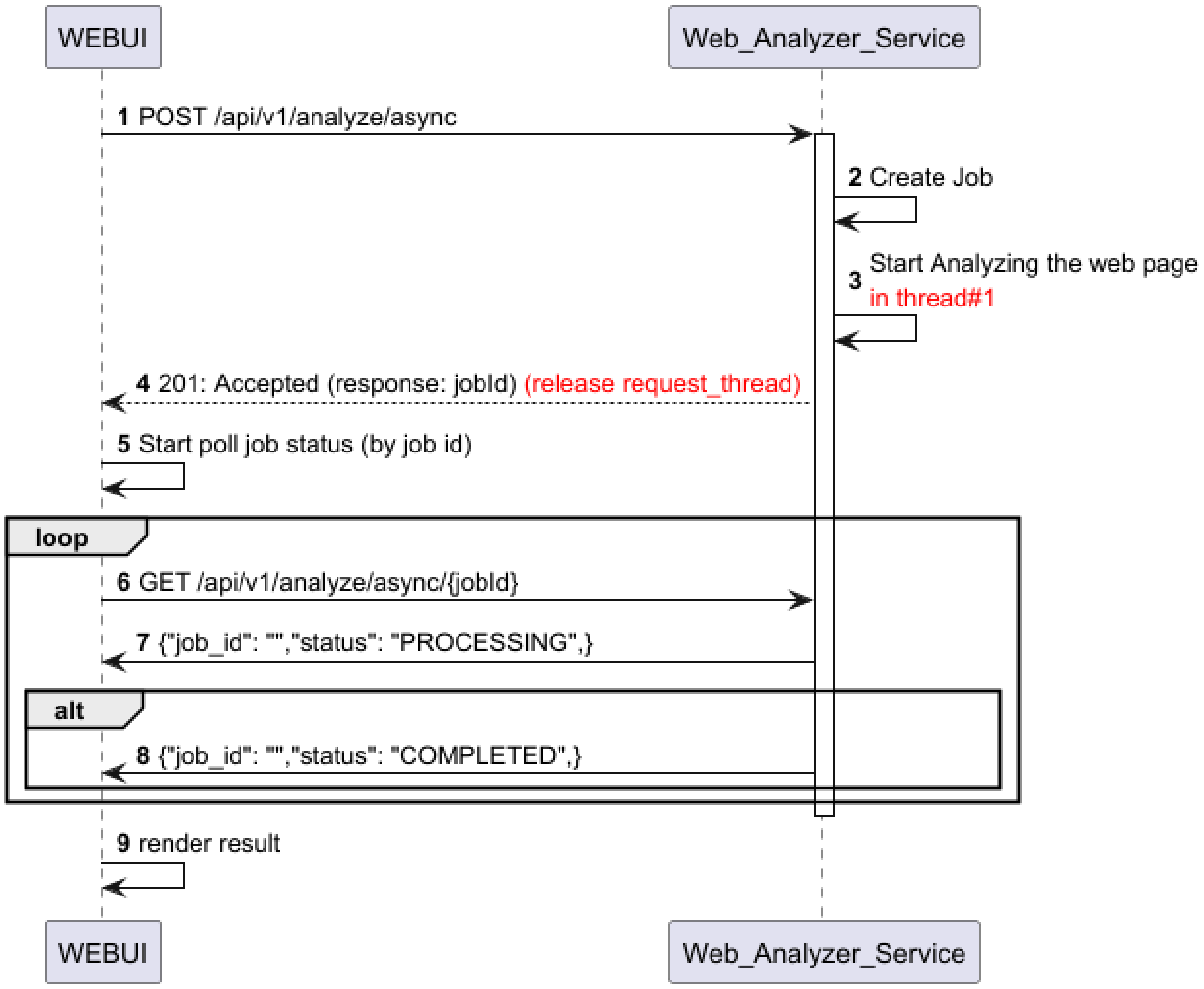
```
{
  "html_version": "HTML5",
  "title": "Managing dependencies - The Go Programming Language",
  "headings": {
    "h1": 1,
    "h2": 15,
    "h3": 2,
    "h4": 0,
    "h5": 0,
    "h6": 0
  },
  "links": {
    "broken": 9,
    "external": 28,
    "internal": 100
  },
  "login_form_found": false
}
```

TO DO: display time for analysis, so this becomes handy!

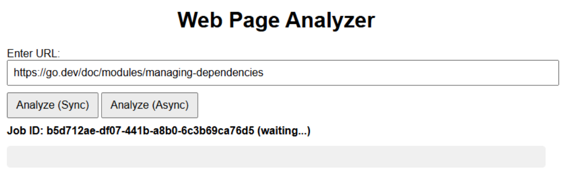
Async Analysis

In real world scenario, we should use async way of analyzing web pages as this HTML parse and evaluate is a time-consuming process

See the sequence:

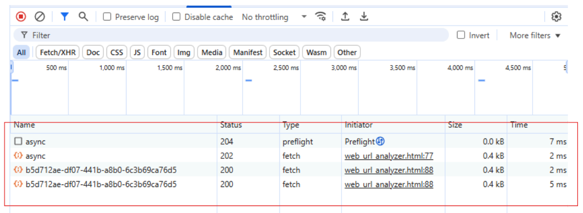

WEBUI
Web_Analyzer_Service
1 POST /api/v1/analyze/async
2 Create Job
3 Start Analyzing the web page in thread#1
4 201: Accepted (response: jobId) (release request_thread)
5 Start poll job status (by job id)
loop
6 GET /api/v1/analyze/async/{jobId}
7 {"job_id": "","status": "PROCESSING",}
alt
8 {"job_id": "","status": "COMPLETED",}
9 render result
WEBUI
Web_Analyzer_Service

Start: Returns the ‘JobId’ and continue processing



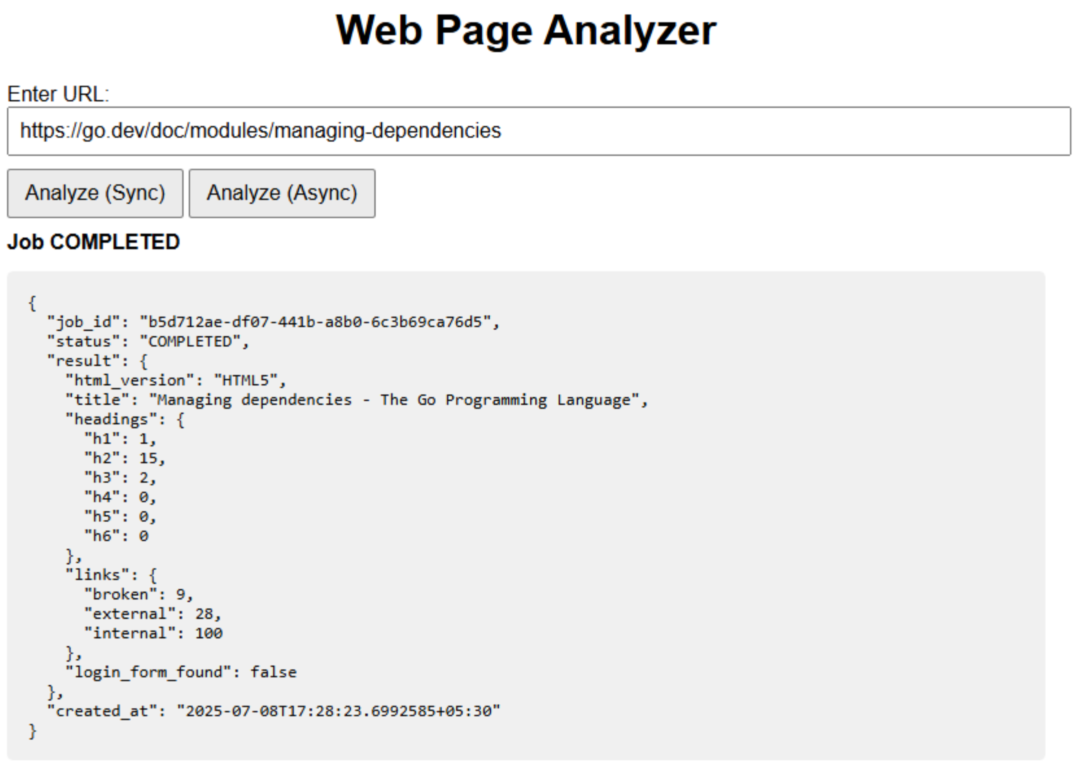
https://go.dev/doc/modules/managing-dependencies

Job ID: b5d712ae-df07-441b-a8b0-6c3b69ca76d5 (waiting...)



Name	Status	Type	Initiator	Size	Time
async	204	preflight	Preflight	0.0 kB	7 ms
async	202	fetch	web_url_analyzer.html:77	0.4 kB	2 ms
b5d712ae-df07-441b-a8b0-6c3b69ca76d5	200	fetch	web_url_analyzer.html:88	0.4 kB	2 ms
b5d712ae-df07-441b-a8b0-6c3b69ca76d5	200	fetch	web_url_analyzer.html:88	0.4 kB	5 ms

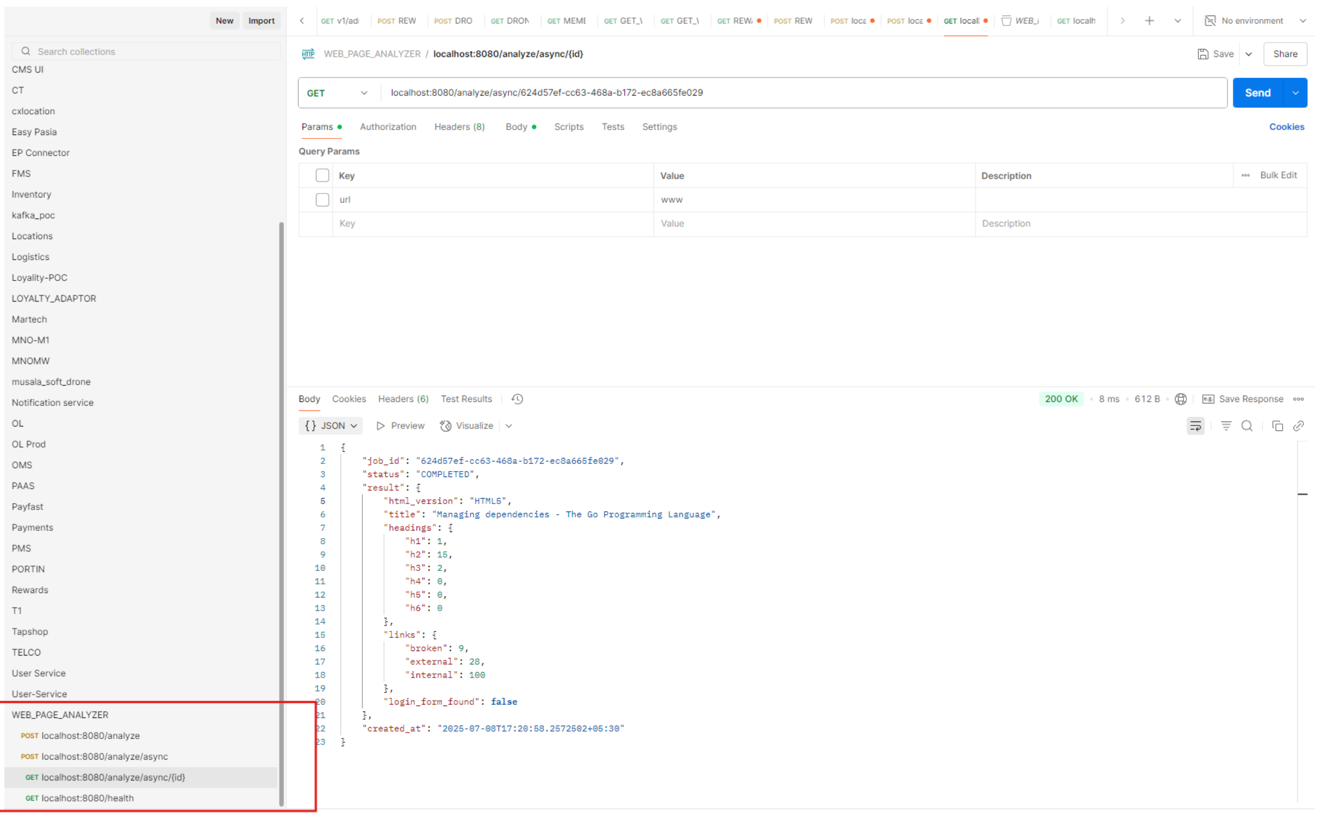
Results display:



```
{
  "job_id": "b5d712ae-df07-441b-a8b0-6c3b69ca76d5",
  "status": "COMPLETED",
  "result": {
    "html_version": "HTML5",
    "title": "Managing dependencies - The Go Programming Language",
    "headings": {
      "h1": 1,
      "h2": 15,
      "h3": 2,
      "h4": 0,
      "h5": 0,
      "h6": 0
    },
    "links": {
      "broken": 9,
      "external": 28,
      "internal": 100
    },
    "login_form_found": false
  },
  "created_at": "2025-07-08T17:28:23.6992585+05:30"
}
```

Back End Api Collection

There are three main endpoints (excluding the health)



WEB_PAGE_ANALYZER / localhost:8080/analyze/async/{id}

GET localhost:8080/analyze/async/624d57ef-cc63-468a-b172-ec8a665fe029

Query Params

Key	Value	Description
url	www	
Key	Value	Description

Body {} JSON

```json
{
    "job_id": "624d57ef-cc63-468a-b172-ec8a665fe029",
    "status": "COMPLETED",
    "result": {
        "html_version": "HTML5",
        "title": "Managing dependencies - The Go Programming Language",
        "headings": {
            "h1": 1,
            "h2": 15,
            "h3": 2,
            "h4": 0,
            "h5": 0,
            "h6": 0
        },
        "links": {
            "broken": 9,
            "external": 28,
            "internal": 100
        },
        "login_form_found": false
    },
    "created_at": "2025-07-08T17:20:58.2572502+05:30"
}
```

[POST] /analyze
Perform **synchronous analysis** of a provided web page URL
Behavior:

- Parses the HTML document.
- Applies multiple analysis patterns (HTML version, headings, links, login form).
- Returns the complete result immediately.

Use Case: Suitable for fast analysis of user interaction in UI.

Request	Response

```
{
    "url":
"https://go.dev/doc/modules/managing-dependencies"
}
```
	```
	{
	    "html_version": "HTML5",
	    "title": "Managing dependencies - The Go Programming Language",
	    "headings": {
	        "h1": 1,
	        "h2": 15,
	        "h3": 2,
	        "h4": 0,
	        "h5": 0,
	```

	`"h6": 0`
	`},`
	`"links": {`
	`"broken": 9,`
	`"external": 28,`
	`"internal": 100`
	`},`
	`"login_form_found": false`
	`}`

[POST] /analyze/async
Initiates **asynchronous analysis** of a given URL
Behavior:

- Creates a job with a unique job ID.
- Starts analysis in a background goroutine.
- Immediately returns a job ID to the client.

Use Case: For long-running analysis or UI polling scenarios.

Request	Response

`{`
`"url":`
`"https://go.dev/doc/modules/managing-dependencies"`
`}`
	`{`
	`"job_id":`
	`"624d57ef-cc63-468a-b172-ec8a665fe029",`
	`"status": "PROCESSING",`
	`"created_at":`
	`"2025-07-08T17:20:58.2572502+05:30"`
	`}`

[GET] /analyze/async/:id
Fetch the **status or result** of an async analysis job
Behavior:

- Checks if the job exists and its current status.
- Returns job details with result if available.

Use Case: For long-running analysis or UI polling scenarios.

Request	Response
../analyze/async/624d57ef-cc63-468a-b172-ec8a665fe029	`{ "job_id": "624d57ef-cc63-468a-b172-ec8a665fe029", "status": "PROCESSING", "created_at": "2025-07-08T17:20:58.2572502+05:30" }` OR `{ "job_id": "624d57ef-cc63-468a-b172-ec8a665fe029", "status": "COMPLETED", "result": { "html_version": "HTML5", "title": "Managing dependencies - The Go Programming Language", "headings": { "h1": 1, "h2": 15, "h3": 2, "h4": 0, "h5": 0, "h6": 0 }, "links": { "broken": 9, "external": 28, "internal": 100 }, "login_form_found": false }, "created_at": "2025-07-08T17:20:58.2572502+05:30" }` OR `{ "job_id":`

	"624d57ef-cc63-468a-b172-ec8a665fe029", "status": "FAILED", "created_at": "2025-07-08T17:20:58.2572502+05:30", “error”: “Html parse error!” }

[GET] /health

Health check endpoint to verify that the backend server is running.
Use case: Used by monitoring tools, load balancers, or during deployments.

Request	Response
../health	{ "status": "ok" }

Curl Commands

/analyze	curl --location 'localhost:8080/analyze' \ --header 'Content-Type: application/json' \ --data '{ "url": "https://go.dev/doc/modules/managing-dependencies" }'
/analyze/async	curl --location 'localhost:8080/analyze/async' \ --header 'Content-Type: application/json' \ --data '{ "url": "https://go.dev/doc/modules/managing-dependencies" }'
/analyze/async/id	curl --location 'localhost:8080/analyze/async/624d57ef-cc63-468a-b172-ec8a665fe029' \

	--data ''
/health	curl --location 'localhost:8080/health'

Postman Collection here