Problem Statement **1**
Solution : Web-Page-Analyzer **3**
Before You Read 3
Repo 3
Backend Architecture 4
Frontend Architecture 4
Development Environment 5
How to Run 5
Working App 7
Swagger Link 7
Sync Analysis 7
Async Analysis 10
Application Logs 12
Back End Api Collection 13
Curl Commands 17
Future Improvements 17
Functional Improvements 17
Codebase Improvements 18

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)

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

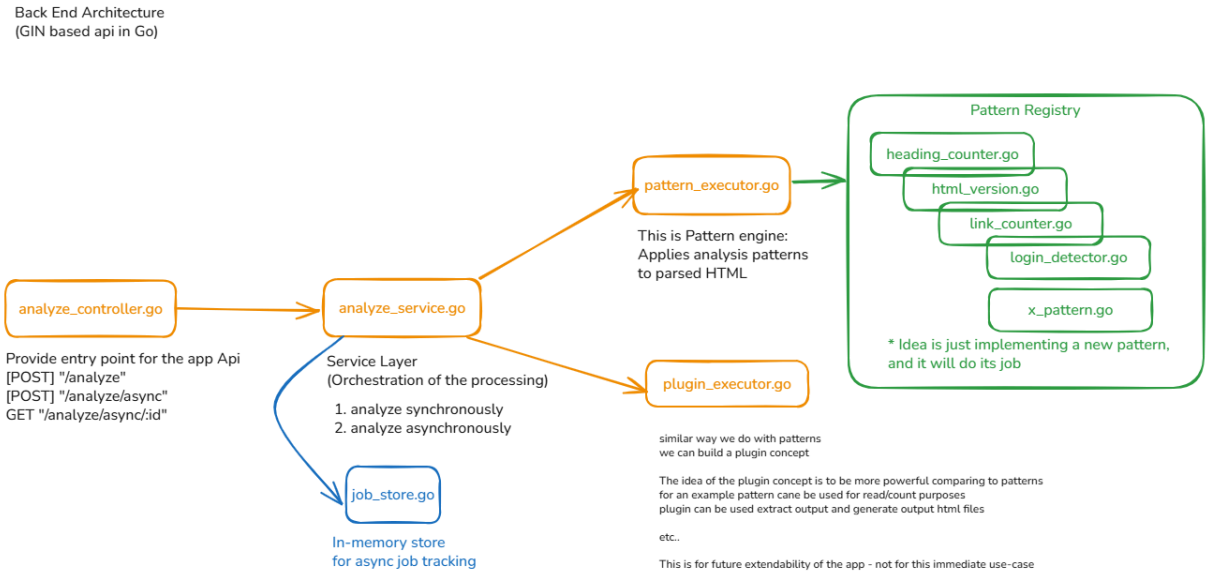
Refer : Future Improvements
https://docs.google.com/document/d/1xfNCxaxOy31HIbkfKR4ZJKKHwKZMReVIBrxqa1vcY-U/edit?tab=t.0#heading=h.e83i5rjshrrq

Repo

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

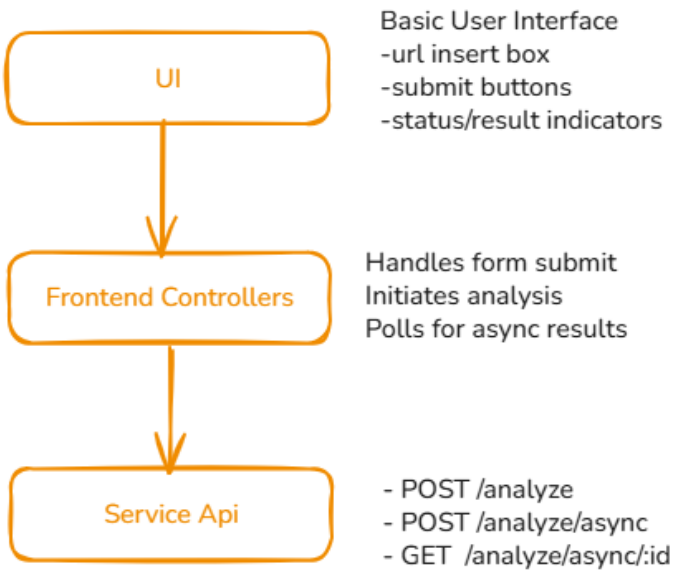
(It's public for now, please let me know once this evaluation is done, so I’ll make it private)

Backend Architecture



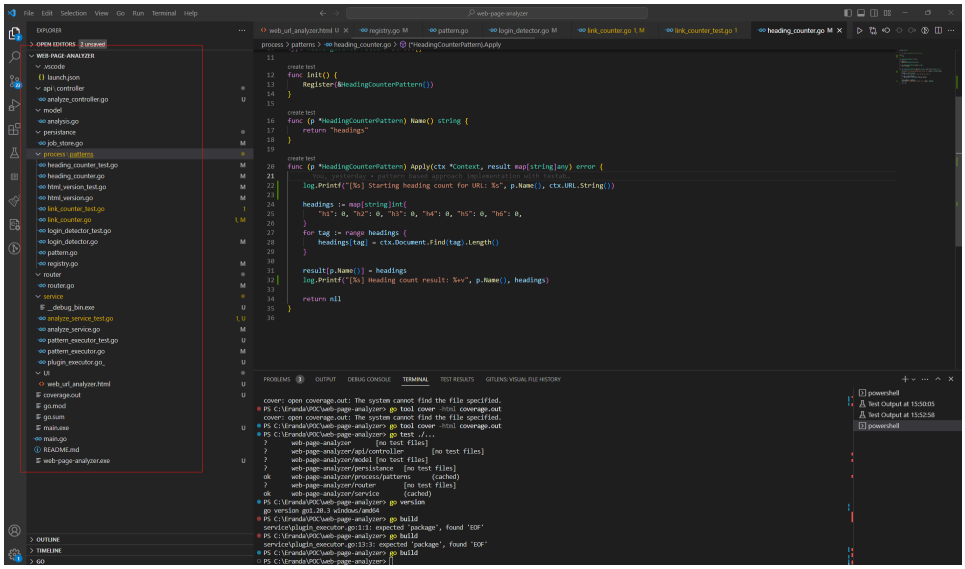
Frontend Architecture

(HTML5 Basic UI)



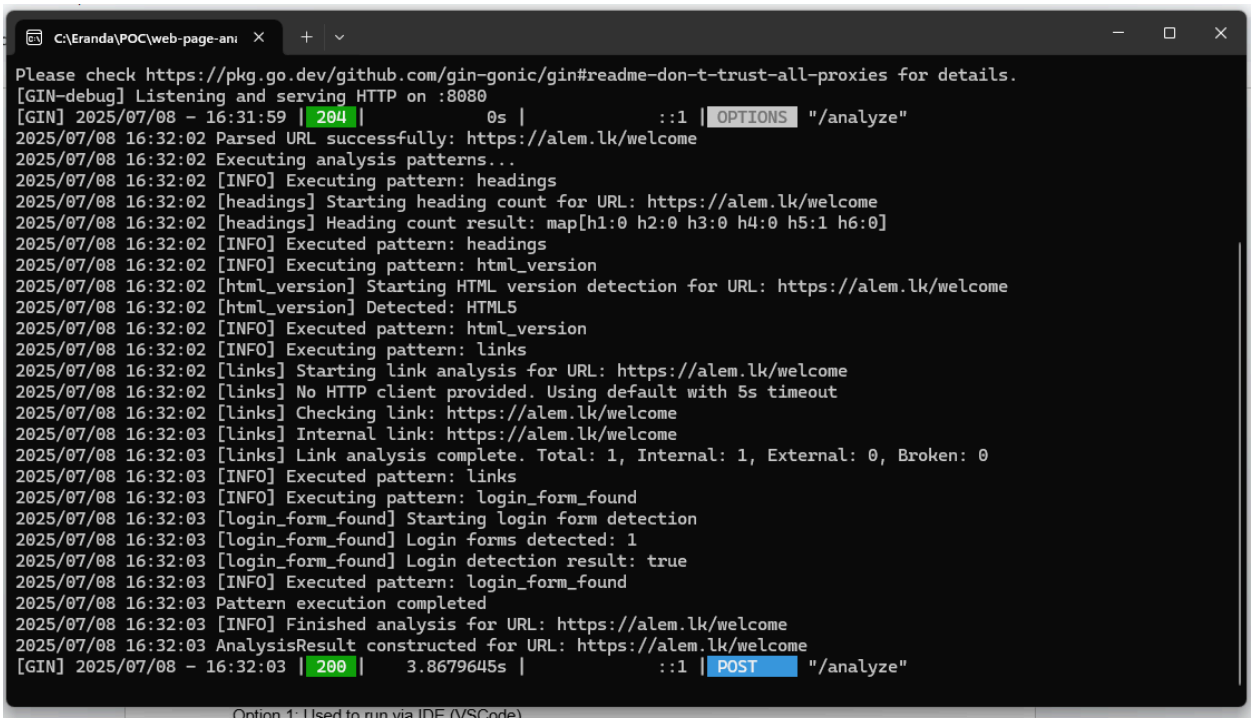
Development Environment

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



How to Run

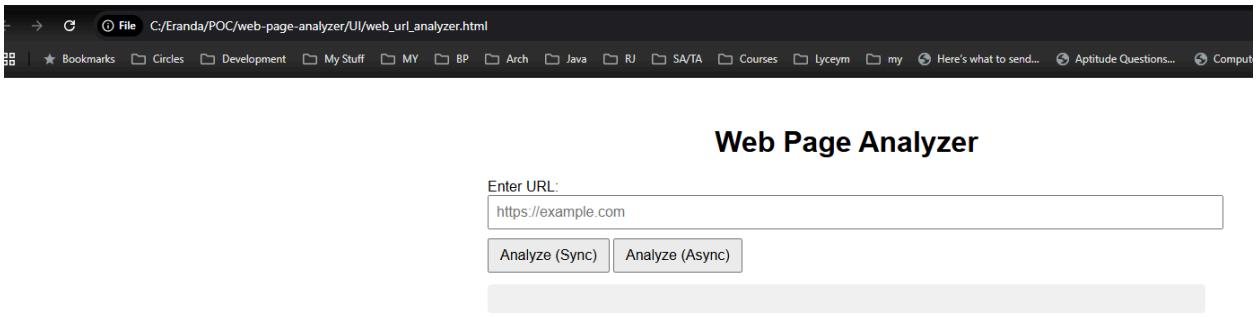
<u>Running the BE Server</u>
Option 1: Used to run via IDE (VSCode)
Option 2: Executable exe “web-page-analyzer.exe” is also in the repo itself (if you are using a windows based PC)



Running the FE

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)

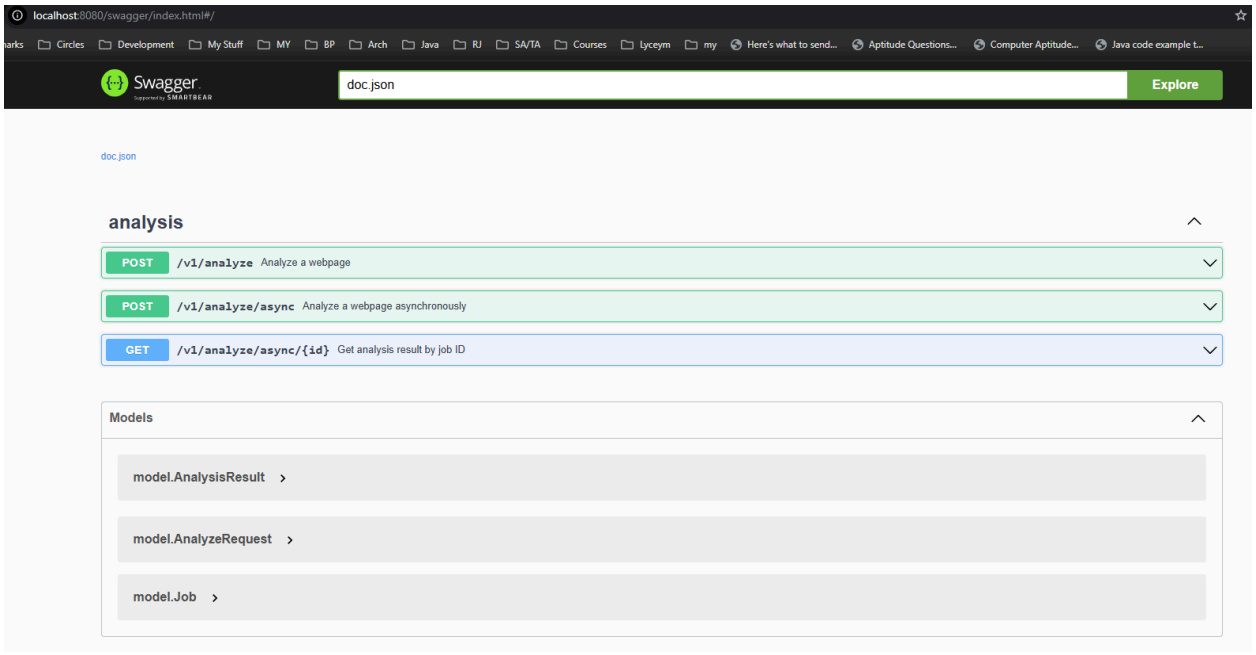


Working App

localhost:8080/health

Swagger Link

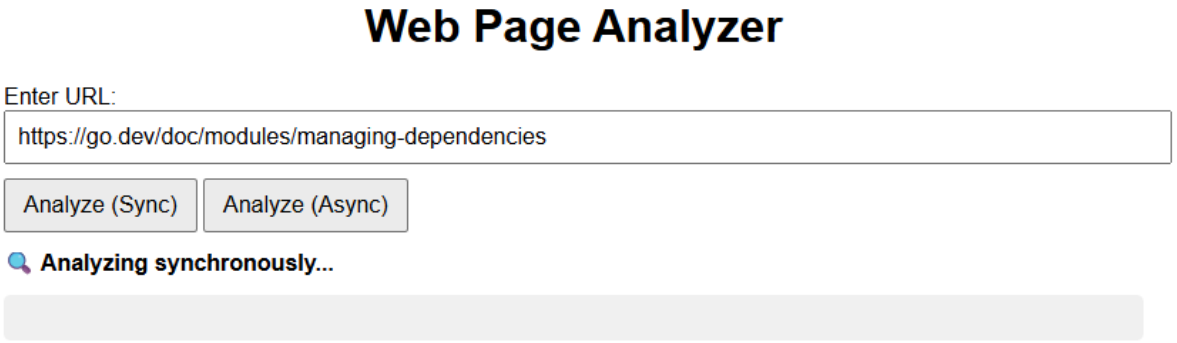
http://localhost:8080/swagger/index.html#/



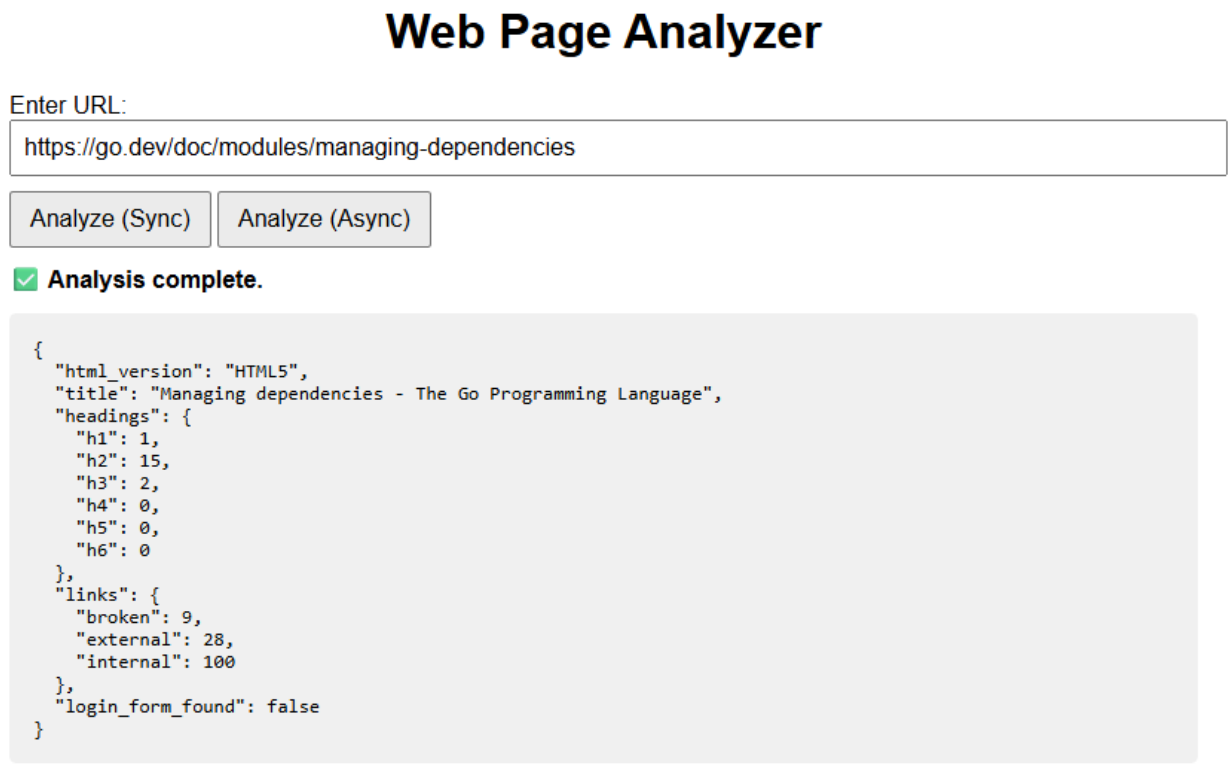
Sync Analysis

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

Start:

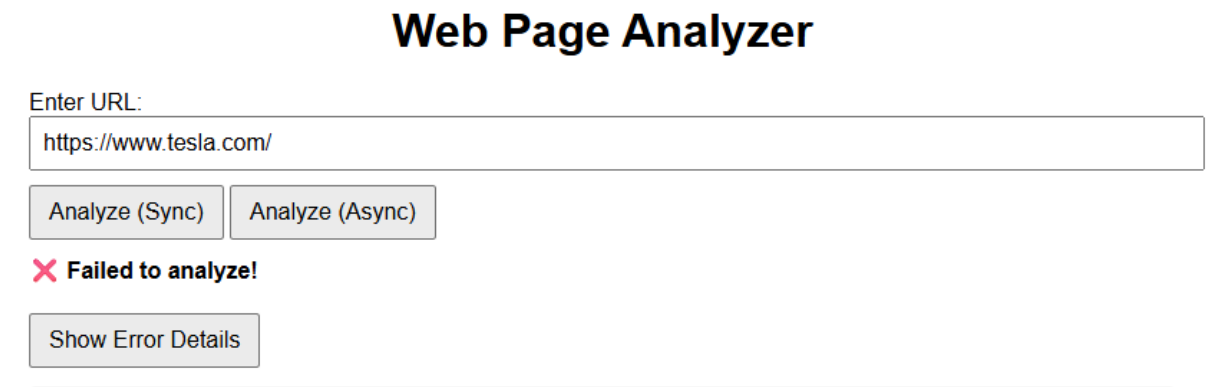


Results:



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

Error:



Web Page Analyzer

Enter URL:

https://www.tesla.com/

Analyze (Sync) Analyze (Async)

❌ **Failed to analyze!**

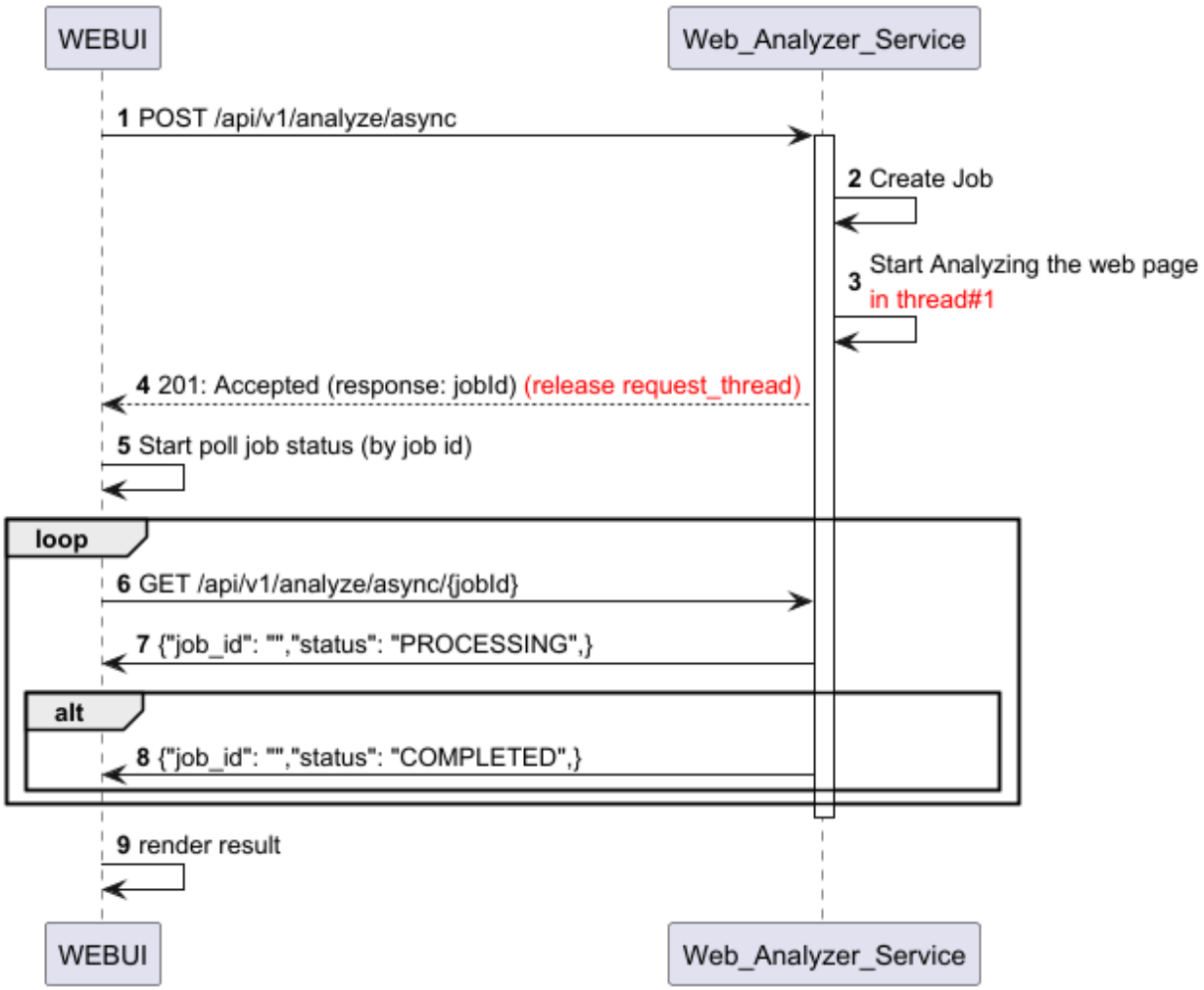
Hide Error Details


```
{
  "error": "Get \"https://www.tesla.com/\": stream error: stream ID 1; INTERNAL_ERROR; received from peer"
}
```

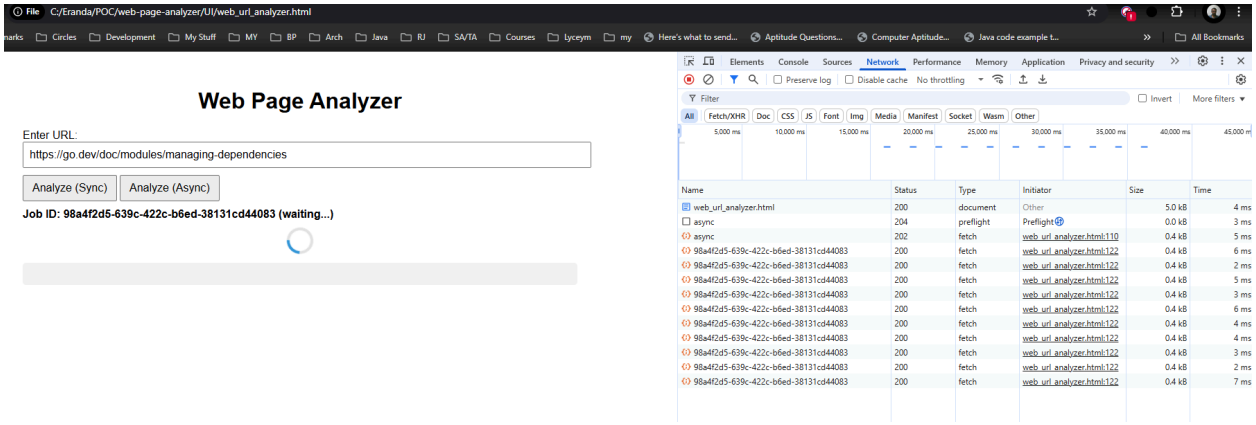
Async Analysis

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

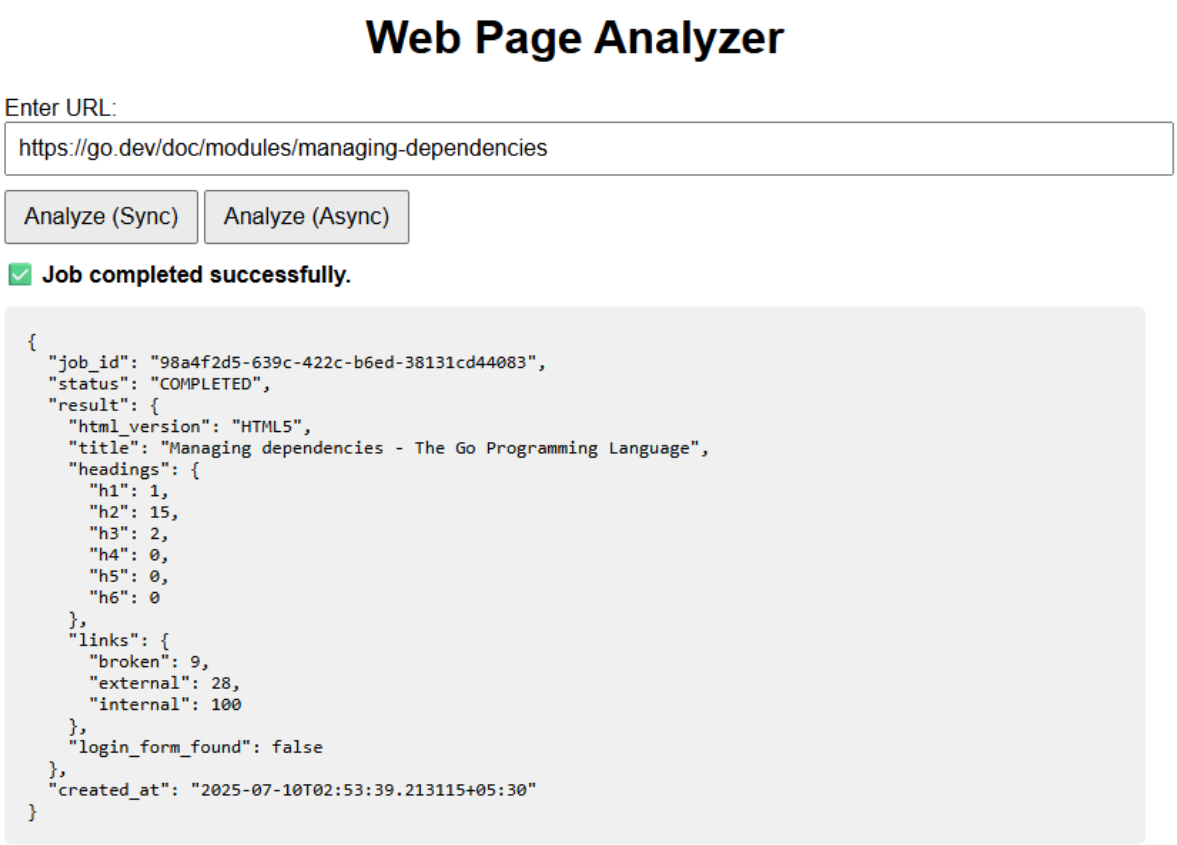
See the sequence:



Start: Returns the 'JobId' and continue processing


File C:/Eranda/POC/web-page-analyzer/UI/web_url_analyzer.html
Web Page Analyzer
Enter URL:
https://go.dev/doc/modules/managing-dependencies
Analyze (Sync)
Analyze (Async)
Job ID: 98a4f2d5-639c-422c-b6ed-38131cd44083 (waiting...)
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Name
Status
Type
Initiator
Size
Time
web_url_analyzer.html
async
98a4f2d5-639c-422c-b6ed-38131cd44083

Results:


Web Page Analyzer
Enter URL:
https://go.dev/doc/modules/managing-dependencies
Analyze (Sync)
Analyze (Async)
Job completed successfully.
{
"job_id": "98a4f2d5-639c-422c-b6ed-38131cd44083",
"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": "2025-07-10T02:53:39.213115+05:30"
}

Error:

Web Page Analyzer

Enter URL:

https://www.tesla.com/

Analyze (Sync) | Analyze (Async)

❌ **Job failed!**

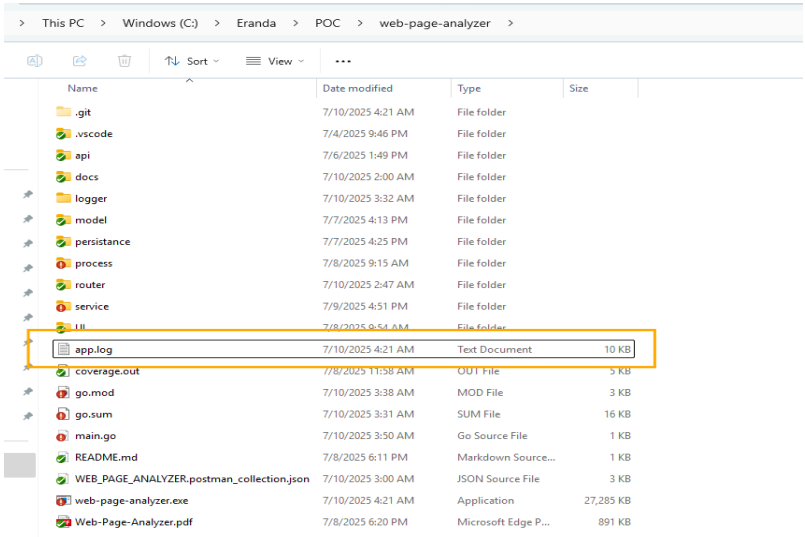
Hide Error Details

```
Get "https://www.tesla.com/": stream error: stream ID 1; INTERNAL_ERROR; received from peer
```


```
{
  "job_id": "24255cc0-b783-486b-96aa-c3ecb3c412a2",
  "status": "FAILED",
  "error": "Get \"https://www.tesla.com/\": stream error: stream ID 1; INTERNAL_ERROR; received from peer",
  "created_at": "2025-07-10T03:12:51.5734907+05:30"
}
```

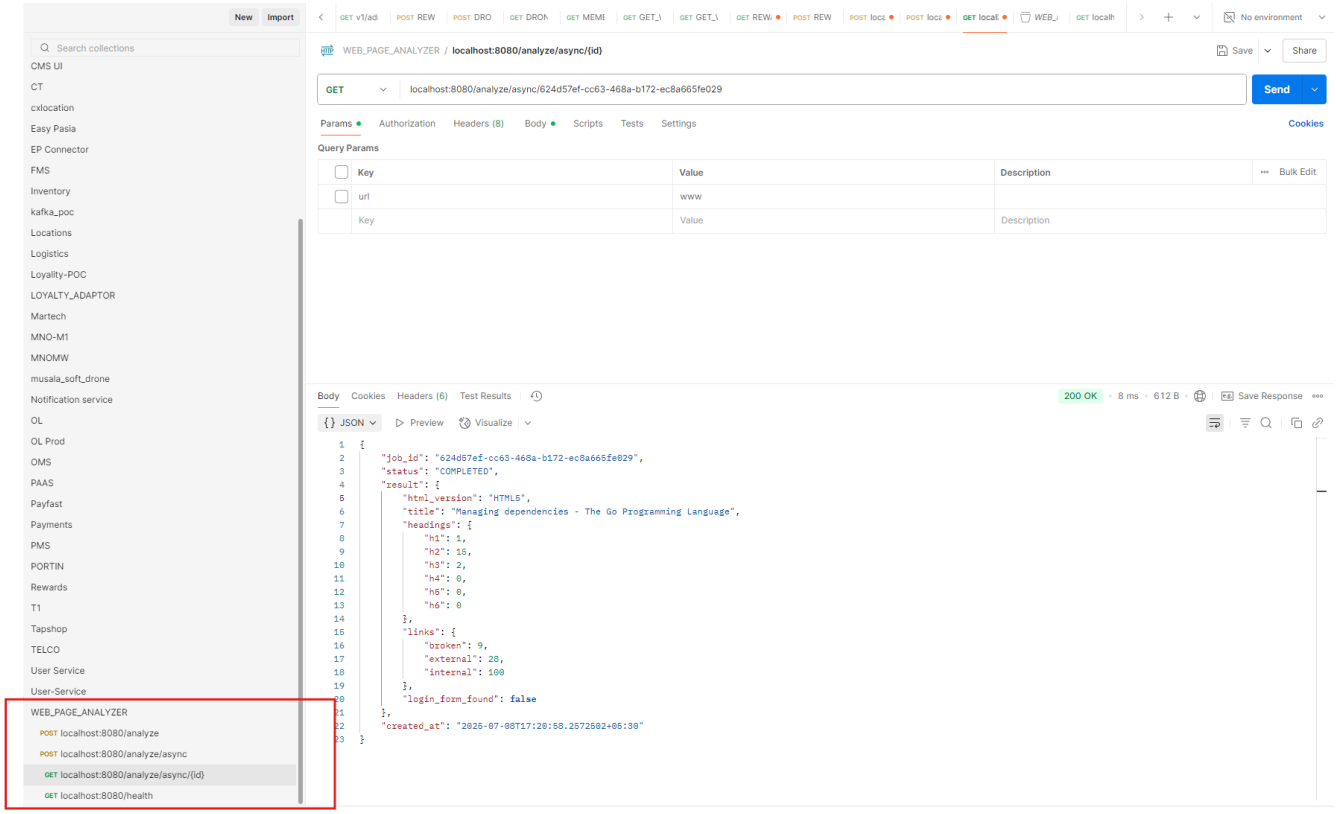

Application Logs

Log file is generated in the app folder with name {*app.log*}. Logrus (pluggable logging framework) is used for logging



Back End Api Collection

There are three main endpoints (excluding the health)



[POST] /v1/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

<pre>{
"url": "https://go.dev/doc/modules/managing-dependencies"
}</pre>
	<pre>{
	"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
	}</pre>

[POST] /v1/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

<pre>{
"url": "https://go.dev/doc/modules/managing-dependencies"
}</pre>
	<pre>{
	"job_id": "624d57ef-cc63-468a-b172-ec8a665fe029",
	"status": "PROCESSING",
	"created_at":</pre>

	"2025-07-08T17:20:58.2572502+05:30" }

[GET] /v1/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

v1/analyze	curl --location 'localhost:8080/v1/analyze' \ --header 'Content-Type: application/json' \ --data '{ "url": "https://go.dev/doc/modules/managing-dependencies" }'
v1/analyze/async	curl --location 'localhost:8080/v1/analyze/async' \ --header 'Content-Type: application/json' \ --data '{ "url": "https://go.dev/doc/modules/managing-dependencies" }'
v1/analyze/async/id	curl --location 'localhost:8080/v1/analyze/async/624d57ef-cc63-468a-b172-ec8a665fe029' \ --data ''
/health	curl --location 'localhost:8080/health'

Postman Collection here: https://github.com/Erandauh/web-page-analyzer/blob/main/WEB_PAGE_ANALYZER.postman_collection.json

Future Improvements

Functional Improvements

1. **Include Caching support for Frequent URLs**
   Cache analyzed results with TTL (e.g., with Redis) to reduce redundant processing overhead.

2. **Support Batch URL Analysis**
   Accept a list of URLs for batch analysis in one request.

3. **Persist Async Jobs (Support Multipods)**
   Replace in-memory store with persistent and centralized job store (e.g., PostgreSQL, Redis).

4. **Introduce Cleanup process/Job**

5. **Add elapsed time as a response attribute** (currently its in application logs)

6. **Add More Patterns**
   e.g., Image analysis

   Allow Uploading HTML Files
   Add endpoint for uploading raw HTML files for offline page analysis.

Codebase Improvements

1. **Introduce Dependency Injection (DI):**

Use a DI framework for better testing and decoupling of dependencies
(need to explore more : Is it Google wire, the best choice??)