TECHNISCHE UNIVERSITÄT
CHEMNITZ

Department of Computer Science

Professorship of Data Management Systems

Database and Web Techniques SS2020

Proxy Management System

Jeevan Srinivasalureddy (616476) (Web Engineering)

Sweta (616463) (Web Engineering)

Table of Contents

1 Contributors

Section	Author(s)
2 Introduction to Task	Sweta
3 Tech Stack	Sweta
4 Project Architecture	Sweta
5 Test Architecture	Jeevan Srinivasalureddy
6 Project Workflow	Jeevan Srinivasalureddy
7 Requirement vs Implementation	Jeevan Srinivasalureddy
Appendix A	Sweta and Jeevan Srinivasalureddy

2 Introduction to Task

The main aim of this project is to have a proxy management application which aggregates the proxies from different sources and provides a graphical user interface to display the proxies as well testing and updating of the proxies. Considering the scope of the tasks, application has been restricted to work with only five proxy providers and corresponding proxies from them. Overall to provide a complete proxy list management for users.

2.1 Backend

The backend aggregates the proxies from various sources, processes them based on the parsing extraction logics, store data to the database, handles the update, test functionalities, delete and remove the proxies after certain threshold time if certain conditions are met.

2.2 Database

Stores the proxy related information in multiple tables and REST API's are exposed to query data from the database and display it in the frontend.

2.3 API Interface

API interface acts as middleware between the frontend and backend. It exposes all the critical HTTP methods such as GET, POST, DELETE, PUT to handle the data between frontend and backend

2.4 Frontend

User interface to display all the information related to proxy providers, extracted proxies and critical application actions such as add, update, delete, test proxy can be performed.

Update and clean up information can be set from the frontend. Users can filter, export and handle the complete proxy management operations from the front end.

3 Tech Stack

There are various open source tools and libraries from which task can be solved in different approaches. Considering the functionalities of the libraries, open community to seek solutions, ease of implementation, complexity of the task - following technologies, have been used.

3.1 Backend - NodeJS

NodeJS is an asynchronous event-driven JavaScript runtime environment. It is most common server-side scripting library and is well known for its open source contributions, libraries, well architected package managers and mainly operates on single thread allowing it to support many concurrent connections.

Since its very lightweight, availability of many node packages and its extensive online community, core backend to solve the task is implemented using NodeJS

3.1.1 Libraries

- Express JS: Web application framework for NodeJS. It's the de facto standard server framework for NodeJS. It's been used because of its minimalistic configurations with many possible functionalities
- Node-cron: Schedule cron jobs in NodeJS
- Request: Handle http requests
- Cheerio: Scrapping tool for HTML pages
- Pg: Postgres node library to integrate the database
- Npm : Package Manager for nodeJS

3.2 Database - Postgres

Postgres is free and open source relation database management system, it adheres with Atomicity, Consistency, Isolation, Durability (ACID) properties for all the transactions. It is designed to handle multiple workloads with many high availability features.

One of the main reasons to select Postgres is, as it was introduced during Advanced Management Data course in WS2019.

3.3 API Interface - REST

Representational state transfer is a lightweight architectural style for creating web services. It is widely known for its stateless protocol, fast performance, reliability and reusability. It can easily be implemented with ExpressJS Web framework.

3.4 Frontend – HTML, JavaScript, Bootstrap

HTML is a standard mark-up language to display documents in the browser. This can be easily integrated with CSS and JavaScript frameworks. Though there are many popular frontend frameworks such as AngularJS, ReactJS, Vue JS etc. This stack is used as it was easy to implement considering the scope of the task and time constraints.

Bootstrap is responsive frontend CSS framework which has both CSS and JavaScript templates for all the HTML elements.

3.5 Tech Stack in Lecture Context

In the course lecture various technologies were discussed such as JDBC, JSP, SAX, DOM, Xpath, JSON Web Services. But to solve this task only concepts related to JSON Web Services are used considering the choice of tech stack

4 Project Architecture

Considering the above use of technologies, complete overview of the architecture of the project is illustrated below.

4.1 Database

There are 5 tables from which all the data for this task has been handled.

- **proxy**: List of all the proxy records are maintained here with proxy specific details such as `ip , port , last_test_success, last_found_in_list, first_found_list, proxy_list_id, test_result, anonimity where proxy_id is PK and proxy_list_id is FK`
- **proxy_list**: List of all the proxy providers are maintained with details such as `url, extraction_type, last_success_update, update_time, age where proxy_list_id is PK`
- **update_summary**: List of the update operations for various proxy providers with details such as `proxy_list, start_time, status where update_summary_id is PK`
- **test_summary**: List of the test operation for various proxy providers with details such as `test_url , proxy_list, start_time, test_status where test_summary_id is PK`
- **test_url**: List of all the available test url's with details such as `test_url where test_url_id is PK`

4.2 REST API's

Multiple API's are designed to integrate the frontend and backend. These interfaces play an important role in the data flow. All the API's are developed using ExpressJS libraries for Web Development. Detailed information is provided in Appendix A. Summary of all the API's and its corresponding backend implementation is as below:

```
app.get("/proxyList", db.getProxyList);
app.get("/proxy", db.getProxy);
app.get("/proxyList/:proxy_list_id",
db.getProxyListById); app.get("/proxy/:proxy_id",
db.getProxyById); app.post("/proxy", db.createProxy);
app.post("/proxyList", db.createProxyList);
app.delete("/proxyList/:proxy_list_id", db.deleteProxyList);
app.put("/testProxy",test.testProxyAndUpdateToDB)
app.post("/testURL",db.createTestURL)
app.get("/testURL",db.getTestURL)
app.delete("/testURL/:test_url_id",db.deleteTestURL)
app.get("/testSummary",test.getTestSummary)
app.get("/updateSummary",update.getUpdateSummary)
app.delete("/updateSummary/:update_summary_id",update.deleteUpdateSu
mmary)
app.delete("/testSummary/:test_summary_id",test.deleteTestSummary)
app.post("/updateProxyList",update.checkForUpdate)
app.put("/updateProxyList",update.upda2teProxyList)
```

4.3 Backend

Core backend for the project is expressJS framework processing the API requests, below are the JavaScript files handling all the operations for each of the API request.

- queries.js: Core backend implementations are handled with functions such as `getProxyList`, `getProxy`, `getProxyListById`, `getProxyById`, `createProxy`, `createproxyList`, `deleteProxyList`, `createTestURL`, `getTestURL`, `deleteTestURL`
- proxy-extraction.js: Extraction functionalities are implemented with functions to extract proxies from JSON and HTML sources such as `getProxiesFromProxyProviderInJSONAndInsertToDB` and `getProxiesFromProxyProviderInHTMLAndInsertToDB`
- proxy-test.js: Testing implementation logic to test proxies with functions `testProxyAndUpdateToDB`, `getTestSummary` and `deleteTestSummary`

- proxy-update.js: Update implementation logic to update proxies with functions `checkForUpdate`, `getUpdateSummary`, `deleteUpdateSummary`, `updateProxyList`

- proxy-delete-age.js: Delete the old proxies which have been failed since certain days with function `deleteOldProxies.` This is executed as a cron job for every 12 hours to check for any old entries.

```
cron.schedule("* 12 * * *", function() {
console.log("Running this Job for every 12 hours");
cleanOldAge.deleteOldProxies() });
```

4.4 Frontend

Frontend uses the standard technologies as described in above sections. Along with HTML and JavaScript, Bootstrap CSS framework is used to for responsive and design benefits. Critical HTML and JavaScript functionalities are discussed below:

4.4.1 HTML

Following HTML sections are implemented as per the task requirements.

- HTML form to add proxy providers with url, extraction type, update interval and age

- Dynamic HTML table to display proxy providers (table size changes as and when new data is added) along with delete operations

- Dynamic HTML table to display proxies extracted from the providers and options to filter and export results

- HTML form to add test url's and dynamic table to display the test url's
- HTML form to perform update and test operation against any proxy providers along with a check to see if update can be made after the update time interval

- Dynamic HTML table to display the summary of update operations and corresponding status

- Dynamic HTML table to display the summary of test operations and corresponding status

4.4.2 JavaScript

There are many functions implemented to handle the frontend functionalities and as well send requests to the backend. Important functions are discussed below:

- `addProxyList()`: Sends an XHR (XML HTTP Request) `POST /proxyList` to the backend to add proxy list and extract proxies.

- `addOperation()`: Sends an XHR (XML HTTP Request) `PUT /testProxy` or `PUT /updateProxyList` based on the operation type

- `addTestURL()`: Sends an XHR (XML HTTP Request) `POST /testURL`

- `loadUpdateSummary()`, `loadTestSummary()`, `loadTestURL()`, `loadProxyList()`, `loadProxies()`: Sends GET requests to load/display corresponding resources in the frontend

- `checkIfUpdateIsPossible()`: Sends an XHR (XML HTTP Request) `POST /updateProxyList` to check if update operation is allowed considering the update interval set for each proxy provider

- `tablefilter.js`: Script to filter the proxy table based on global search criteria

5 Test Architecture

As per the requirement of the task (2.2.1.4 and 2.2.1.5), all the extracted proxies from the proxy providers has to be tested for basic functionality test and against certain test URL. This is achieved using following implementation techniques:

5.1 Basic Functionality Test

For basic functionality test, a local http server is started and a http request is sent to the local server through the proxy. If the request is successfully reached the server, proxy is working and if request times out or fails for any other reason, proxy is considered as fail.

5.2 Proxy Anonymity

By the basic workflow of the proxies, the proxies would add certain HTTP headers to the response which can be used to classify the proxy anonymity level.

Level 1 (Elite Proxies) does not add any of the following HTTP headers to the request `Authorization`, `From`, `Proxy-Authorization`, `Proxy-Connection`, `Via` or `XForwarded-For`

Level 2 (Anonymous Proxies) adds at least `Via` header for the requests going through the them

Level 3 (Transparent Proxies) does not hide the proxy and adds `Via` or `X-ForwardedFor`

All the above headers would be checked and corresponding anonymity is recorded for each proxy

5.3 Test URL

Task requirement is to have at least 3 test URL's to check the unrestricted access of geolocation dependent services. Since with most of the extracted proxies from the provider s, testing of unrestricted access of service was not possible, this requirement is implemented in the following manner.

Identify the service/website which when made a HTTP GET request returns the current IP address of the system. Now when request is made to those IP discovery websites through the proxy, it is expected that proxy IP is returned. If proxy IP is successfully returned then proxy is working. This implementation is handled in the server using `proxy-test.js` script.

Following test URL's are identified which returns the IP address of the system and are used in the application.

Test URL
https://httpbin.org/ip
http://ip-api.com/json/
https://api.myip.com/

Table 2.0 Test URL

6 Project Workflow

This section covers all the user action flows from frontend to the backend and all the interfaces and functions participation in each of the scenarios.

Follow the init script enclosed within the zip file to start the server before trying any of the following workflows.

As per the task requirement to have 5 different proxy providers and total of 100 proxies from these providers, below identified proxy providers in table 1.0 are used:

Proxy Providers	Extraction Type
https://www.proxy-list.download/api/v1/get?type=http	HTML
http://pubproxy.com/api/proxy	JSON
https://free-proxy-list.net/	HTML
https://api.getproxylist.com/proxy	JSON
https://proxy11.com/api/proxy.json?key=MTMzMg.XtwxEw.KJG96gEm4jgrMPkpv8DS4pum0lA	JSON

Table 1.0: Proxy Providers

6.1 Create Proxy List

1. Open the application in the browser e.g. http://localhost:3334
2. In the left menu, click on 'Add Proxy Provider' as shown in Fig 1.0
3. Enter the proxy provider URL from the above table and its extraction type
4. Enter update time interval in mins and age in days
5. Submit
6. XHR request with JSON payload from the entered form data is sent to `POST /proxyList`
7. Server receives this request and executes `createProxyList()` and inserts the proxy provider information into `proxy_list` table and passes the `proxy_list_id` PK to the extraction functions to add individual proxies for each provider sources (more information in 4.2) and returns 201 Created for successful request and 400 Bad request for failures

8. XHR request in step 5 waits for successful response and displays the response messages in alert.

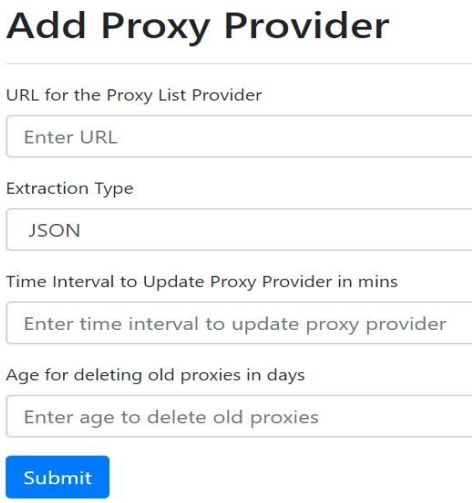

Fig 1.0 Add Proxy Provider

6.2 Proxy Extraction

As soon as server receives the request to add the proxy list, based on the extraction type extraction functions are invoked.

1. `getProxiesFromProxyProviderInJSONAndInsertToDB` or `getProxiesFromProxyProviderInHTMLAndInsertToDB` are invoked based on the extraction type.

2. These functions receive the `proxy_list_id` from `createProxyList()` and adds it as FK in proxy table for the extracted proxies.

3. Since extraction of each proxy is based on unique parsing logic, considering the scope of the task, extraction is possible and works only for the proxy providers mentioned in table 1.0

6.3 Load Proxy Providers and Proxies

As the data in `proxy_list` and `proxy` table get inserted and XHR responses with 2xx, HTML page is auto refreshed and `loadProxyList()` and `loadProxy()` JS functions are called this send `GET /proxyList` and `GET /proxy` to the backend and response data is added dynamically to the HTML table.

Proxy providers can also be deleted by clicking on 'x' icon and client sends a `DEL proxyList/:proxy_list_id` with request param as `proxy_list_id`. Server deletes the proxy provider entry in `proxy_list` table and also all associated proxies for that proxy provider in proxy table.

Proxies can be searched using the search input field and search is based on global text match for the entire table. This has been achieved using `tablefilter.js` in the client side.

Filtered proxies can be exported to HTML modal with IP and Port's by clicking on the export button and client invokes the JavaScript function `loadExportFunction()`

Proxy Providers

ID	URL	Extraction Type	Last Successful Update	Last Update Attempt	Delete
118	https://proxy11.com/api/proxy.json?key=MTMzMg.XtwxEw.KJG96gEm4jgrMPkpv8DS4pum0lA	JSON	23-6-2020 0:0	23-6-2020 0:0	x
122	https://www.proxy-list.download/api/v1/get?type=http	HTML	23-6-2020 0:0	23-6-2020 0:0	x
123	http://pubproxy.com/api/proxy	JSON	23-6-2020 0:0	23-6-2020 0:0	x
124	https://free-proxy-list.net/	HTML	23-6-2020 0:0	23-6-2020 0:0	x
125	https://api.getproxylist.com/proxy	JSON	23-6-2020 0:0	23-6-2020 0:0	x

Fig 2.0 Proxy Providers list

Proxies

Count

Search...

Export

ID	IP	Port	Last Test Successful	Last Found In List	First Found In List	Proxy Provider ID	Test Result	Anonymity
1500	1.0.0.63	80	23-6-2020 15:33	23-6-2020	23-6-2020	118	Fail	NA
1501	104.28.10.51	80	23-6-2020 15:33	23-6-2020	23-6-2020	118	Fail	NA
1502	1.0.0.206	80	23-6-2020 15:33	23-6-2020	23-6-2020	118	Fail	NA
1503	1.0.0.171	80	23-6-2020 15:33	23-6-2020	23-6-2020	118	Fail	NA
1504	1.1.1.37	80	23-6-2020 15:33	23-6-2020	23-6-2020	118	Fail	NA

Fig 3.0 Proxy List

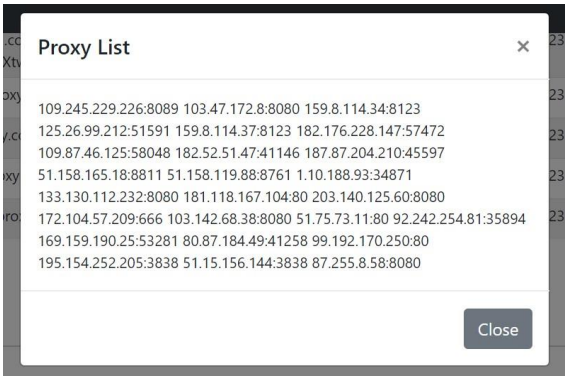

Fig 4.0 Export Proxy List

6.4 Add Test URL

Test URL to perform the functionality test of individual proxies can be added but entering the identified URL and clicking on submit and `POST /testURL` XHR request is sent to the server which then invokes `createTestURL()` and returns 2xx response. Once client receives the success status code, added test URL is loaded to the table using `loadTestURL()`

Test URL's can also be deleted by clicking on 'x' icon, `DEL /testURL /:test_url_id` XHR is sent to the server, where `test_url_id.`

Criteria to select the type of test url is describe in section 5.0 Test Architecture

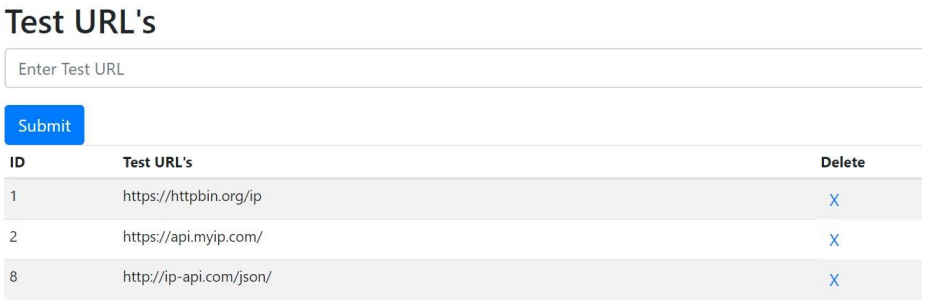

Fig 5.0 Test URL

6.5 Update and Test Proxies

6.5.1 Update

1. Select operation as Update
2. Select the proxy provider for which proxies has to be updated
3. By default, submit button is disabled for update operations and only Check for Update button is active.
4. Click on Update button and a `POST /updateProxyList` XHR request is sent to the server and it invokes `checkForUpdate()` where update interval set during creating the proxy list is compared with the data from `update_summary` table for corresponding proxy list current time stamp and then submit button is enabled accordingly.

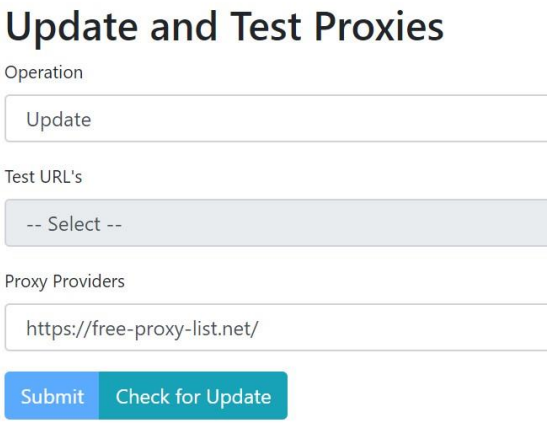

Fig 6.0 Update and Test Proxies

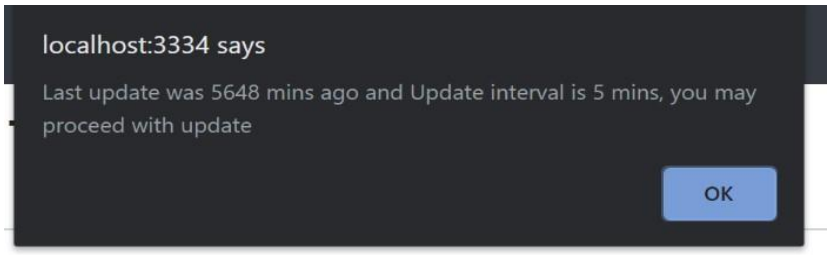

Fig 7.0 Check for Update Pass

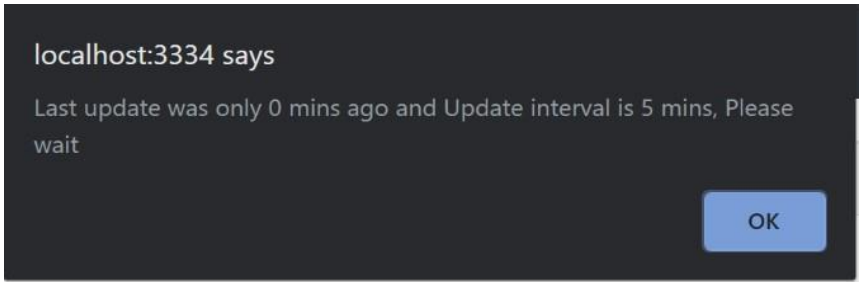

Fig 8.0 Check for Update Fail

6.5.2 Test

1. Select operation as Test
2. Select the test URL
3. Select the proxy provider from which proxy's functionality test has to be performed
4. Submit

`PUT /testProxy` XHR request is sent to the server which then invokes `testProxyAndUpdateToDB()` and tests the proxies based on the test architecture as described in 5.0

6.6 Update and Test Summary Tables

Each update and test requests are recorded with all necessary time stamp and status. As soon as update and test requests from above section are complete , update_summary and test_summary tables gets updates with the necessary information .

loadUpdateSummary and loadTestSummary GET's the latest information from the databse through corresposnign XHR requests .

Individual summary record can be deleted by clicking on ,'x' icon and `DEL` XHR request is sent to the server which then invokes `deleteTestSummary()` and `deleteUpdateSummary()`

In case if there are any error's during the operations , status is updated with error messages in the corresponding tables.

Test Summary

ID	Test URL	Proxy Provider	Date	Status	Delete
3	https://httpbin.org/ip	https://proxy11.com/api/proxy.json?key=MTMzMg.XtwxEw.KJG96gEm4jgrMPkpv8DS4pum0lA	18-6-2020 0:0	Completed	X
4	http://ip-api.com/json/	https://proxy11.com/api/proxy.json?key=MTMzMg.XtwxEw.KJG96gEm4jgrMPkpv8DS4pum0lA	19-6-2020 0:0	Completed	X
20	https://api.myip.com/	https://proxy11.com/api/proxy.json?key=MTMzMg.XtwxEw.KJG96gEm4jgrMPkpv8DS4pum0lA	23-6-2020 0:0	Completed	X

Fig 9.0 Test Summary

Update Summary

ID	Proxy Provider	Date	Status	Delete
11	https://proxy11.com/api/proxy.json?key=MTMzMg.XtwxEw.KJG96gEm4jgrMPkpv8DS4pum0lA	21-6-2020 0:16	Completed	X
12	https://free-proxy-list.net/	23-6-2020 2:54	Completed	X
13	http://pubproxy.com/api/proxy	23-6-2020 3:9	Completed	X

Fig 10.0 Update Summary

7 Requirement vs Implementation Matrix

This section details the requirements of the task against implementation logic.

Task ID	Implementation Summary
2.2.1.1	Proxies are add from the front end and based on the individual proxy parsing logic , backend aggregates the data
2.2.1.2	All the update entries are stored in update_summary table and for every update request this table is queries and a check is performed
2.2.1.3	proxy and proxy_list tables are set to acceept only UNIQUE inserts
2.2.1.4	Send http request to local server and through the proxy and verify
2.2.1.5	Refer 5.3
2.2.1.6	Cron job is scheduled to delete old records which are test fail. As per the database schema no proxy is without a proxy provider.
2.2.2.1/2/3	Database stores all the information and front end communicates through REST API's to retrieve data from database
2.2.3.1/2	REST interface is exposed through ExpressJS framework
2.2.3.3	No configurations are stored in the database
2.2.3.4	REST interface /updateProxyList is implemented to handle updates
2.2.4.1	Front end display all the data

2.2.4.2	Test URL information is displayed. Test result is displayed individually for all the proxies and test summary is displayed in Test Summary sections for all the test operations for proxy list and corresponding test url's.
2.2.4.3/4	Basic usability and design aspects are considered

2.2.4.5	Frontend display all the defined information
2.2.4.6	Three test URL's are identified as per test architecture. Refer 5.3
2.2.4.7	Five different proxy provider with more than 100 proxies are identified. Refer 6.0
2.2.5.1	Anonymity check is implemented. Refer 5.2
2.2.5.2	Refer 6.1 and Fig 1.0
2.2.5.3	Proxy List can be created , deleted and updated Refer 6.1 , 6.2 , 6.3
2.2.5.4	Filter is implemented by global string search across the table along with export of the filtered data. Refer 6.3

Bibliography

[1] https://nodejs.org/en/

[2] https://expressjs.com/

[3] https://getbootstrap.com/

[4] https://www.postgresql.org/

[5] https://www.restapitutorial.com/

[6] https://www.npmjs.com/

Appendix A

- **GET /proxyList**
    - Request Headers
        - Content-type: application/json
    - Response: JSON Array
        - proxy_list_id : Integer
        - url: String
        - extraction_type: String
        - last_success_date: Date
        - last_update_attempt: Date
        - update_time: String
        - age: String
- **GET /proxy**
    - Request Header
        - Content-type: application/json
    - Response: JSON Array
        - proxy_id: Integer
        - ip: String
        - port: String
        - last_test_success: Date
        - last_found_in_list: Date
        - first_found_in_list: Date
        - proxy_list_id: Integer
        - test_result: String
        - anonymity: String

- **GET /proxyList/: proxy_list_id**
    - Request Header
        - Content-type: application/json
    - Request Param: proxy_list_id: Integer
    - Response: JSON Array
        - proxy_list_id: Integer
        - url: String
        - extraction_type: String
        - last_success_date: Date
        - last_update_attempt: Date
        - update_time: String
        - age: String
- **GET /proxy/: proxy_id**
    - Request Header
        - Content-type: application/json
    - Request Param: proxy_id: Integer
    - Response: JSON Array
        - proxy_id: Integer
        - ip: String
        - port: String
        - last_test_success: Date
        - last_found_in_list: Date
        - first_found_in_list: Date
        - proxy_list_id: Integer
        - test_result: String
        - anonymity: String

- **POST /proxy**
    - Request Header
        - Content-type: application/json
    - Request Body : JSON
        - Ip : String
        - Port: String
        - proxy_url_id : String
    - Response: Success Message in String

- **POST /proxyList**
    - Request Header
        - Content-type: application/json
    - Request Body: JSON
        - url : String
        - extraction_type: String
        - update_time: Timestamp
        - age: String
    - Response: Success Message in String

- **DELETE /proxyList/: proxy_list_id**
    - Request Param:
        - proxy_list_id : Integer
    - Response: Success Message in String

- **PUT /testURL**
    - Request Body: JSON
        - test_url : String
        - proxy_list: String
        - proxy_list_id: Integer
    - Response: Success Message in String

- **POST /testURL**
    - Request Body: JSON
        - test_url : String
    - Request: Success Message in String

- **GET /testURL/**
    - Request Header
        - Content-type: application/json
    - Response: JSON Array
        - test_url_id: Integer
        - test_url: String

- **DELETE /testURL /: test_url_id**
    - Request Param:
        - test_url_id : Integer
    - Response: Success Message in String

- **GET /testSummary/**
    - Request Header
        - Content-type: application/json
    - Response: JSON Array
        - test_summary_id: Integer
        - test_url : String
        - proxy_list : String
        - start_time: Timestamp
        - test_status: String

- **GET /updateSummary/**
    - Request Header
        - Content-type: application/json
    - Response: JSON Array
        - update_summary_id: Integer
        - proxy_list : String

        - start_time: Timestamp
        - test_status: String

- **DELETE /testSummary /: test_summary_id**
    - Request Param:
        - update_summary_id: Integer
    - Response: Success Message in String
- **DELETE /testSummary /: test_summary_id**
    - Request Param:
        - test_summary_id : Integer
    - Response: Success Message in String

- **POST /updateProxyList [Check if Update is Possible]**
    - Request Body:
        - proxy_list: String
    - Response: Check Result in String

- **PUT /updateProxyList [Perform Update]**
    - Request Body:
        - proxy_list: String
        - proxy_list_id: Integer
    - Response: Update Result in String