

**פרויקט מעשי לתואר B.Sc. במדעים**

**עדכוני סייבר שליליים**

**Negative Cyber News**



**מגישים:** ניב עמוס – 316464486

רועי עבודי – 315976415

מתן צופי - 312539539

**בהנחיית:** רועי זימון **חתימה: <**חתימות המנחים>

**תאריך הגשה:** 27/6/2021

**Table of Contents**

**Project summery………..……………………………………………….3**

**Introduction….………………………………………………………….4**

**Background…………………………………………………………………..4**

**Targets………………………………………………………………………..4**

**Project structure,…………………………………………………………….5**

**Literature study….…………………………………………………………..7**

**Project implementation…………………………………………………8**

**Implementation………………………………………………………………8**

**Issues………………………………………………………………………….9**

**Algorithms…………………………………………………………………..11**

**Project products………………………………………………………..15**

**Project conclusions…………………………………………………….17**

**OSINT Project…………………………………………………………18**

**OSINT project preview…………………………………………………...18**

**The base tool…………………………………………………………18**

**Base tool use case diagram…………………………………………………19**

**Base tool showcase……………………………………………………….…20**

**Project Summary**

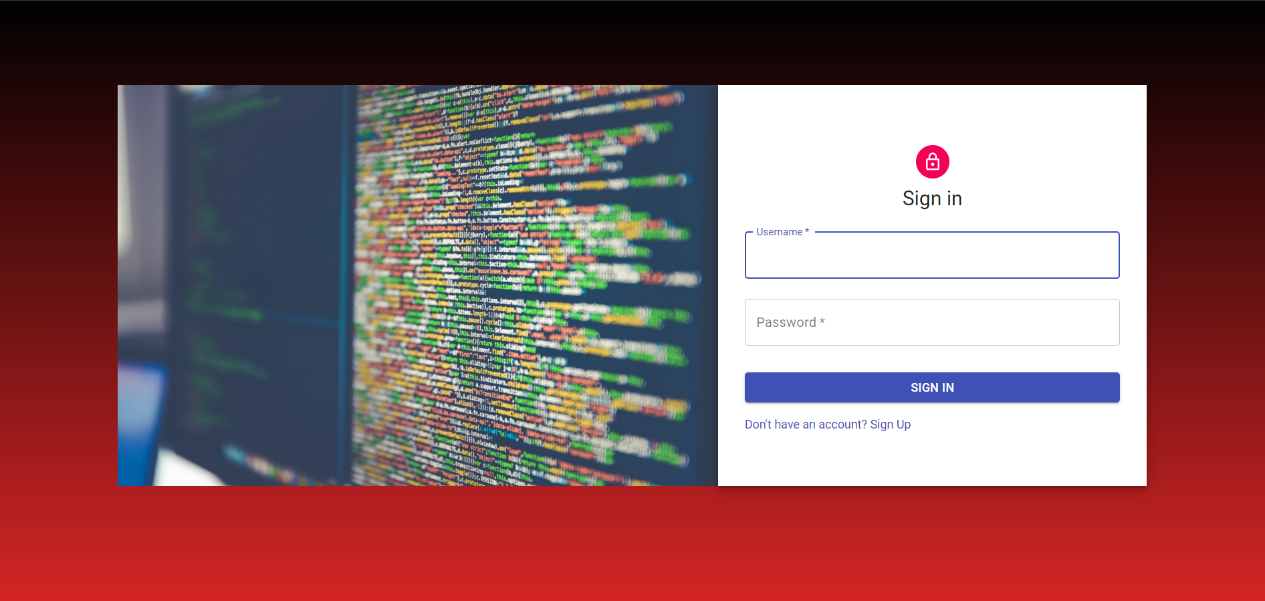
Our project goal is to show recent reports/articles, about cyber-attacks on companies. We have built a web application that includes a search engine, in the search engine the client will insert a company name, and then our application will search for cyber-attacks reports about this company and show them to the client. The web application requires registration and login in order to use.

The cyber-attacks reports are collected from credible resources involved in the cyber industry, in the role of open-source intelligence. Each resource has been tested and chosen because it has the most recent and reliable reports.

The products that were achieved are:

* Rest API: The server side of the application, this part is in charge of the collection of cyber-attacks reports, registration, and login. Each potential client can use this API in order to get the data.
* Web application: The client-side of the application, this part is in charge of the user interface in a friendly and simple way, it sends the desired company search term to the server, which will search relevant cyber-attack reports, and then display the results to the client.
* Database: The database store the user’s login credentials.

The main flow of the web application is a registration and then login. Once authenticated, the user is authorized to search for the company name, and browse the cyber-attacks articles.

****

**Figure 1 . Login Page.** Login page to the website.

**Introduction**

**Background**

In these days organizations are more vulnerable to outside and inside cyber-attacks, leaks and exploits. Most organizations are not ready to handle a cyber-attack. For example, a bug in the system can leave access to valuable information open for the public, or hackers hack into the system and acquire personal information of clients and employees. Leaks can include personal data of existing and past clients, which makes the organization uncredible for new clients. Some of the threats are not always well known and exposed to the public, and some organizations even try to cover-up their vulnerabilities, in the purpose of hiding it from their clients or potential clients and business partners. In some cases, the organization may not prioritize or capable of fixing the vulnerability, leaving it open for repeating attacks.

Additionally, in recent years, cyber-attacks became more frequent, and encompass broader range of targets that are not from the hi-tech industry. For example, banks are one of the most common cyber-attack targets. As more products start connecting to the cyber space, they became more vulnerable to cyber-attacks. Products such as smart houses, cars and TV's can become targets for cyber-attacks as they become more common.

Therefore, since people start connecting to the cyber space through means other than their computers and cellphones, they are exposed to direct attacks from other products. Furthermore, as organizations get bigger, having more and more clients, they start providing services outside of their scope, and therefore they become more vulnerable to cyber-attacks and exploits. Big companies, like Microsoft or Google, have array of different services and products, directed to different clients. Each one of these products can be a target to a cyber-attack.

**Targets**

Since cyber-attacks are a lot more common than what people think and can occur in wide variety of products, it's easy to lose track of all of them. People sometimes don't fully know all of the cyber-attacks that occur against their products. For instance, data breach against an organization can leave behind personal data of hundreds or even thousands of clients. These can include email addresses and passwords, which their owners must be aware of. Our project is designed to help people in this kind of situations.

While they are not our primary audience, business partners and shareholders can also use our tool to know better about the organization they are associated with. It can help them make better business decisions based on the organization security and credibility.

Our project condenses the latest news about cyber-attacks on organizations, using a simple interface that allows to search for an organization name. We also incorporate configurable keywords that help narrowing down results and display only the relevant sections. Only registered users can access the tool. Our goal is to help clients stay informed about all of the ongoing cyber-attacks that occur against the organizations they mainly consume from. With our tool, clients can look up any organization name and read about the latest data breaches and exploits that occur in the products they use. For example, an Adobe Photoshop user can look up on Adobe and see if they had any data breach or a company that uses Microsoft Azure service can see if the service had any exploits.

**Project Structure**

Our project is a web application written in JavaScript. We use React for our front-end with components from Material-UI, to achieve a modern feel and look to the application. We also use MongoDB database to manage the users. The main page asks the user to log in or register and then loads the search page. The application is built as Single-page application, meaning that the search results will load up on the current page instead of on an entire new page.

For our search algorithm, we use scripts written in Python to scrap the latest news.

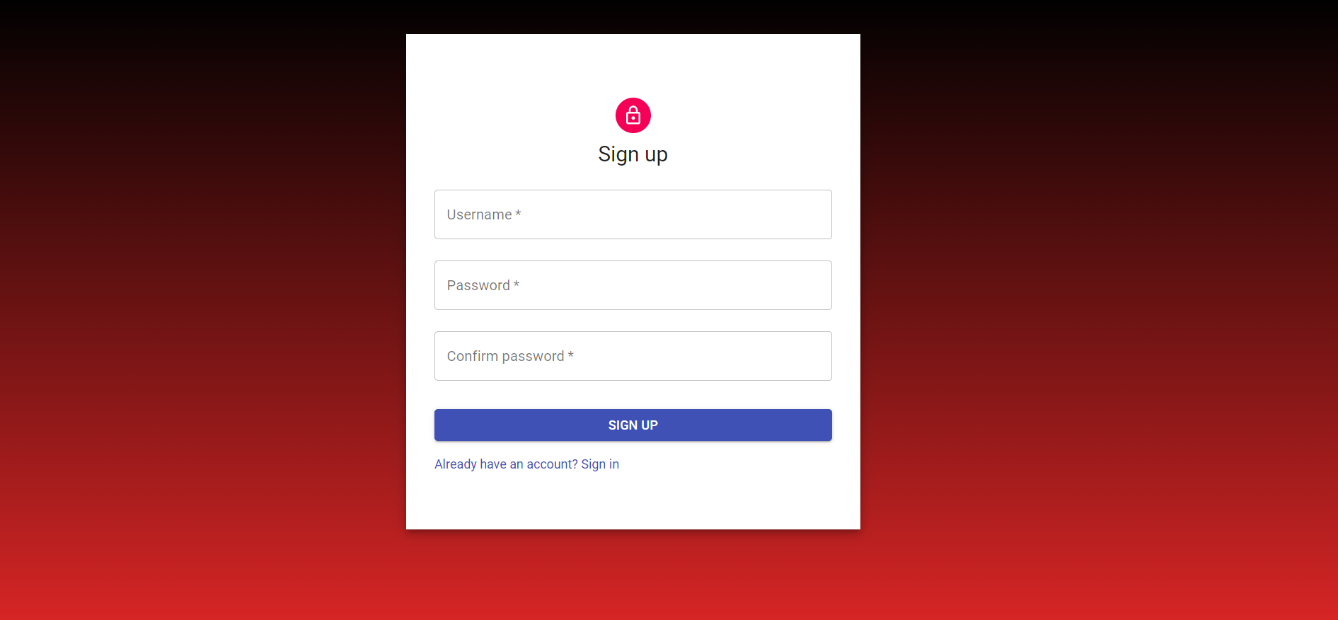
|  |  |
| --- | --- |
| Usage | Library/Package/Language |
| Framework used to write, review, edit and debug our code. | Visual Studio Code |
| The entire web application is written in JavaScript. | JavaScript |
| Handles the front-end; user interface. | React |
| Database for managing registered users. | MongoDB |
| Handles HTTP requests. | Axios |
| Used to scrap relevant articles from the web. | Python (BeautifulSoup) |
| Handles the back-end; server-side. | Node.js + Express |

**Literature study**

Since our project only display articles from the internet, we had to search for websites that have cyber related articles and a search functionality. When searching from our application, we'll show different articles from selected websites. The first sites we used were "upguard.com", "cyware.com" and "cybernews.com". The former two had been proven difficult to use, meaning we ran into technical difficulties while trying to display articles so we had to replace them with "threatpost.com" and "welivesecurity.com".

When using our web application to search for a company, articles from "cybernews.com", "threatpost.com" and "welivesecurity.com" show up in the results, with links directing to the articles themselves.

Other methods we considered were using Google API for searching through the web, or using Selenium library in Python, but eventually we decided to use BeautifulSoup library with HTTP GET request.



**Figure 2. Sign Up Page.** Sign up to the website. requires username and minimum of 6 chars password.

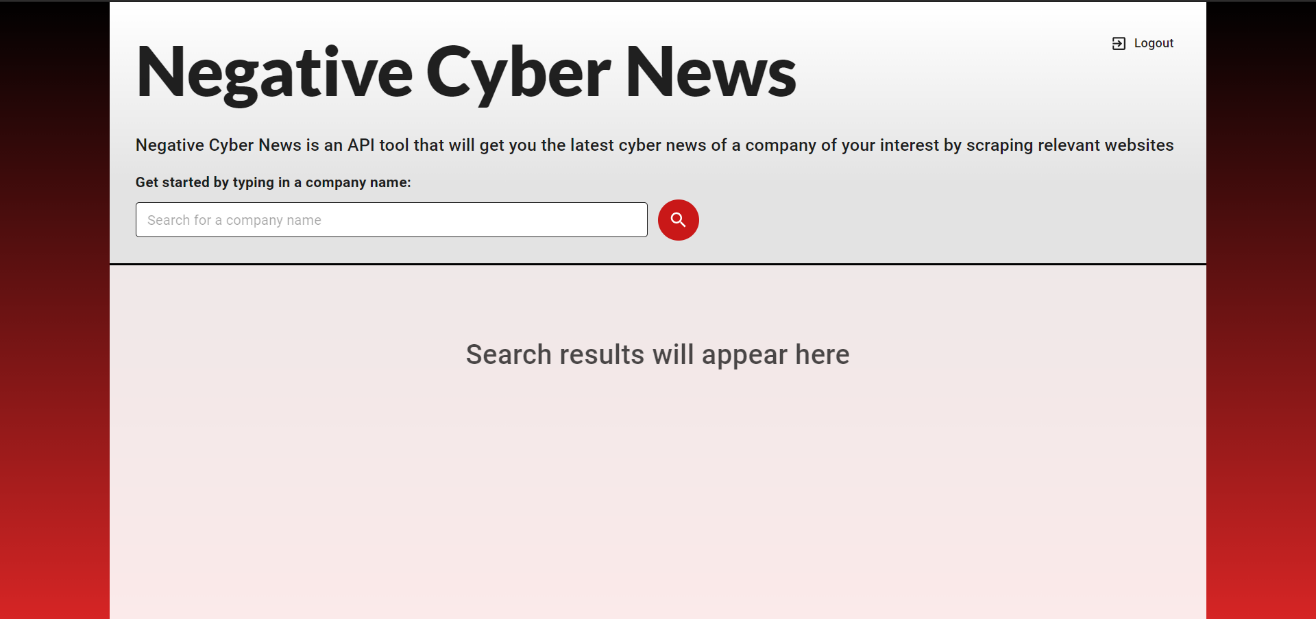
**Project implementation**

**Implementation**

At first, we used Selenium library with BeautifulSoup to gather the relevant articles about the company cyber-attacks. Selenium was used to open browser process and to execute the search query automatically, while BeautifulSoup was used to scrape the webpage for the article. Each website was scraped in parallel to save processing time.

We decided to use articles from specific websites that we selected. After intense search we came across several websites that fit our requirements.

* Google News: Has search functionality, easy to scrape and have many sources to different articles, but the results are too generic and not focused on cyber-attacks.
* Cyware.com: Has search functionality and shows relevant and recent cyber articles about the desired company. The website is difficult to scrape because the server loads nonrelevant articles from time to time.
* Cybernews.com: Easy to scrape and show good results. Most of the articles are written by them and focused only on cyber-attacks.
* Upguard.com: Harder to scrape because the search process was slow and required to load several pages until the results displayed. Sometimes the articles were not up to date. The website has risk assessment tool about some companies we can display to the user.
* Threatpost.com: Takes a little bit of time to scrape but have excellent results that are written by the website staff.
* Welivesecurity.com: Shows good results with relevant articles and can be scraped easily and quickly.



**Figure 3. Landing Page.** The main page of the website. Here the user can search for a company name.

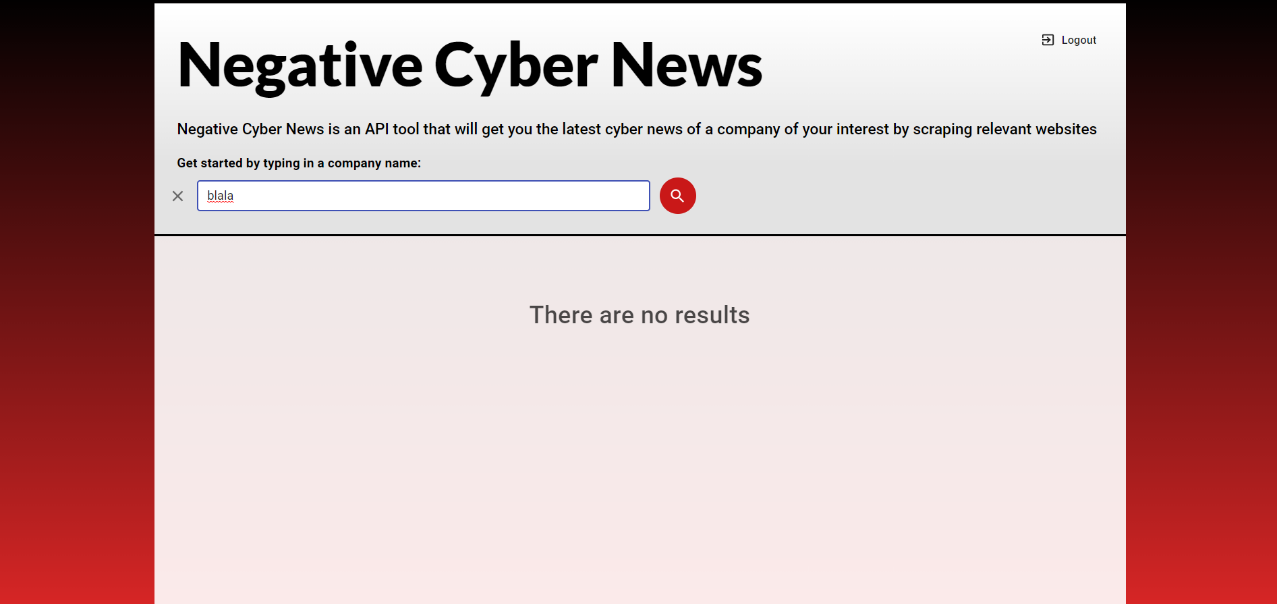
**Issues**

1. **Scraping websites**

As mentioned earlier, some websites were problematic to scrape or showed nonrelevant results. For example, "Cyware.com" sometimes loaded up the most recent cyber news in general before displaying search results on the desired query. This had been proven difficult to scrape because it required to timeout the scraper until the relevant articles loaded up. In order to solve this issue, we decided to remove websites with this kind of problems and keep only the reliable websites for scraping.

1. **Nonpersistent session**

The session between the user and the server remains open for several hours and only close when the user wants to log out or after the session time limit. We faced an issue that if a logged in user refreshed the webpage, it would log him out and close the session with the server. This issue was caused by the way React JS render pages. After refreshing React would delete the current state and render the webpage without the logged in user. We discovered that React called the render method before getting the user data, so the webpage loaded without the user. We managed to solve this by inserting the get user data method inside the render method.



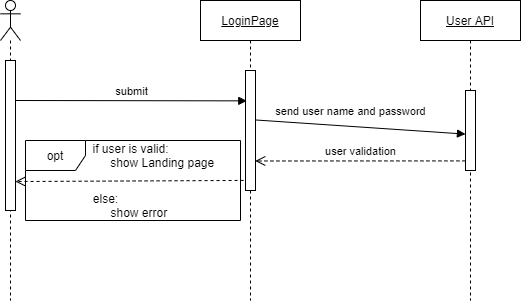
**Figure 4. No Results.** Shows that sometimes there are no results to show.

1. **Selenium**  
   Selenium is a Python package that used to perform automated tasks on webpages. We used Selenium to scrape webpages for articles. This proved to be inefficient and too slow, as Selenium is using browsers process to get the needed data. A single search could take up to a minute to load. This was unacceptable and we had to replace Selenium. We eventually settled on BeautifulSoup package which is faster than Selenium. Unlike Selenium, BeautifulSoup is using HTML GET requests to acquire the needed data so it uses much less resources from the server. Now every search can take up to fifteen seconds to load, that’s over 60% improvement.

**Algorithms**

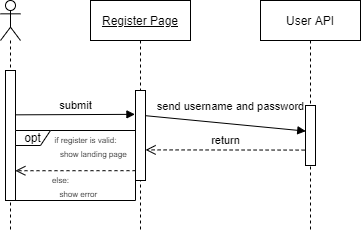
1. **Login**

At first, the login page is displayed to the user. The user is required to enter his username and password in order to log in to the website. After the user enters his username and password**,** he clicked on the log in button. The landing page component send a POST request with the user credentials to the user API**.** User API is the route that handle user authentication on the server side. When the POST request arrives to the user API, it sends the user credentials to "passport.js" package, which verify the user credentials with the DB.

2. **Register**

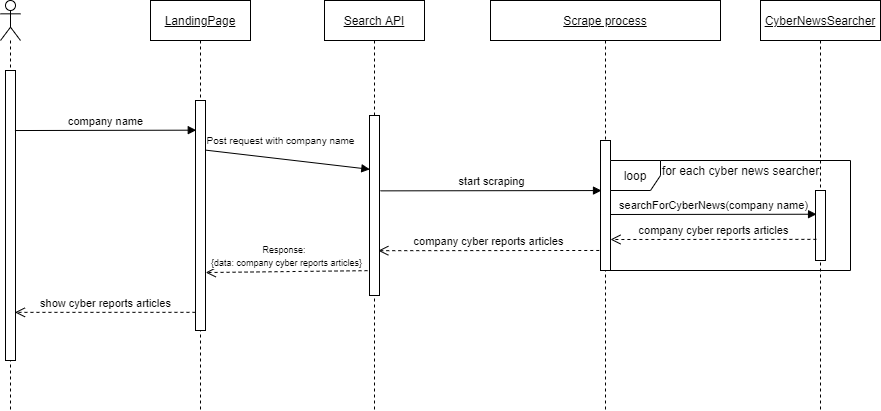
**Figure 5. Login Process Sequence Diagram.** Shows the login and validation process.

From the login page there is an option to register, after the user choose this option, he arrives to the register page. In the register page, the user is required to enter his desired username and password. When the user clicks the register button, a POST request is being sent from the register page component to the user API route. The API creates a new user schema, and then sends the new user schema to "passport.js" package. It checks if the user is already exists in the DB,if not it converts the user schema to an unique hash code and store it in the DB. If the register is valid, then the website will be redirected to the landing page.



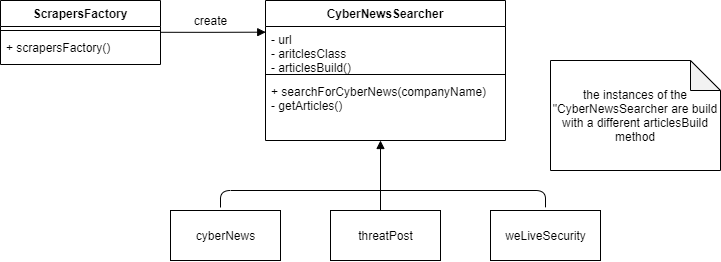
**Figure 6. Register Process Sequence Diagram.** Shows the registration process.

3. **Search**

Once the user logs in, he is redirected to the landing page. There the user can make a search request (HTTP GET request) with the desired company name. The request is being sent to the search API route with the user's input. The route activates a python script with the search term as an input. The python script is responsible for scraping the web and its output is an array containing several articles from different cyber sites. The python process creates three classes, each class represents a cyber news website. Each class makes a search query in each website with the user input term, and keeps the source page of the search results. "BeautifulSoup" library then scraps the source code and extracts the latest articles from each website. Each class returns the articles to the process script, which creates a JSON format containing all of the returned articles. The search API returns a response to the landing page with the JSON articles and then shows them to the user.

**Figure 7. Search Process.** Shows the search process.

4. **Scraping classes**

****Each one of the scraping classes is created with the factory design pattern.The creating function gets an ENUM that indicates which class to create. Then, it creates the appropriate class with the website URL, scraper parameters and method. Every class have a method that responsible to collect the articles, on each class this method uses a different strategy (strategy design pattern) to collect the articles, depending on the website.

**Figure 8. Scrapers Class Diagram.** Shows the relations between the scrapers' classes.

**Project** **products**

* **Server**

The server handles the user authentication, the python scripts and the routing. It's a standalone API that responsible for sending data between the client and the database when logging in, registering or when there is an error in the authentication process. It is also responsible for sending user input, activating the scraping process and returning the results to the client.

* **Database**

The database stores the registered users' credentials as a hash code. When registering, a new entry is added with the user's credentials as a hash code. When logging in, the user's credentials is compared with the stored hash codes, when there is a match, the user is allowed to log in, if there's not, the user receives an error accordingly.



**Figure 9. Search Results.** Results to the search term "apple".

* **Scripts**

Scripts are used for scraping webpages and written in Python. All the websites are scraped in parallel with different threads, and the data is saved until the entire scraping process is finished. Then, the scraped data is combined into one object with a JSON format which returns to the server.

* **Client**

The client handles all the user interface and displaying articles to the user. It is responsible for organizing the page layout and loading the proper design for each webpage. It is also responsible for sending the user input to the server and displaying any error it receives.

**Project conclusions**

**Conclusions**

The end product of the project mostly did meet our expectations; therefore, we consider the project as a success.

* Fully implemented user registration and authentication features, with database that stores the users' credentials.
* Fast search engine that shows the latest and most relevant results.
* Modern design to the website.
* Fully implemented error handling and error presentation to the user.
* Complete separation between client and server.

**Ideas for improvements**

* An additional possible improvement for the application is to create a new route, that displays the latest CVEs. It will do so by occasionally (possibly once a day) scraping a website or twitter page that displays the latest CVEs.
* Adding features that include machine learning to improve the search algorithm.
* Adding score to each result that indicates its credibility.

Due to time constraints, we couldn't implement most of our ideas.

**Table of Pictures**

**Login Page……………………………………………………………….3**

**Sign Up Page…………………………………………………………….7**

**Landing Page……………………………………………………………8**

**No Results……………………………………………………………….9**

**Login Process Sequence Diagram…………………………………….11**

**Register Process Sequence Diagram………………………………….12**

**Search Process…………………………………………………………13**

**Scrapers Class Diagram……………………………………………….14**

**Search Results………………………………………………………….15**

**OSINT Project**

**OSINT project preview**

OSINT (open-source intelligence) is the collection and analysis of information that is gathered from public, or open, sources. Our OSINT tool helps the user to identify breaches and leaks without compromising and revealing sensitive information. The tool uses Peer API which provides historical data of downloading and sharing content using the BitTorrent protocol. The data is then stored in an external database.

The tool called "ips to torrents", is built with Node.js as the server-side, and React.js for the client-side.

The purpose of the tool is to extract data from external API with IP addresses and will return an indication about malicious torrents, with an algorithm to reduce false positives.

**The base tool**

First, before the creation of the main tool, we created an environment (base tool) that the main tool will use for its implementation. The base tool is a web application that uses Peer API to store and provide the information from the API. The information is stored in an external database that was built in MongoDB. The web application is divided to 2 parts, client-side and server-side. The client-side includes 3 main pages, for every aspect of the Peer API:

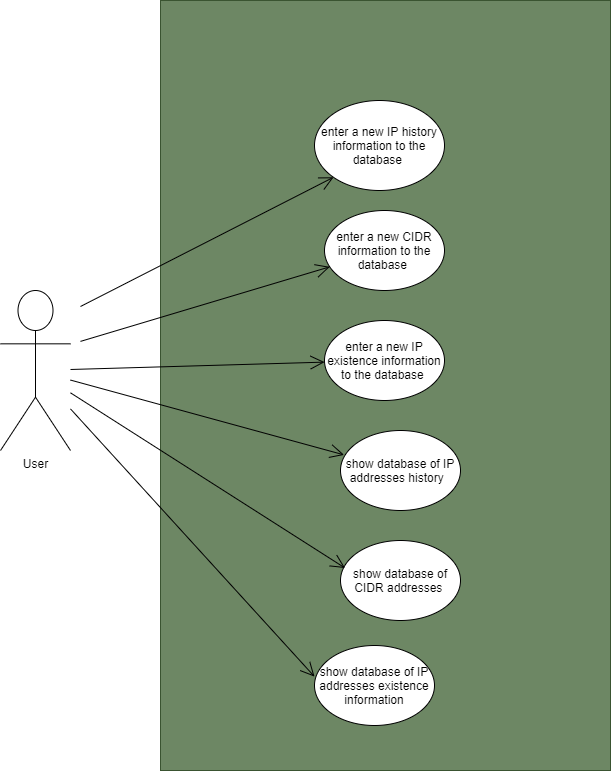
• IP history: Display information and downloaded torrents from IP entered by the user.

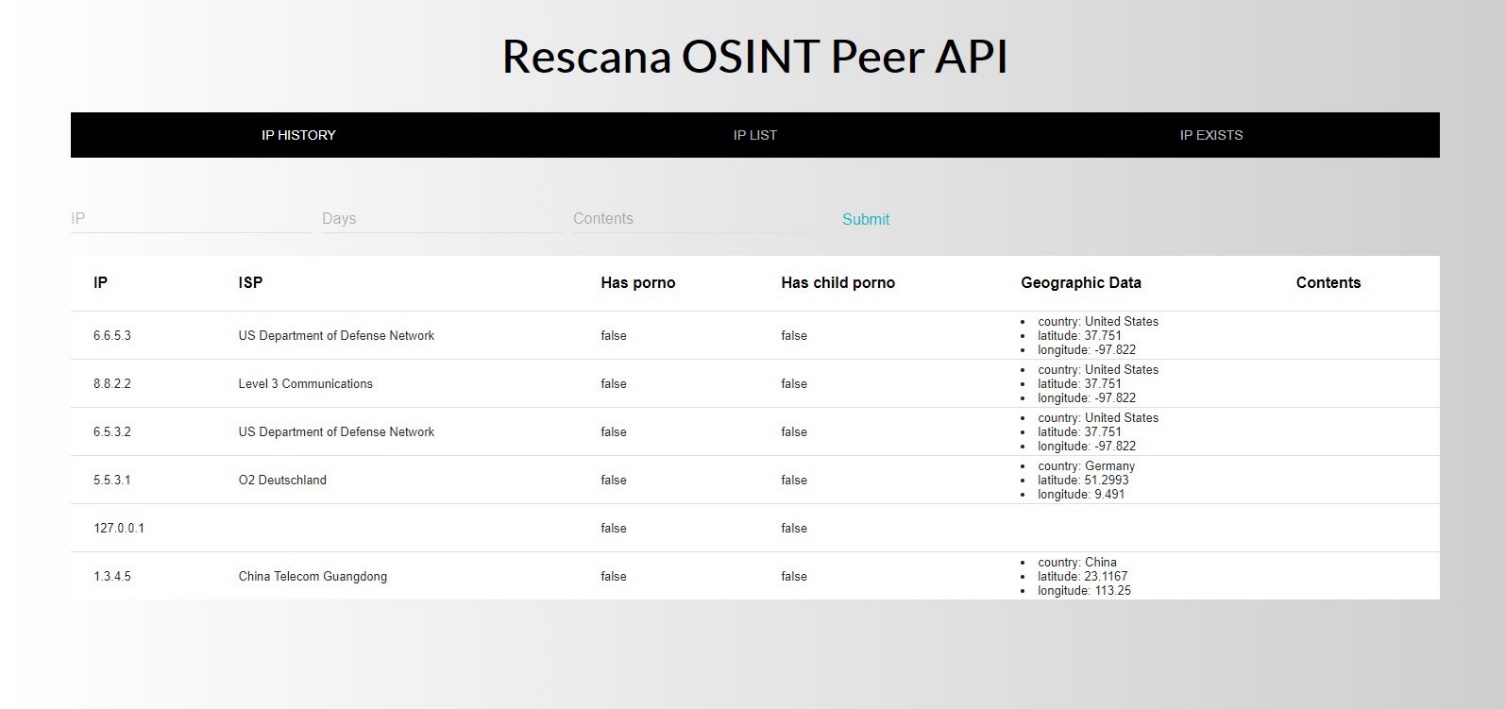
• IP list: Display a list of IP addresses from a CIDR entered by the user, that currently tracked by Peer API.‏

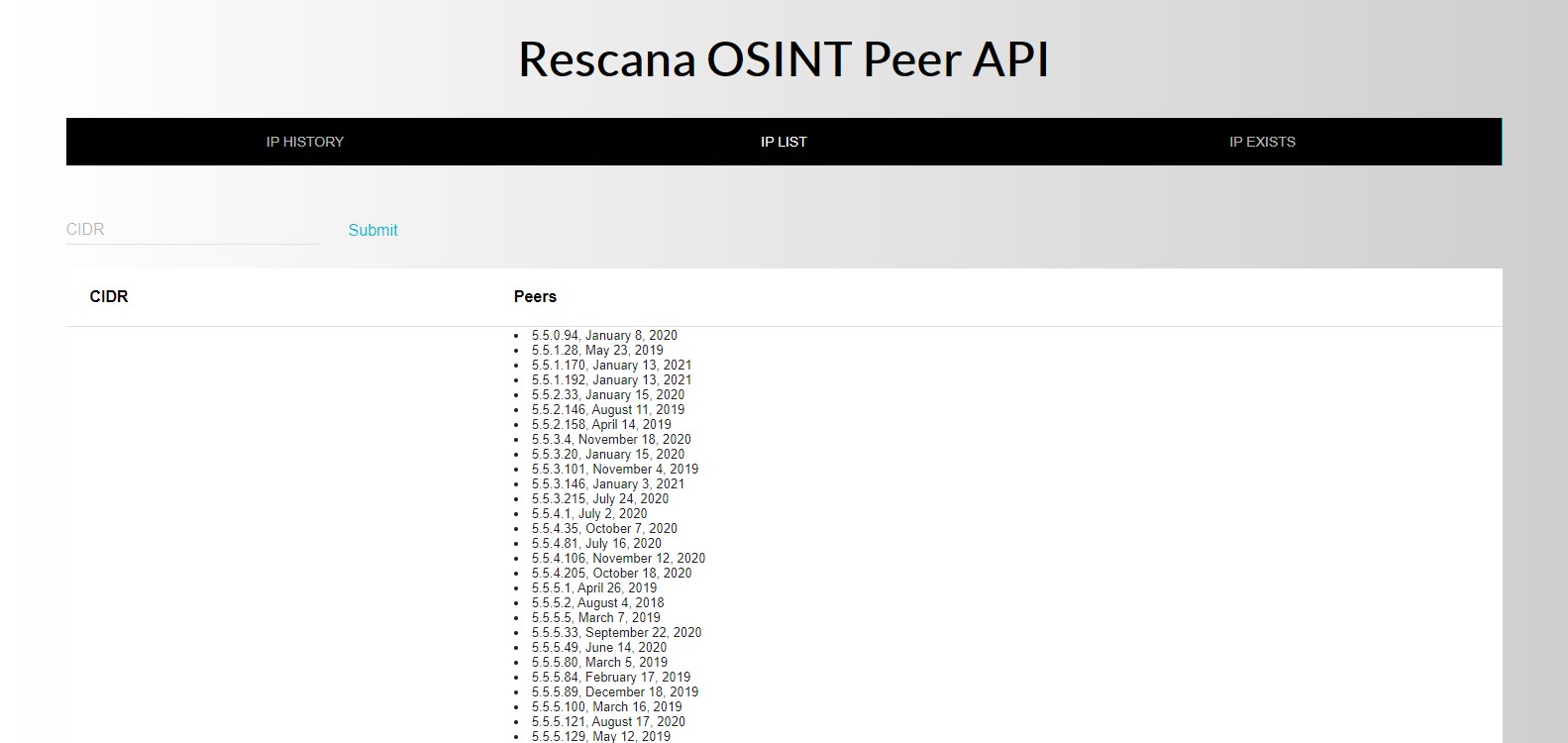
• IP exists: Display if the entered IP from the user is exists in the Peer API.

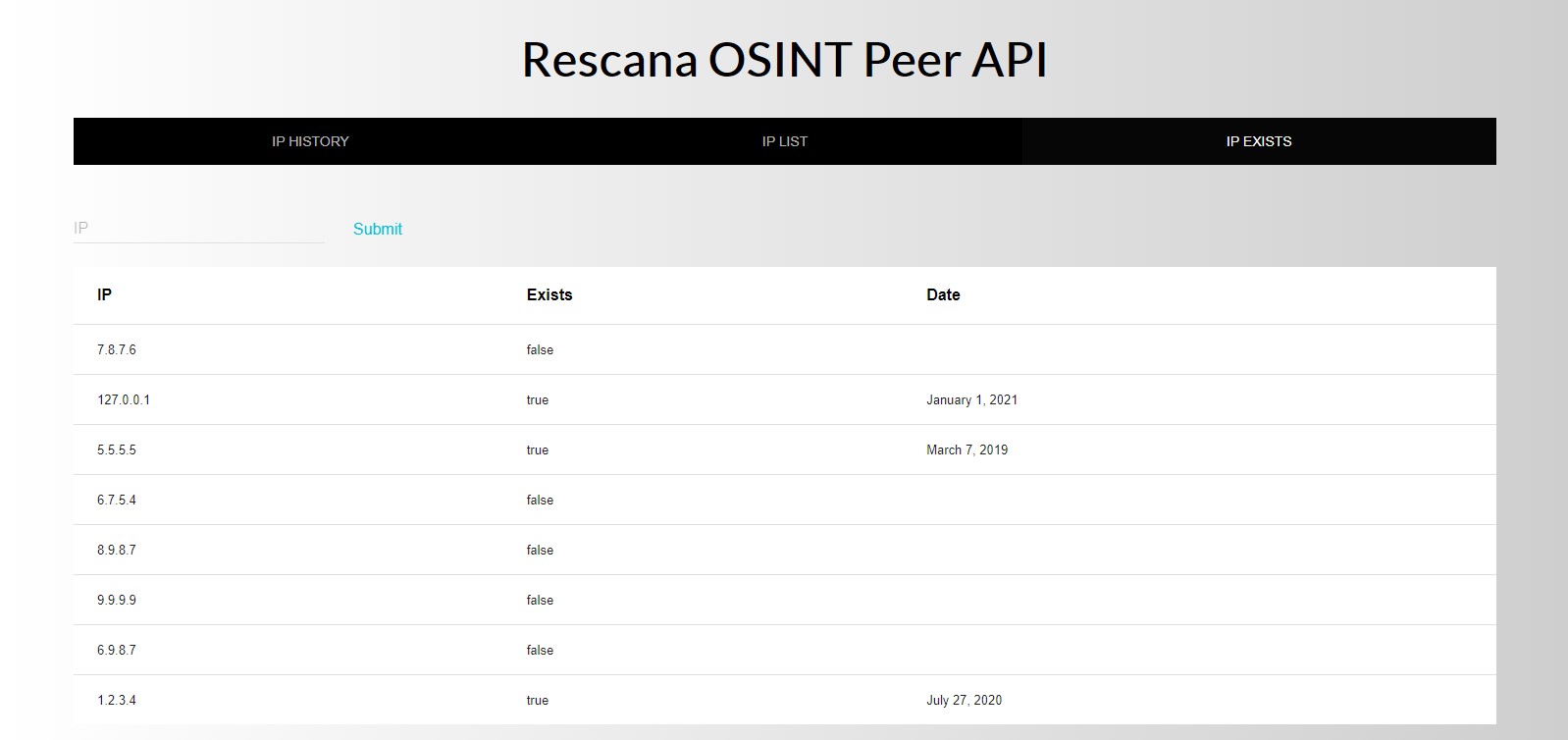
The server-side includes: connection to the Peer API, handling the database and to provide routing to get the required data.

**Base tool use case diagram**



**Base tool showcase**

****

****