An Application Programming Interface (API) defines the way in which computer systems interact. It provides an interface that simplifies the complexity of systems in a way that allows developers to build applications that can make use of external services, which can communicate with each other without effort from the end user. An Application Programming Interface (API) consists of different rules that formats and process data to facilitate interactions between computer programs thereby allowing different software applications or systems to share or exchange information with each other.

APIs enable businesses and software developers to create additional products and features faster rather than starting from afresh. These features and products will enhance a specialized platform using data or interactions. The design of APIs was out of the necessity to share information with data providers who solve specific problems so that individuals in other organizations do not spend time finding solution to those same problems. A specialized professional for example, who designs and creates maps, can easily attach a newly designed map on Google page rather than creating another Google map.

**Application Programming Interfaces (APIs) enable inter process communication**

APIs play an important role in inter-process communication (IPC) for both local communication within a system (local IPC) or across distributed systems over a network (Remote IPC).

In a local inter-process communication, a program in a machine using shared memory calls the API provided by a different program in the same machine either in a single process or between multiple processes running at the same time. Whereas in a remote inter-process communication, the exchange of data and interaction is between different servers occurring over a network like REST and SOAP (Web APIs) through hypertext transfer protocol (HTTP) or HTTPS requests and responses. These request–response APIs define a set of endpoints through which a client or customer can make HTTP requests and receives responses as JavaScript object notation (JSON) or extensible markup language (XML) from the server.

**Representational State Transfer (REST) APIs**

REST API is a structural design that facilitates communication between several web applications. REST APIs make the data available in the web server as resources and exposes the resource through a URL (Uniform Resource Locator). A client uses the URL to access the resources. The client receives the information in either XML format or JavaScript object notation (JSON). Since both humans and machine can read JSON format, it is the most preferable.

REST APIs communicate through HTTP requests to perform standard database functions like creating, reading, updating and deleting records within a resource. HTTP methods such as Create, Read, Update, and Delete (CRUD) inform the server about the action the REST APIs will perform on the resource. “Create’ generates a new resource, “Update” replaces a resource, “READ” gets the resource and “DELETE” removes the resource. The server returns HTTP response indicating a success or a failure. Codes in the 2XX range indicate success, 3XX codes indicate a resource has moved, 4XX range indicate a client-side error and 5XX range indicate server-side errors (Massé, 2012).

REST is more suitable for public APIs, cloud services, web and mobile applications. Several providers such as GitHub, Google and Twitter use REST APIs where performance, flexibility, simplicity, change in size and volume are crucial. The Twitter API is a powerful tool for accessing and interacting with a wide range of Twitter's data and functionality. The Twitter APIs is a set of RESTful endpoints that expose Twitter resources (such as tweets, media, trends, search, likes, and timeline.) to developers. The Twitter API allows third-party applications to read from and write to Twitter, enabling automation, data collection, analysis, and integration with other services.

**Simple Object Access protocol (SOAP)**

Simple Object Access Protocol (SOAP) is the standard messaging protocol for exchanging information between systems and applications using extensible markup language (XML) messages over hypertext transfer protocol (HTTP). SOAP service is independent to both language and platform; this makes the interaction of applications that are running on different operating systems and in different programming languages possible. It is a secure, reliable and trusted way to send and receive messages between systems. It is more appropriate for enterprise level applications in the financial services, insurance, healthcare and telecommunication companies.

SOAP messaging observes certain rules and has a well standard structure. The envelope, the header and the body make up the structure of a SOAP message. The envelope identifies and verifies the format of the provided document in the SOAP message, the header may be optional, but it contains the authentication data while the body, which is the essential part, contains the request and response information and the actual data in XML format.

SOAP uses two different binding style-messaging requests to exchange messages over HTTP. They are Remote Procedural Call (RPC) and Document Style. A remote procedure call (RPC)-style Web service appears as a remote object to a client application. The interaction between a client and an RPC-style Web service centers around a service-specific interface. Clients express their request as a method call with a set of arguments, which returns a response containing a return value. In the document-style of messaging, the SOAP body contains an XML document fragment. The body element reflects no explicit XML structure. The SOAP run-time environment accepts the SOAP body element as it stands and hands it over to the application it is destined for unchanged. There may or may not be a response associated with this message (Papazoglou, 2008).

**Comparative analysis of SOAP and REST**

Simple Object Access Protocol (SOAP) API is a messaging protocol. It uses the service interface to expose its functionality to clients, it is complex to develop, and it is highly specific.

Whereas representational State Transfer (REST) is, an architecture style for designing communication interfaces that allows client to access the resources using URL. It is simpler to develop, and it is more flexible.

SOAP API finds its use in legacy applications and private APIs whereas REST API in modern applications and APIs publicly used. SOAP APIs is independent, supports only XML format and data exchange can work with SMPT AND HTTP transport protocol but REST APIs works only with HTTPS and supports XML, JSON, plain text and HTML formats. REST API has a built-in error handling mechanism whereas SOAP API does not have.

SOAP APIs requires more bandwidth and resource than REST APIs. With SOAP APIs, the payload must comply with SOAP schema while REST APIs has no payload restrictions. SOAP APIs have larger messages, which makes communication slower. REST APIs have a faster performance because of its smaller messages.

SOAP APIs support encryption with additional overheads whereas REST APIs support encryption without affecting performance. SOAP APIs support SSL in addition to XML signatures and encryption in terms of security while REST APIs support SSL. SOAP APIs ensure ACID transactions, but REST APIs are limited to single HTTP transactions. It expects client to retry during a connection failure. SOAP APIs are used where high security and reliability is critical whereas REST APIs are used where high security and reliability is not critical.

**Beautiful Soup**

To get data from websites we rely on traditional web scrapping techniques like requests or Beautiful Soup in Python. Beautiful soup is a python library used for extracting data such as images, links and text from websites using simple method to navigate and manipulate the HTML or XML contents of the web pages.

To scrape the Contents from a website using Beautiful Soup involves the installation of the “requests” and “beautiful soup” libraries using pip. These libraries are essential when using beautiful soup to collect data from a specific website. The request library gets the raw HTML (hypertext markup language) content of the webpage while the beautiful soup parses the HTML content, finds and extract the required data. The creation of Tag objects occurs when parsing the website. These tag objects allow for searching and navigation within the HTML/XML document. Tag objects get the name, ids, classes and other attributes associated with each HTML tag. With the help of searching methods such as “find (), find\_all (), and find\_next ()”, provided by Beautiful Soup, the extraction of data out of the HTML/XML document becomes possible. These techniques work on static websites where the contents do not change after loading the page. They will not work for content of a dynamic website whose pages are loaded using JavaScript and provides different content after the initial page loads. It will return a 403 forbidden HTTP Status Code as a ban page. A 403 Forbidden error means that the server has refused to allow access to a particular resource, often due to permission restrictions.

To scrape the contents from a dynamic website, requires the use of other methods like using a headless Browser (e.g. Selenium), playwright, scrappy with (Splash) and browser developer tools to Inspect Network Traffic. The University of Michigan website is a dynamic website that returned a 403 forbidden error while scraping with Beautiful Soup. I used selenium to scrape the contents from the website. To scrape this website, using a chrome browser, I installed selenium and web driver-manager in python. From the URL (https://www.si.umich.edu/people/) I made use of html parser to passes the HTML of the page into the Beautiful Soup class then also used “prettify” to improve the view of the output. Using “find () and find\_all ()”, I was able to scrape the title, paragraphs(p), headers(h), “span” and “div” tags. I also used “find ()”, to scrape elements by class, a list and id.

**References:**

Geewax, J. (2021) API Design Patterns. New York: Manning Publications Co.

Jin, B., Sahni, S., and Shevat, A. (2018) Designing Web APIs. California: O’Reilly Media, Inc.

Nair, V. (2014) Getting Started with Beautiful Soup. Birmingham: Packt Publishing Ltd.

Massé, M. (2012) REST API Design Rulebook. United States of America: O’Reilly Media, Inc.

Mitchell, R. (2018) Web Scraping with Python. 2nd ed. United States of America. O’Reilly Media, Inc.

DiMarco, J. (2006) Web portfolio design and applications. United States of America: Idea Group Publishing.

Tidwell, D., Snell, J., and Kulchenko, P. (2001) Programming Web Services with SOAP. California: O’Reilly Media, Inc.

Soni, A., Ranga, V. (2019) ‘API Features Individualizing of Web Services: REST and SOAP’, International Journal of Innovative Technology and Exploring Engineering, 8(9S), pp. 664 – 671.

Jovita, J et al (2022) ‘REST API and SOAP - Web Services Validation in IoT environment’, NEUROQUANTOLOGY, 20 (16), pp. 695-710.

Papazoglou, M. (2008) Simple Object Access Protocol. Available at: [Chapter 4 SOAP: Simple Object Access Protocol - M.MOAM.INFO](https://m.moam.info/chapter-4-soap-simple-object-access-protocol_59877a451723ddd169540ec8.html) (Accessed: 10 July 2024).

Goodwin (2024) what is an API. Available at: What Is an API (Application Programming Interface)? | IBM (Application Programming Interface)? | IBM (Accessed: 12 October 2024).

Engineering (2019) what is SOAP API: Formats, Protocols, and Architecture. Available at: [What is SOAP API: Formats, Protocols, and Architecture (altexsoft.com)](https://www.altexsoft.com/blog/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/) (Accessed: 18 October 2024).

Hussein, S. (2021) Review of Web Service Technologies: REST over SOAP Review of Web Service Technologies: REST over SOAP (Accessed: 20 October 2024).