# What is API?

An API, or Application Programming Interface, is a set of rules that specifies how two software programs should interact with each other. APIs enable software developers to build programs that can use the functionality of other programs. For example, if a developer wanted to use a specific feature of an operating system or a piece of software, they could use the API provided by that software to access that feature. APIs are a way for different software programs to communicate with each other and share data and functionality.

# Why we need to secure our APIs?

APIs are used to allow different software programs to communicate with each other, and as such, they can often be accessed over the internet. If an API is not secured, it can potentially be accessed by anyone with an internet connection, which can lead to security vulnerabilities. Hackers could use an unsecured API to gain access to sensitive data or to disrupt the operation of the system that the API is a part of. In order to protect against these types of threats, it is important to secure APIs by implementing authentication and authorization measures, as well as other security measures such as encryption and input validation.

# What is API Gateway?

An API gateway is a software layer that sits between an API and its clients. Its primary purpose is to route API requests to the appropriate backend service and return the service's response back to the client.

API gateways can provide additional functionality such as:

* Load balancing: Distributing incoming API requests across multiple backend services to improve performance and availability.
* Caching: Storing the responses from backend services in memory to reduce the number of requests made to the backend and improve performance.
* Throttling: Limiting the number of API requests that can be made in a given time period to protect against denial of service attacks.
* Authentication: Verifying the identity of API clients before allowing them to access the API.
* Authorization: Controlling which clients or users are allowed to access the API and which actions they are allowed to perform.
* Monitoring: Tracking the performance of the API and its backend services and generating reports or alerts when issues arise.

API gateways can also act as a single entry point for multiple APIs, making it easier for clients to access the API and for API developers to manage the API.

# API Security Controls

# API Authentication:

API authentication is the process of verifying the identity of a client that is trying to access an API. This is important because it helps ensure that only authorized clients are able to access the API and perform actions on it.

There are several ways to authenticate API clients, including:

1. Basic authentication: Clients provide a username and password with each API request. This is sent in the header of the request in an encoded format.
2. Token-based authentication: Clients are issued a unique token that they must include with each API request. The server verifies the token to authenticate the client.
3. OAuth: An open standard for authorization that allows clients to access resources on behalf of a user without sharing the user's credentials.
4. API keys: Clients are issued a unique key that they must include with each API request. The server verifies the key to authenticate the client.

It is important to use a secure method of authentication to protect against unauthorized access to the API

# API Authorization:

API authorization is the process of controlling which clients or users are allowed to access an API and which actions they are allowed to perform. This is important because it helps ensure that only authorized clients or users are able to access the API and that they can only perform the actions that they are authorized to perform.

There are several ways to implement authorization for an API, including:

1. Role-based access control (RBAC): Clients or users are assigned to roles, and permissions to access the API or perform specific actions are granted to those roles.
2. Attribute-based access control (ABAC): Access to the API or specific actions is granted based on the attributes of the client or user, such as their location or job title.
3. OAuth scopes: When using OAuth for authentication, scopes can be used to control access to the API and specific actions.

It is important to carefully design the authorization model for an API to ensure that only authorized clients or users are able to access it and perform the appropriate actions.

# API Encryption:

API encryption is the process of encoding data transmitted between an API and its clients in order to protect it from unauthorized access. This is important because it helps ensure that sensitive data is not able to be intercepted and viewed by unauthorized parties while it is being transmitted over the internet.

There are several ways to encrypt data transmitted through an API, including:

1. HTTPS: This is a secure version of the HTTP protocol that is used to transmit data over the internet. It encrypts the data transmitted between the API and its clients using SSL/TLS (Secure Sockets Layer/Transport Layer Security).
2. SSL/TLS certificates: These are digital certificates that are used to establish an encrypted connection between the API and its clients.
3. API keys: These are unique keys that are issued to API clients and are used to authenticate and encrypt requests made to the API.

It is important to use encryption to protect sensitive data transmitted through an API and to ensure the privacy and security of API clients.

# API Rate-limiting:

API rate limiting is the process of limiting the number of API requests that can be made in a given time period by a client. This is used to protect APIs from being overloaded or subjected to denial of service attacks.

There are several ways to implement rate limiting for an API, including:

1. Fixed rate limiting: A fixed number of requests is allowed within a specified time period. Once this limit is reached, further requests are denied until the time period expires. It can be set to 1/10th of bandwidth for normal threshold and aggressive for critical APIs.
2. Token bucket: A client is issued a certain number of tokens that represent allowed API requests. Each request consumes a token, and when the client runs out of tokens, further requests are denied until more tokens are issued.
3. Leaky bucket: A client is allowed to make a certain number of requests per time period, but if the client exceeds this limit, the excess requests "leak" out and are not counted against the limit.

Rate limiting is an important security measure for APIs as it helps prevent them from being overloaded or subjected to denial of service attacks. It is important to carefully design the rate limiting strategy for an API to ensure that it is effective at protecting the API while still allowing legitimate clients to access it.

# API Input Validation:

Input validation is the process of checking that input data provided to an API is in the expected format and does not contain any malicious content. This is an important security measure because it can help prevent attackers from injecting malicious data into an API in an attempt to exploit vulnerabilities or access sensitive information.

There are several ways to perform input validation, including:

1. Data type validation: Ensuring that input data is of the correct data type (e.g. a string, a number, etc.).
2. Format validation: Ensuring that input data is in the correct format (e.g. a date should be in the format "YYYY-MM-DD").
3. Length validation: Ensuring that input data is not too long or too short.
4. Range validation: Ensuring that input data falls within a specified range (e.g. a number should be between 1 and 100).
5. Whitelist validation: Allowing only a predetermined set of values for a given input.
6. Blacklist validation: Blocking a predetermined set of values for a given input.

It is important to implement input validation on both the client side (e.g. in a web form) and the server side (e.g. in the API backend). This can help ensure that malicious data is not able to bypass client-side validation and reach the server.

# API Security Best Practices

Here are top API security best practices for developers:

1. Use secure coding practices: Follow secure coding guidelines to prevent common vulnerabilities such as injection attacks and cross-site scripting (XSS). This includes validating user input, sanitizing output, and using secure coding frameworks and libraries.
2. Implement authentication and authorization: Ensure that only authorized users can access the API by implementing authentication and authorization mechanisms such as API keys, OAuth tokens, or username and password.
3. Use transport security: Use transport security such as TLS (Transport Layer Security) to encrypt the communication between the API and the client.
4. Use rate limiting: Implement rate limiting to prevent abuse or denial of service attacks.
5. Monitor and test: Monitor the API for security vulnerabilities and perform periodic security assessments, penetration testing, and vulnerability scanning to ensure that the API is secure.
6. Keep the API and its dependencies up to date: Regularly update the API and any dependencies to ensure that they are secure and free of known vulnerabilities.
7. Use secure communication protocols: Use secure communication protocols such as HTTPS to secure the communication between the API and the client.
8. Use secure data storage: Store sensitive data in a secure manner, such as using encryption or hashing.
9. Implement access controls: Use access controls to limit access to sensitive data and functionality to authorized users only.
10. Use strong passwords: Use strong, unique passwords for all accounts associated with the API.
11. Use multifactor authentication: Implement multifactor authentication to add an extra layer of security to user accounts.
12. Use a firewall: Use a firewall to protect the API from external threats.
13. Use a content delivery network (CDN): Use a CDN to distribute the API across multiple servers and locations, improving performance and security.
14. Use a web application firewall (WAF): Use a WAF to protect the API from common web-based threats such as injection attacks and cross-site scripting (XSS).
15. Use secure data transmission: Use secure data transmission methods such as SSL or TLS to protect data in transit.
16. Use secure data deletion: Ensure that sensitive data is securely deleted when no longer needed.
17. Use encryption: Use encryption to protect sensitive data at rest and in transit.
18. Use secure coding frameworks: Use secure coding frameworks and libraries to help prevent common vulnerabilities.
19. Use secure development practices: Follow secure development practices such as code review and testing to ensure that the API is secure.
20. Use secure hosting: Use a secure hosting environment to host the API, such as a private cloud or virtual private server (VPS).