# Application Programming Interface (API) Enterprise Technology Guideline

## Introduction

## This document represents the official enterprise technology guidelines for Application Programming Interfaces (API) in the Department of Veterans Affairs (VA). An API is a set of protocols, routines, and tools that developers use to build software applications. APIs enable developers to build complex applications more quickly and efficiently by abstracting away low-level details of the underlying systems.

## API Standards

The current VA API Standards document is now maintained in the Catalog of Developer Essentials (CODE), which is an implementation of the Backstage service catalog tool:

[VA API Standards | CODE VA](https://code.va.gov/docs/default/component/va-api-standards/)

The official developer portal for VA APIs is in CODE VA, and developer.va.gov is the official location for APIs used by third-party applications:

* Internal APIs: [Software Catalog](https://code.va.gov/catalog?filters%5Bkind%5D=api&filters%5Buser%5D=all)
  + CODE VA requires user access through a GitHub account or visiting as a guest.
* Third-party APIs: <https://developer.va.gov>
  + Contains APIs that are available to external developers outside the VA firewall.

This section will be kept up to date to reflect changes to policy and guidance for both internal and external API development efforts.

## API Guidelines

* For APIs built in any environment/location, please document them in CODE VA. This can be done using the “Add an API” card on the main landing page ([Add to Catalog | CODE VA | CODE VA](https://code.va.gov/catalog-import)) for knowledgeable teams with more detailed help in the Starter guide.
  + Use the approved software catalog - [Software Catalog](https://code.va.gov/catalog?filters%5Bkind%5D=api&filters%5Buser%5D=all)
  + Ensure that developers are using only the approved [VA API Standards | CODE VA](https://code.va.gov/docs/default/component/va-api-standards)
* Search for existing APIs on CODE VA before going about building a new API. [Software Catalog | Browse the collection of APIs | CODE VA](https://code.va.gov/catalog?filters%5Bkind%5D=api&filters%5Buser%5D=all)
* If you are looking to build a new API or modernize one, follow the standards at [VA API Standards | CODE VA](https://code.va.gov/docs/default/component/va-api-standards)
* If you are looking to make an API available outside VA, there are additional expectations and governance provided at <https://developer.va.gov/api-publishing>

In general, the decision to make an API available outside VA should be a business decision, not at a technical one. Product teams are building APIs of a quality that is acceptable to third-party commercial consumers for all API, even those that stay inside VA.

## Portfolio-specific API Guidelines

Future versions of this document will include links to portfolio-specific API guidelines.

Internal API Catalog: [Software Catalog | Browse the collection of APIs | CODE VA](https://code.va.gov/catalog?filters%5Bkind%5D=api&filters%5Buser%5D=all)

Developer.VA.gov API Catalog: <https://developer.va.gov/explore>

Each catalog contains API entries that can be viewed and sorted to find the ones that best fit the business needs. Examples related to specific portfolios include Appeals, Benefits Intake, Clinical Health (Fast Healthcare Interoperability Resources), and Loan Guaranty.

## Appendix A: API Requirements (Reference Only)

The following sections provide requirements that product teams address when using VA APIs. These requirements are also documented in CODE VA along with the published APIs.

### **API Documentation Standards**

APIs enable business agility across VA. Maintaining a description of an API supports understanding of the API’s business context for both the development community and non-technical users. API documentation contains important notes about the API that provides definitive guidance to developers to help them code.

* All new API developments must be documented using the Open API specification, except for APIs that need lower latency or higher performance in transferring large amounts of data that cannot be accommodated by REST.
* API documentation must reflect pertinent methods (e.g., GET, POST, DELETE, PUT) and operations for the service that the API exposes, with the following exception: when it is determined for security reasons that access to all the Path Object Information must be subject to access controls.
* All new API developments should follow Representational State Transfer (REST), except for APIs that need lower latency or higher performance in transferring large amounts of data that cannot be accommodated by REST.
* Namespaces for APIs should refer to business entities that align to the Business Reference Model (BRM).
* When using structured data, use the JavaScript Object Notation (JSON) schema across databases, data, and Open API documentation to define access to an API.

### **API Management Standards**

* VA APIs should retain the same versioning, so that existing VA projects can continue to interface with the corresponding, underlying service.
* When an API is to be deprecated:
  + Mark older versions of APIs for retirement to address API version management.
  + Alert the application owners 180 days ahead of deprecation and work with them to enable application to use the new API version.
  + Alert developers through deprecation warnings.
  + Retire and remove deprecated APIs from the platform to avoid maintenance overhead.
* Provide a service level agreement (SLA) with at least 99% uptime in the production environment.
* Providers must configure preproduction environments (e.g., those for testing, user acceptance testing, development) to have at least 99% uptime.
* When providing an API and the underlying service, project teams must:
  + Document the intended API purpose, required data and data quality expectations.
  + Review a catalog of Authorities Data Sources (ADS) to determine ADSs for each data element.
  + Gather relevant API or data source artifacts (e.g., API documentation, data dictionaries, data models, reference data, business rules, data use, and disclosure requirements) to understand the candidate data, its associated requirements, and limitations.
  + Analyze the privacy implications of the data within APIs through a documented privacy threshold analysis (PTA) and privacy impact assessments (PIAs).
  + Define data quality requirements based on the desired use of the API.
  + Establish a Memorandum of Understanding (MOU) for data that includes the following: data access, frequency of access, volume of data covered, and data quality expectations that is permitted via API.
  + Project teams must use machine-readable and open formats for information collected or created.
* When consuming an API, project teams must:
  + Document the intended business use of know API consumers, including required data and data quality expectation.
  + Discover, understand, and implement an API data agreement (i.e., API data specification).
  + Establish a data sharing agreement and any other necessary mechanisms for initial agreement between the API consumer and the data source partner.
  + Obtain agreement from the Business Data Steward that data standards are identified and reused appropriately.
  + Obtain agreement from the Technical Data Steward that data quality is understood and maintained.
* Developers of APIs and related service developers must:
  + Monitor and log API traffic and performance to aid security posture.
  + Develop key business metrics (e.g., number of developers using the API, usage and adoption trends, user sources) on a case-by-case basis.
  + Use analytics to understand API usage and plan for improvements.
* Ensure that all APIs share a common schema or structure (e.g., common date time format, common image format, common record format) to reduce the number of service response structures that must be managed.

### **API Release Standards**

Using standard approaches when implementing, testing, and releasing APIs ensures consistency across the VA enterprise.

* Provide self-service access to APIs so that other developers can more easily integrate, test, and iterate.
* Create and use an automated build pipeline process to facilitate delivery of each version of the API and service.
* Health data must use the Health Level Seven (HL7) Fast Healthcare Interoperability Resources (FHIR) standard, and the Substitutable Medical Applications Reusable Technology (SMART) standard in API and service design.
* Use only standard data interchange formats in API implementations (e.g., JavaScript Object Notation (JSON), Extensible Markup Language (XML)).
* Run automated testing before each commit to the software's source code repository, and before each release.
* Separate testing activities from development activities to provide testers an objective position from which to evaluate an API and its underlying service.
* Test functionality on the VA API Management Platform's test accounts while developing each VA API platform-based application.
* All source code must be posted on the VA public GitHub instance ([Department of Veterans Affairs (github.com)](https://github.com/department-of-veterans-affairs).
* Include release notes with each release.
* Publish open data APIs in the VA Open Data Portal (https://www.data.va.gov/) based on VA's Open Data policy.
* For pre-production environments and developmental software, avoid personally identifiable information (PII), protected health information (PHI), or other protected information. One technique used to avoid data leaks is to use example data that is free of PII or PHI.

### **API Security Standards**

* When documenting APIs that process sensitive information:
  + Perform access control at each API endpoint for non-public Representational State Transfer (REST) services. This can be done with user authentication, authorization logic (e.g., using a message authorization header), and session management.
  + Connect external APIs through a VA-approved API gateway solution.
  + Encrypt all sensitive information in transit using Transport Layer Security (TLS) 1.2 or higher. Implementations may additionally use Internet Protocol Security (IPsec).
* When a request is not authorized, the API and service should return the authorization error (HTTP status code 401), instead of the file not found error (HTTP status code 404), to prevent an unauthorized mapping of the file and directory structure.
* If Cross Origin Resource Sharing (CORS) is enabled on the corresponding web server or API gateway, HTTP headers should include Access-Control-Allow-Origin set so that multiple origins are permitted.
* Employ techniques and tools that can be used to ensure API security and privacy such as:
  + ***Support the Zero Trust Architecture:*** Zero trust is a security model that assumes that all networks, devices, and users are inherently untrustworthy and must be verified and authenticated before they are granted access to any resources or data. The Zero Trust model is based on the principle of "never trust, always verify."
  + ***Enforce Authentication and Authorization:*** Authentication and authorization are the most important techniques to ensure API security. By requiring users to authenticate and authorize themselves before accessing the API, it ensures that only authorized users can access the API. This can be implemented using techniques such as OAuth2, JSON Web Tokens (JWT), IAM or API keys.
  + ***Enable HTTPS Encryption:*** HTTPS encryption ensures that all communication between the client and the server is encrypted, making it difficult for attackers to intercept or read the data in transit. HTTPS encryption can be implemented by using SSL/TLS certificates. Use TLS 1.2 or greater.
  + ***Perform API Security Audits and Penetration Testing:*** Audits and penetration testing can help identify vulnerabilities in the API and ensure that the API is secure. This can be done using tools such as Burp Suite, OWASP ZAP, or Nessus.
  + ***Enable Rate Limiting:*** Rate limiting is a technique used to prevent attacks by limiting the number of requests that can be made to an API over a certain period. This can help prevent DoS (Denial of Service) attacks and protect against brute-force attacks.
  + ***Enforce Input Validation:*** Input validation is a technique used to ensure that only valid data is accepted by the API. By validating input data, the technique can prevent attacks such as SQL injection or XSS (Cross-Site Scripting) attacks.
  + ***Partner with API Gateways******:*** API gateways act as a middle layer between the client and the API, providing an additional layer of security. They can perform functions such as rate limiting, authentication, and encryption.

## Appendix B: API Scaling Guidance

### **Scaling APIs**

* API management capabilities help manage API traffic and distribute it across multiple servers or instances. By using API management capabilities, the service can increase the capacity of the API infrastructure and handle more requests.
  + API management should be configured to provide the following services:
    - Act as a front end to backend services by accepting API calls and then routing them to appropriate backends.
    - Verify API keys and other credentials such as JWT tokens and certificates.
    - Enforce usage quotas and rate limits.
    - Transforms requests and responses in support of policy statements.
    - Caches responses to improve latency and minimize load.
    - Emit or Enable logging, metrics, and traces for monitoring, reporting, and general troubleshooting.
* Implement load balancing to distribute incoming API requests across multiple servers or instances to ensure that each server is handling a manageable load. Load balancing can help to increase the scalability of the API infrastructure.
* Leverage caching to help reduce the number of requests that the API infrastructure needs to handle by storing frequently accessed data in memory. By using caching, the VA can improve the performance and scalability of the APIs.
* Leverage serverless architectures to automatically scale the API infrastructure based on demand. By using serverless architectures, the VA can avoid having to manage servers and instances and rely on a cloud provider to handle scaling.
* Optimize database performance to store and retrieve data. Optimizing database performance to ensures that the APIs are running efficiently and can handle many requests.
* Monitor API performance to identify bottlenecks and other issues that may be impacting scalability. Use analytics tools to measure API usage, response times, error rates, and other KPIs to optimize performance and scale.
* Test the APIs for scalability. Use load testing tools to simulate high volumes of traffic and ensure that the API infrastructure can handle the load. Continuously test and optimize the APIs to ensure that they can scale to meet demand.

### **Scaling APIs for High Availability and Performance**

Scaling APIs for high availability and performance, there are several important considerations to keep in mind. Key strategies for achieving this are horizontal and vertical scalability.

Horizontal scalability involves adding more instances of an API to handle increased traffic. This can be achieved by deploying multiple instances of an API across multiple servers or containers and using a load balancer to distribute traffic evenly between them. To design for horizontal scalability, APIs should be stateless, meaning that they don't rely on data stored on a specific server or container. Instead, they should be able to access data from a centralized data store or database.

Vertical scalability, on the other hand, involves adding more resources (such as CPU, memory, or storage) to an individual API instance to handle increased traffic. This can be achieved by upgrading the hardware or infrastructure of a server or container running an API. To design for vertical scalability, APIs should be able to efficiently use the available resources, and not rely on any one specific resource being available.

Both horizontal and vertical scalability can help address API scalability challenges such as:

* **Traffic spikes:** When API traffic spikes, horizontal scalability can help by distributing the traffic across multiple instances, while vertical scalability can help by providing more resources to each instance.
* **Single points of failure:** By deploying multiple instances of an API across different servers or containers, horizontal scalability can help ensure that if one instance goes down, traffic can be automatically redirected to other instances. Similarly, vertical scalability can help ensure that if a specific resource fails, the API can still function using the remaining resources.
* **Bottlenecks:** APIs may experience bottlenecks when processing certain types of requests. Horizontal scalability can help by distributing these requests across multiple instances, while vertical scalability can help by providing more resources to handle these requests.

API integration challenges can also occur when scaling APIs. These include:

* **Data consistency:** When multiple API instances are deployed, ensuring that data remains consistent across all instances can be a challenge. One solution is to use a distributed database or data store that can synchronize data across all instances.
* **Service discovery:** When multiple API instances are deployed, it can be challenging to ensure that requests are routed to the appropriate instance. Service discovery solutions such as DNS or service registries can help address this challenge.
* **API versioning:** As APIs are scaled, it can become more challenging to manage changes and updates to the API. API versioning can help ensure that changes are made in a controlled manner, and that clients are able to continue using the API without disruption.

Design for horizontal and vertical scalability. Use appropriate solutions for integration challenges, APIs can be scaled to meet the needs of a growing user base and ensure reliable and consistent service.

### **Securing API Gateways**

## API Gateways are an intermediary inspection and management device, embracing a Zero Trust approach containing a Policy Decision Point (PDP) and a corresponding Policy Enforcement Point (PEP). Utilizing robust and granular access rules that can establish as strict of rules as possible to enforce the least number of privileges to perform a request. API Gateways also provide the ability to establish identity verification, federation, multi-factor authentication, continuous validation, and real time analysis. They can also provide additional security mechanisms such as DDoS prevention, web application firewall (WAF) capabilities, and support encryption in transit.

## Appendix C: Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Change Details** | **By** |
| 2/23/2023 | 0.1 | Original Draft | Mike Dance |
| 3/24/2023 | 0.7 | Updated version with edits based on feedback from OIS, AES, and SPM. | Mike Dance |
| 5/17/2023 and 6/20/2023 | 0.8/0.9 | Updated version that integrates content from the Health Technology Office API strategy. | Mike Dance |
| 7/26/2023 and 8/17/2023 | 1.0 | Final edits to Version 1, including moving the requirements to Appendix A and adding a placeholder for portfolio-specific guidance. | Mike Dance |
| April – June 2024 | 1.5 | Update for 2024 reflecting updated API standards and guidelines, and move to CODE VA | Mike Dance |
| Sept 2024 | 2.0 | Final version reflecting FY25 status and updates/corrections to the published version. | Mike Dance |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Disclaimer:** This document serves both internal and external customers. Links displayed throughout this document may not be viewable to all users outside the VA domain. This document may also include links to websites outside VA control and jurisdiction. VA is not responsible for the privacy practices or the content of non-VA websites. We encourage you to review the privacy policy or terms and conditions of those sites to fully understand what information is collected and how it is used.

**Statement of Endorsement:** Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, and shall not be used for advertising or product endorsement purposes.