API Producer Guidelines

These guidelines are provided to properly document, design and implement a REST API.

They are organized into 5 **themes** as point of views that will be the focus of distincts peoples.

Security rules ensure that you properly **authorize** your consumer applications and users, and that you **reduce risks of data leakage and compromission** from malicious people.

Security Officers will have a specific focus on this rules, as part of the *security first* approach.

> **Hint**
>
> Not all these guidelines apply to your perimeter !
>
> see SG Group API Guidelines and SGM API Wholesale Guidelines first.

Information¶

Properly declare and maintain your API in the Catalog, on the security panel.

Declare security information¶

Provide valid security information about your API as specified in Security

All this fields must be input : Application authentication, End user authentication, Data sensitivity, Data Integrity, Traceability Level, Critical Impacts, Network access.

The computed fields are not part of this rule : Criticality level, Security assessment

API-216 manual **Complete all security fields accurately** Catalog fields related to security information are valid, you may justify the data sensitivity in this review point comment or directly in API's description	Information

Make the Product Owner certifies the security fields¶

The security fields set in the Security Panel of the API card must be certified by the Business Product Owner of the API.

API-600 manual **Make the Product Owner certifies the security fields in API's card (cf: API-216)** The Product Owner is aware and accountable for the security information, checking this rule should be done by the PO directly	Information

Data¶

Document your operations & in/out data (interface contract) using OpenAPI.

Identify sensitive data¶

Some APIs are used in other teams or could even be exposed on the internet. In order to keep information segregated we need to identify sensitive data.

You should reference all sensitive data (categorized C2 or C3) in a glossary or with a specific marker.

API-252 manual **Identify sensitive data**
SG internal fields are identified explicitely, via Glossary or direct marker — Documentation

Identify regulatory contrains on your data from your Data Protection Officer¶

As part of european GDPR regulation and all other internal or external regulations, your Data Protection Officer (GDPR role), a Data Officer from GBSU/DAT for Wholesale, must be aware of the data managed by the API, and must have identified the one impacted by some regulation.

Do not expose on internet some SG only data¶

When an API is exposed on internet to non SG Clients/Partner, the API must not output to this organization any data which is specific to SG usage, even if this data is useless to them, no appearing in the documentation they see, or needed by SG entities consuming the same API.

Such SG internal only data must be identified with your Data Officer (see Identify sensitive data).

If same API is used by both non SG organization and SG entities, then the rights management must ensure to exclude from the output the SG only data, to the non SG client app or to the non SG user using an SG client app.

Another way is to build a specific API for the non SG usage, that will ensure to filter out any SG only output field, and also the invalid inputs in this use case, then call the normal API.

Examples of such SG only data is any internal referential ids, like Third Party BDR id for Wholesale, or SG staff GGI number and RH identifiers for the whole SG Group.

Expose your documentation on official Developer Portal¶

The documentation of your API and its formalized specification file (OAS/Swagger) should be exposed only to authentified developers.

For APIs used only internally within SG entities, and only available within SG Network, UI documentation and specification file can be freely exposed from the API itself, using OAS/Swagger standard library.

Note that the specification file MUST be made available to the Developer Portal (cf: Reference your swagger file in Catalog).

For APIs exposed to internet, the documentation UI and specification file must be exposed only via the official Developer Portal and not the API itself, nor via some other web site specific to API's application team.

Wholesale's official Developer Portal is https://developer.sgmarkets.com

API-605 manual **Use only the official developer portal if you want to expose your API documentation / contract to the internet**
developer.sgmarkets.com is the single allowed way to expose an API documentation / contract on internet for Wholesale APIs — Data

Correctly format the call logs with required information¶

The calls logs should contain a minimum set of information to be usable for security diagnostics, as per Book C in SG Code.

Fields to have in logs :

- Timestamp (UTC) : do not consider a default Timezone (like Paris), the timezone must be explicit via the ISO format or systems where datetime is stored as UTC
- HTTP request verb
- HTTP response status code
- HTTP request url path, but usually not the query string part as it may contain sensitive data
- Caller IP : if reverse proxies or gateway are between the original client and the API, the X-Forwarded-For header should be used to collect it
- Client id : the identifier of the calling application, usually it is the OAuth2 client_id (SG Connect)
- User id : if the request was initiated by a end user, some way to identify it, ex: SGConnect ID, Contact ID, email, ...
    - Beware to verify this information is not considered as too sensitive on your context (ex: GDPR)

Following fields are recommended but not mandatory :

- HTTP request body size
- HTTP response body size
- Request duration as milli-seconds

Note that other information may also be usefull but for other needs, as the user-agent when the client is a Browser to diagnose some issues

API-614 manual **Correctly format the call logs with required information** Fields to have in logs : caller IP, user id, client id, timestamp, duration, HTTP verb/path/status/sizes		Monitoring

Log for the needed retention period¶

You must define with the Business, Compliance and Regulatory people a set of activity information that are needed to be kept for some retention period.

If part of this information comes only from the logs of the API calls, then such logs must be kept available for all the duration of the retention period.

When the used centralized log system has a duration retention lower than your need, you then have to copy the needed log information to some other space.

Note that some cold storage may be used for long retention due to Compliance/Regulatory and not diagnosis/support needs.

API-601 manual **Log for the needed retention period defined with Business, Compliance and Regulatory** Log for the needed retention period defined with Business, Compliance and Regulatory		Monitoring

Delete logs older than retention period¶

Keeping logs for a too long period may be a security issue.

Once the retention period is defined (see Log for the needed retention period), and if such logs are not needed for diagnosis/support needs, then they must be purged.

Note that your capacity planning or business steering may require to keep long term aggregated information. Such statistical information usually does not need to be purged.

Eg: number of calls per operation per client_id per status code per week

API-602 manual **Delete logs older than the retention period** Delete logs older than the retention period	Monitoring

Authorize access to logs only to identified profiles¶

Logs contains information that may be C2, especially when then contain the identifier of the client application and optionaly of a end user, or when they contain some C2 business value within the url, or when log contains the full query and response body content.

So only authorized profiles must be allowed to see the detailled logs.

A good practice is by default to not log the request body, and for the response body to not log it on OK calls (2xx).

The aggregated statistical information should also by protected when one of the aggregate is a sensitive data for the related business (like the client app identifier or an end user identifier).

Note: SG Monitoring implements rights on its vizualization dashboards and underlying data repository.

API-603 manual **Authorize access to logs only to identified profiles** Authorize access to logs only to identified profiles	Monitoring

Authorization¶

Securize your API using OAuth2 and SG Connect.

Secure endpoints with SG Connect¶

Every SGM API endpoint needs to be secured using SG Connect based on OAuth 2.0 protocol, so at least the client program is clearly authentified and authorized to call the API.

This below example declares OAuth2 with implicit flow as security standard used for authentication when accessing endpoints. Additionally, three API access rights are defined via the scopes section for later endpoint authorization usage (see next section).

```
securitySchemes:
  oauth2:
        type: oauth2
        description: SG Connect supported flows
        flows:
          # you should document all supported flows
          implicit:
                authorizationUrl: 'https://sso.sgmarkets.com/sgconnect/oauth2/authorize'
                scopes:
                  openid: OpenIDConnect
                  profile: Profile
                  api.service-catalog.v2: Allow access to API v2
                  api.service-catalog.v3: Allow access to API v3
                  api.service-catalog.generate-credentials-for-client: Allow to generate credentials for
a client
```

Note that the openid and profile scope are not access rights, but related to user information.

The SGM API stack provides libraries to secure an API and interact with SG Connect. Check our tutorials Secure your API.

Incorrect

```
openapi: 3.0.1
components:
  securitySchemes:
        company-oauth2:
          type: oauth2
          scheme: Bearer
          flows:
                clientCredentials:
                  tokenUrl: https://identity.company.com/oauth2            => rejected not SG Connect
-----------------------------------------------------
openapi: 3.0.1
components:
  securitySchemes:
        company-api-key:
          type: apiKey                                                  => rejected not Oauth2
```

Correct

```
openapi: 3.0.1
components:
  securitySchemes:
        company-oauth2:
          type: oauth2
          scheme: Bearer
          flows:
                clientCredentials:
                  tokenUrl: https://sso.sgmarkets.com/sgconnect/oauth2/access_token
                  scopes:
                        api.foo.v1: access to all the resources of this API
```

API-350 auto-swagger-check **Declare OAuth2 security**
OAuth2 security is declared in OpenAPI specification (API-242 must be GREEN, else this rule may stay Pending) — Security

Use SG Connect for client app authentication¶

To authenticate a client application, and so be able to check it can legitimately call your API, use OAuth2 protocol, and SG Connect (for Wholesale APIs).

The client is identified by one client ID which is an opaque identifier (GUID).

The SGM API stack provides libraries to retrieve the OAUth2 token from the HTTP payload, to check it and retrieve related information from SG Connect.

API-352 manual **Client application is authenticated and authorized**
API uses a way to authenticate & authorize the client application, like Oauth2, SGConnect, Basic auth, SSL client certificate, ... — Security

API-353 manual **Client application is authenticated with OAuth2**
OAuth2 protocol is used to authenticate & authorize the client application — Security

API-354 auto-catalog-check **Use and declare SGConnect as client app authentication system**
Field "Application authentication" is set to sgconnect — Information

API-405 auto-catalog-check **Use and declare a recognized OAuth2 client app authentication system**
Field "Application authentication" is set to one of sgconnect, iamaas, oauth2-bsc, sgsignin — Information

Use SG Connect for user authentication¶

When you API needs to authenticate the human who is the immediate requester, through an user device client program (web, desktop, mobile, excel, ...) or who is at the origin of a chain of API calls, also use SG Connect and the Oauth2 protocols

Via SG Connect you can retrieve multiple information about the user : email, lastname/firstname, sesameid, windows login, sgconnectid, ...

Note

For client app: on the initial authentication flow triggered by the client, the scopes `openid profile` must be requested in order for user infos to be accessible by the called API.

See samples :

- C# : `this.User.UserInfo()` from TokenInfoController.cs
- Java : `(UserClaims) oAuth2Authentication.getOAuth2Request().getExtensions().get(SecurityConstants.USER_CLAIMS)` from RetrieveTokenInfoResource.java

Please consult the source code describing the available information on an user :

- C# : UserInfo.cs
- Java : UserClaims.java

API-356 manual **Authenticate user with SG Connect** End user is effectively always authenticated only via the SG Connect token	Security
API-357 auto-catalog-check **Use and declare SGConnect as user authentication system if user authent is needed** Field "End user authentication" is set to sgconnect or no-users	Information

Use SG Connect for user rights¶

When you need to implement some logic to authorize actions depending on the authenticated user, by blocking some API unitary operation, or even deeper logic, you can also use SG IAM user rights process (RBAC model).

You can use SG IAM to register and manage access rights. When you do so then the user access rights are retrieved when validating an user token. These claims are added to the existing user claims.

See also Use SG Connect for user authentication

Why?

Relying on SG Connect to manage at least coarse-grain rights removes you the burden of managing yourself this data and the approval workflow, and also makes this information easily visible and consistently manageable by security officers.

API-359 manual **Retrieve macro user rights from SG Connect** Macro user rights are retrieved from the SG Connect token, and provisionned by SG Identity	Security

Chain user authentication through API calls¶

Sometimes, the permissions or the whole authentication of an end user have to be checked in every component from Front to Back including the intermediate ones. So you should always consider that your API will be called within a chain of API calls. In such case, you must forward the end user's authentication via the related SG Connect process (delegation / token exchange).

Do not pass only part of the end user's identifier outside of the OAuth 2 token like setting his email in json payload or ad-hoc HTTP header, because this will not be considered as a trusted identity (not a strong authentication). Remember that only the SG Connect server can be trusted for retrieving the user's identity (a strong authentication).

Use official security stack

SGM API security stack should be used to interoperate properly with SG Connect. It provides several features to help you authenticate users and applications as well as securize your operations while operating intelligently with SG Connect. For instance:

- token renewal before expiraton time
- token validation retry
- token enrichment for user roles
- cache token to avoid requesting known and valid data from SG Connect

For more details see the tutorials.

> **Important**
>
> For automated rule API-363 to properly detect your usage of SGAPI Stack on sgconnect interaction, you must :
>
> - Use SG Monitoring for APIs (zipkin) with the most recent SGAPI Stack In Java you should use the latest available version matching your Spring Boot minor version or (3.4.2, 3.9.2 or >= 3.13.0)
> - Connect to SGMON with your own API's credential = from the credential tab within your API Card in the API Catalog, not from a Client App
>   - As temporary solution, you may also well declare the apiCode and apiVersion fields in your SGMON/APIs stack config

API-363 auto-catalog-check **Use the SG API stack to interop with SGConnect**
For proper detection : use latest SGAPI stack (ex: java 3.11) and connect to SGMON with a credential from your API card's credential tab (not from a client app) Security

Protection

Declare your HTTPS address

A Web API on production is accessible by consumers using only a unique base url, on the **HTTPS** protocol.

Calling an API must not require any initial "discovery" protocol from the client application. Your can manage multiple instances for resilience or scalability by using an intermediate L4 or L7 load balancer.

Declare your HTTPS base url within your OpenAPI specification.

Note that for APIs exposing their OpenAPI specification file directly from the API's instance, the base url of the API is also the base url of the OpenAPI file.

Incorrect

```
swagger: '2.0'
host: fashion.com
basePath: "/api"
schemes:
- http                              => http
-----------------------------------
openapi: 3.0.1
servers:
  - url: "http://inter.net/api"     => http
  - url: "inter.net/api"            => http if no protocol
```

Correct

```
swagger: '2.0'
host: fashion.com
basePath: "/api"
schemes:
- https
-----------------------------------
openapi: 3.0.1
servers:
  - url: "https://inter.net/api"
```

API-310 auto-swagger-check **Use HTTPS on your API's url**
When server field is specified in OpenAPI Specification then it is https (API-242 must be GREEN, else this rule may stay Pending) Security

Expose only on HTTPS¶

Security and data privacy is a mandatory matter on APIs.

On production, all API flows must be encrypted, and so an API is exposed from client point of view on standard HTTPS port (443). The only port allowed from APIM (API management Gateway) to SG LAN is 443.

API-364 auto-swagger-check **Expose only on HTTPS**
All declared environments flaggued as production use HTTPS on default port and uses a domain name (not a pattern identified as a server name) Protection

Respect OWASP security rules¶

Security and preventing Vulnerabilities is a mandatory matter on APIs. The OWASP foundation defines a : specific set of rules for APIs, which is more accurate for APIs than the original OWASP top 10 security rules made for Web Apps.

- A1 Broken Object Level Authorization
- A2 Broken Authentication => See our rule about proper OAuth2 usage, on Wholesale: SG Connect with SG Stack
- A3 Excessive Data Exposure => See our rule about sensitive data
- A4 Lack of Resources & Rate Limiting => See our rule about Quota
- A5 Broken Function Level Authorization => See our rules about client app and users rights, enforced by the SG Stack checks at operation level
- A6 Mass Assignment
- A7 Security Misconfiguration => See our rule about CORS, no internal data in error messages, proper configuration of OAuth2
- A8 Injection
- A9 Improper Assets Management => See our rules about proper documentation and versioning
- A10 Insufficient Logging & Monitoring => See our rules about monitoring

The original OWASP rules about Web Apps:

1. SQL Injection Attacks
2. Broken Authentication & Session Management
3. Cross-Site Scripting (XSS) Attacks
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure

7. Missing Function Level Access Control
8. Cross Site Request Forgery Attacks (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

Discuss me on Jive

API-365 manual **Respect OWASP security rules** OWASP top 10 rules are fulfilled, including the dedicated API parts, see details	Security

Do not expose internal info on error¶

Stack traces contain implementation details that are not part of an API, and on which clients should never rely. Moreover, stack traces can leak sensitive information that partners and third parties are not allowed to receive and may disclose insights about vulnerabilities to attackers.

So an API must not return any stack trace or any other internal information on errors to client. In case of server errors, an API may return some information explicitly meant for understanding and usage.

The details of internal errors should stay within the API logs/monitoring system.

Good example

```
500 Internal Server Error
{
  error: {
    code:"InternalIssue",
    message: "Sorry, some issue prevented to perform this operation",
    requestId:"1234532"
  }
}
```

Bad example

```
500 Internal Server Error
{
  error: {
      code:"OutOfMemory",
      message: "OutOfMemory Exception on SensitiveCodePartMethod() with arguments :
SensitiveField=SensitiveValue"
  }
}
```

You can still provide some information when the request is invalid:

Good example of invalid request

```
400 Invalid Request
{
  error: {
    code:"ExistingClientId",
    message: "An application with id my-reuse-app already exist",
    requestId:"1234532"
  }
}
```

API-366 manual **Do not expose internal info on error** Error messages (status 5xx) should not give internal information	Security

See also Use standardized error body

Retrieve secrets at runtime from a safe vault¶

The main recommended service is MyVault for on-premise APIs, and when on Kubernetes, using Kub Secrets is also OK.

Do not commit secrets in your source code. If you want to version your secrets then they should be encrypted with a strong key kept out of your code base.

Deploying secrets with your application means you will store them in several places and on the server. At the same time it is difficult to protect all access to a service especially on shared infrastructure (Virtual Machines, Cloud). Such secrets are easier to access thus less protected.

Changing regularly secrets is also more time consuming if they follow a long deployment process on several places.

For these reasons it is recommended to store secrets in a safe vault and recover them at runtime.

The secret protecting the access to the vault should be protected as much as possible:

- not committed in the source code management system
- stored on the server running the application (or as a secret for container platforms)
- accessible only by the account which runs the application
- kept (for minimum redundancy purpose) in personal vaults by a few responsible people.

API-367 manual **Retrieve any secrets at runtime from a safe vault**
Secrets needed by API to operate (as db credentials) are stored in a Vault (not in config files populated on deploy) Security

Use official pattern for internet exposure¶

Internet exposure implies more risk than internal exposure. As such you must use the standard tools provided to configure internet access to your application.

API-368 manual **Use official pattern for internet exposure**
API uses official internet gateway for its internet exposition Security

Fix issues identified by the Vulnerability Scan¶

The Vulnerability scan is a robot that test your API at runtime to identify vulnerabilities.

If such tool is to be run on your API as per its security context, then all High & Critical issues must be corrected before the initial production and any following production updates.

The frequency of execution of such tool will also depend on its security context.

API-607 manual **Fix all High and Critical vulnerabilities identified by the Vulnerability scan or Penetration tests** Protection
Fix all High and Critical vulnerabilities identified by the Vulnerability scan

Fix issues identified by the Penetration Tests¶

The Penetration tests is a process, usually requiring human intervention, that test your API at runtime to identify vulnerabilities.

If such process is to be run on your API as per its security context (eg: internet accessible API), then all High & Critical issues must be corrected, before the initial production and any following production updates.

The frequency of execution of such process will also depend on its security context.

API-608 manual **Fix all High and Critical vulnerabilities identified by the Penetration**

tests Protection
Fix all High and Critical vulnerabilities identified by the Penetration tests

Control API network flows¶

Some APIs, like the ones exposed on internet, are especially sensitive to malicious intrusion tentatives and must be isolated from the remaining of the network by being able to access only few explicitely specified network dependencies, so in case of compromission the intrusion does not spread easily.

Beware that current isolation mecanisms are usually **IP based** and so you must be well organized with dependencies managed by other teams to support potential IP changes.

Note that the component in the isolated zone must execute the functional logic of the API and not be just some passthrough component to a non-isolated backend API.

Allowed isolation mechanisms are :

- Public Cloud's isolated sub networks : Azure VNet, Amazon VPC subnet, ...
- Kubernetes Network Policies (egress rules)
- Security Groups : contact Khalid FAIQ or FR-GTS-GCR-GBS-PPS
- DTZone DMZ Back : contact Khalid FAIQ or FR-GTS-GCR-GBS-PPS

Contact your local IT Security representative for more information.

API-615 manual **Control API network flows**
Apply a supported isolation mechanism as : Azure subscriptions, Security Group, K8S Network Policies, DMZ Protection