Design of a Good API - Part 2

Tips and guideline on the design of APIs

Ayoub Ouarrak

16/10/2019

To the REST and beyond

Topics

- Definition of an internal Standard
- API resources organization
- Search, Pagination and Sorting
- API Gateway
- Security
- Backwards Compatibility and Deprecation
- Questions

Definition of an internal Standard

The process of API design is a very intellectual and artistic one, and since every one of us have a different view and work differently, is highly recommended (actually mandatory) for a team to define and agree on internal set of guidelines or standard that will help produce a **consistent** API.

Organize API resources

Focus on the business entities that the API exposes:

- Get **preferences** of a specific **user** $\implies$

$$\textbf{GET}/users/\{id\}/preferences. \tag{1}$$

- Get **tasks** of a specific **team** $\implies$

$$\begin{gathered}\textbf{GET}/teams/\{teamName\}/tasks\\ \vee \\ \textbf{GET}/tasks?team=teamName\end{gathered} \tag{2}$$

Organize API resources

More examples:

- Get **commits** from a specific **user** on a specifc **branch** $\implies$?

Organize API resources

- **GET** */branches/{branchName}/commits?user=username*

Organize API resources

- Create a **Pull Request** from **branch** *feature1* to *develop* on the **repository** *gitrepo-api* $\implies$?

Organize API resources

- **POST** */repositories/{gitrepo-api}/pull-requests*

Organize API resources

- Reject/Approve a specific **Pull Request** on the **repository** *gitrepo-api* $\implies$?

Organize API resources

- **GET** */repositories/{gitrepo-api}/pull-requests/{id}/reject*
- **GET** */repositories/{gitrepo-api}/pull-requests/{id}/approve*

Organize API resources

Lemma

We should group the first level of resources routes by the resource that we suppose will provide more actions.

Search, Pagination and Sorting

Search

Two main philosophies for the search:

- See the **Search** as resource itself.
  Example: **POST** */search/costellations*
- Use **Query** param.
  Example: **GET** */stars/search?magnitude=3.5&galaxy=andromeda*

Pagination

Big collections should be paginated, using query parameters to define the number of items per page and which page to retrieve:

- **?page=** specify the page of results to return.
  Example: *../stars?page=1*
- **?per_page=** specify the number of records to return in one request.
  Example: */stars?page=1&per_page=20*

Sorting

In addition to the pagination query parameters, sorting parameters can be used to order the result:

- **?order=** control whether results are returned in ascending or descending order.
  Example: *../stars?order=asc*, *../stars?order=desc*
- **?orderby=** control the field by which the collection is sorted.
  Example: */stars?order=asc&orderby=magnitude*

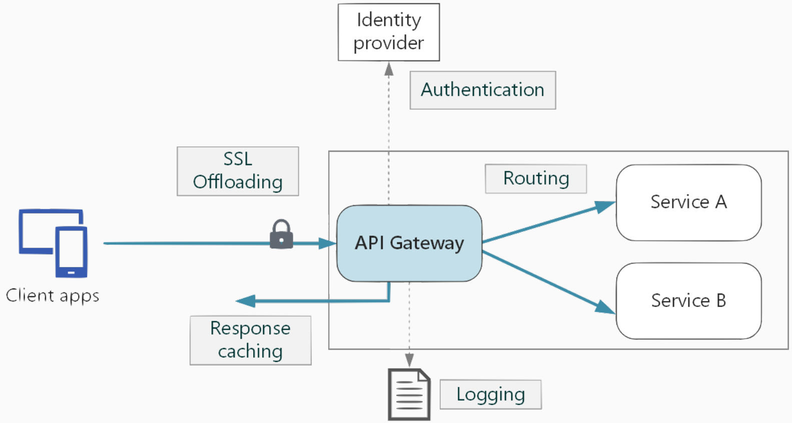
API Gateway

API Gateway

An API gateway sits between clients and services. It acts as a reverse proxy, routing requests from clients to services. It may also perform various cross-cutting tasks such as:

- Authentication
- IP whitelisting
- Client rate limiting (throttling)
- Logging and Monitoring
- Response Caching
- ...

API Gateway

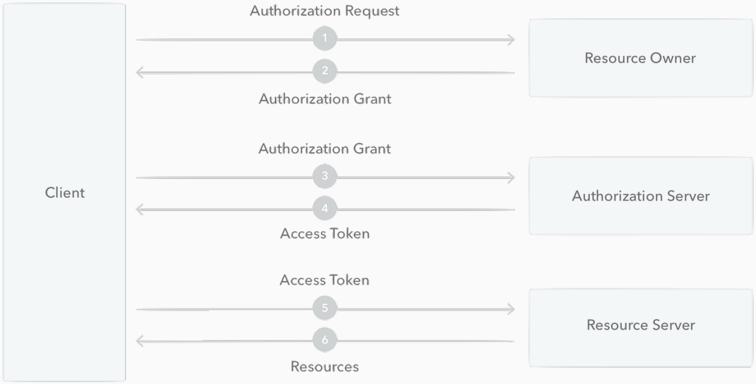

Identity provider
Authentication
SSL Offloading
Routing
Service A
Client apps
API Gateway
Service B
Response caching
Logging

Security

OAuth

OAuth is an open-standard authorization protocol that provides applications the ability for “secure designated access.”
For example, you can tell Facebook that it’s OK for MyShittyApp.com to access your profile or post updates to your timeline without having to give MyShittyApp your Facebook password. This minimizes risk in a major way: In the event MyShittyApp suffers a breach, your Facebook password remains safe.

How OAuth (2.0) works

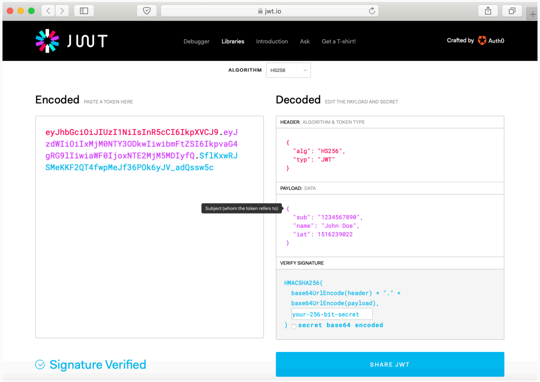


How OAuth (2.0) works

1. The Application (Client) asks for authorization from the Resource Owner in order to access the resources.
2. Provided that the Resource Owner authorizes this access, the Application receives an Authorization Grant. This is a credential representing the Resource Owner's authorization.
3. The Application requests an Access Token by authenticating with the Authorization Server and giving the Authorization Grant.
4. Provided that the Application is successfully authenticated and the Authorization Grant is valid, the Authorization Server issues an Access Token and sends it to the Application.
5. The Application requests access to the protected resource by the Resource Server, and authenticates by presenting the Access Token.
6. Provided that the Access Token is valid, the Resource Server serves the Application's request.

JWT

JSON Web Tokens are an open and standard (RFC 7519) to represent user's identity securely during a two-party interaction. When two systems exchange data, JWT identify a user without having to send private credentials on every request.



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJ
SMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

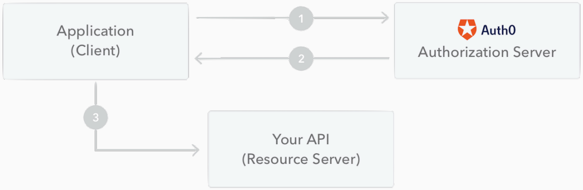
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Figure 1: JWT structure

How JWT works

1. The client requests authorization to the authorization server. This is performed through one of the different authorization flows. For example, a typical OpenID Connect compliant web application will go through the /oauth/authorize endpoint using the authorization code flow.
2. When the authorization is granted, the authorization server returns an access token to the client.
3. The client uses the access token to access a protected resource.



Common security tips

- Don't reinvent the wheel in *Authentication, Token generation*.
- Don't use *Basic Auth*, use instead *Oauth or JWT*.
- Use https to avoid MITM
- Sensitive data should be encrypted

Backwards Compatibility and Deprecation

Backwards Compatibility and Deprecation

APIs **SHOULD** be designed in a forward and extensible way to maintain compatibility and avoid duplication of resources, functionality and excessive versioning.

- All changes should be additive.
- All changes should be optional.
- Query-parameters and request body parameters MUST be unordered.
- The API specification **MUST** highlight the deprecated elements and the client must be aware of them.
- The API server should inform clients app regarding deprecated elements in the responses/requests at runtime.
- Deprecated elements must remain supported for the lifetime of a major version or until customers are no longer using them.

Questions?

References

- https://github.com/shieldfy/API-Security-Checklist
- https://docs.microsoft.com/en-us/azure/architecture/microservices/design/gateway
- https://medium.com/netflix-techblog/optimizing-the-netflix-api-5c9ac715cf19
- https://github.com/paypal/api-standards/blob/master/api-style-guide.md