Software Engineering

Part (IX)- API Design Principles

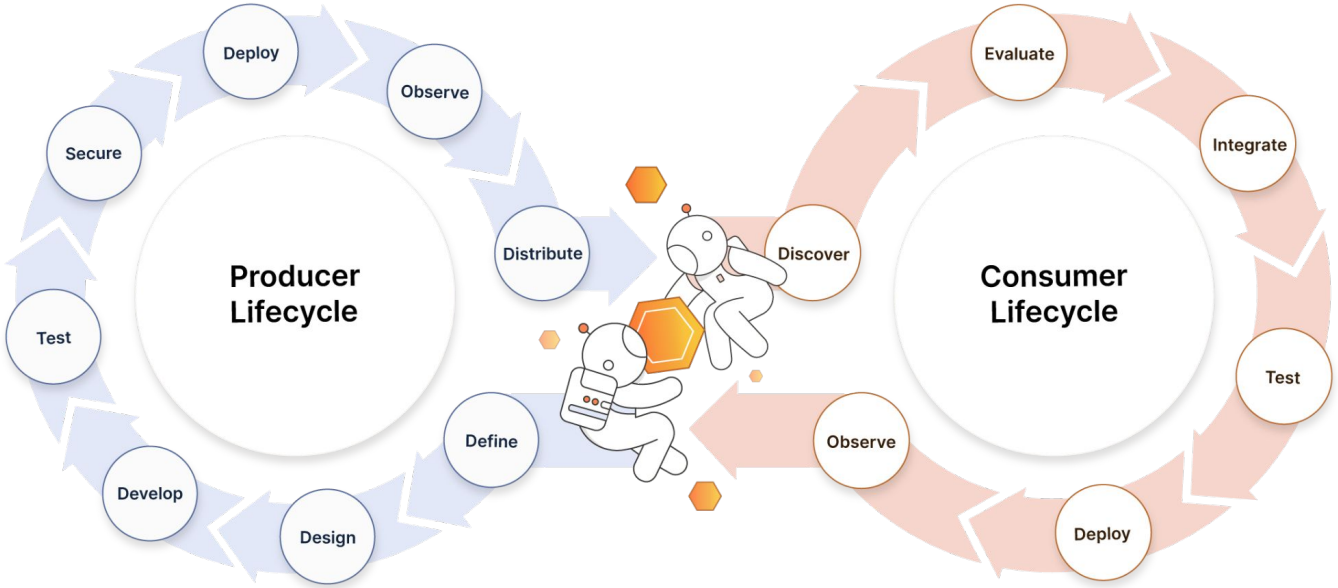
By: Mehran Alidoost Nia
Shahid Beheshti University, Fall 2023

Web API Design

- A well-designed web API should aim to support:
  - Platform independence
  - Service evolution
    - Add functionality independently from client applications.
    - As the API evolves, existing client applications should continue to function without modification.

API First Approach

- The API-first approach prioritizes APIs at the beginning of the software development process, positioning APIs as the building blocks of software.



In our State of the API survey, at least 75% of respondents agreed that developers at API-first companies are happier, launch new products faster, eliminate security risks sooner, create better software, and are more productive.

"The Postman Team"

API Design Life Cycle

- Requirement Engineering
- API Design
- API Review
- Implementation
- User testing
- Release
- Deprecate

API Design Steps

API Requirement Engineering

- Collect data with pair programming (current version)
- User experience research
  - What challenges do developers face when using our API reference docs?
- Exposure hours

What we're calling API-first [today], at Stripe, we call it developer-first because we're building for developers.

CJ Avilla, developer advocate at Stripe

API Review

- Steps in API Review
- Platform operations
- API product’s Integrity
- Developer experience (DX)
- Frontend tooling
- Security practices

API Release

- What are released?
  - Postman collections
  - Mock servers
  - SDKs
  - Documentation
- How are they released?
  - Beta release
  - Gated features

REST API

- Representational State Transfer (REST) as an architectural approach to designing web services.
- REST is an architectural style for building distributed systems based on hypermedia.
- REST is independent of any underlying protocol and is not necessarily tied to HTTP.

Design Principles of RESTful APIs

- REST APIs are designed around resources, which are any kind of object, data, or service that can be accessed by the client.
- A resource has an identifier, which is a URI that uniquely identifies that resource.

https://adventure-works.com/orders/1

Design Principles of RESTful APIs

- Clients interact with a service by exchanging representations of resources.
- Many web APIs use JSON as the exchange format.
- For example, a GET request to the URI listed above might return this response body:

{"orderId":1,"orderValue":99.90,"productId":1,"quantity":1}

Design Principles of RESTful APIs

- REST APIs use a uniform interface, which helps to decouple the client and service implementations.
- For REST APIs built on HTTP, the uniform interface includes using standard HTTP verbs to perform operations on resources.
- The most common operations are GET, POST, PUT, PATCH, and DELETE.
- REST APIs use a stateless request model.

Best Practices: Naming

- URIs should be based on nouns (the resource) and not verbs (the operations on the resource):

https://adventure-works.com/orders // Good

https://adventure-works.com/create-order // Avoid

- Avoid industry jargons.

Best Practices: Collections

- Entities are often grouped together into collections (orders, customers).
- A collection is a separate resource from the item within the collection, and should have its own URI.
- Sending an HTTP GET request to the collection URI retrieves a list of items in the collection (plural noun).

https://adventure-works.com/orders

Best Practices: Collections

- For example, `/customers` is the path to the customers collection, and `/customers/5` is the path to the customer with ID equal to 5.
- many web API frameworks can route requests based on parameterized URI paths, so you could define a route for the path `/customers/{id}`.

Best Practices: Associations

- Consider the relationships between different types of resources and how you might expose these associations.
- For example, the `/customers/5/orders` might represent all of the orders for customer 5.
- provide URIs that enable a client to navigate through several levels of relationships, such as `/customers/1/orders/99/products`.

Best Practice: Define API operations

- The effect of a specific request depends on whether the resource is a collection or an individual item.

Resource	POST	GET	PUT	DELETE
/customers	Create a new customer	Retrieve all customers	Bulk update of customers	Remove all customers
/customers/1	Error	Retrieve the details for customer 1	Update the details of customer 1 if it exists	Remove customer 1
/customers/1/orders	Create a new order for customer 1	Retrieve all orders for customer 1	Bulk update of orders for customer 1	Remove all orders for customer 1

Patch Method

- Suppose the original resource has the following JSON representation:

```
{
    "name":"gizmo",
    "category":"widgets",
    "color":"blue",
    "price":10
}
```

- Here is a possible JSON merge patch for this resource:

```
{
    "price":12,
    "color":null,
    "size":"small"
}
```

Best Practices: Filter

- Suppose a client application needs to find all orders with a cost over a specific value.
- It might retrieve all orders from the `/orders` URI and then filter these orders on the client side.
- Clearly this process is highly inefficient.
- It wastes network bandwidth and processing power on the server hosting the web API.

Best Practices: Filter

- Instead, the API can allow passing a filter in the query string of the URI, such as `/orders?minCost=n`.
- The web API is then responsible for parsing and handling the minCost parameter in the query string and returning the filtered results on the server side.

Best Practices: Paginate Data

- GET requests over collection resources can potentially return a large number of items.
- You should design a web API to limit the amount of data returned by any single request.
- Consider supporting query strings that specify the maximum number of items to retrieve and a starting offset into the collection.

```
/orders?limit=25&offset=50
```

Best Practices: Sort and Fields

- You can use a similar strategy to sort data as it is fetched, by providing a sort parameter that takes a field name, such as `/orders?sort=ProductID`.
- You can limit the fields returned for each item, if each item contains a large amount of data.
- You could use a query string parameter that accepts a comma-delimited list of fields, such as `/orders?fields=ProductID,Quantity`.

Asynchronous Operations

- Return HTTP status code 202 (Accepted) to indicate the request was accepted for processing but is not completed.
- You should expose an endpoint that returns the status of an asynchronous request.

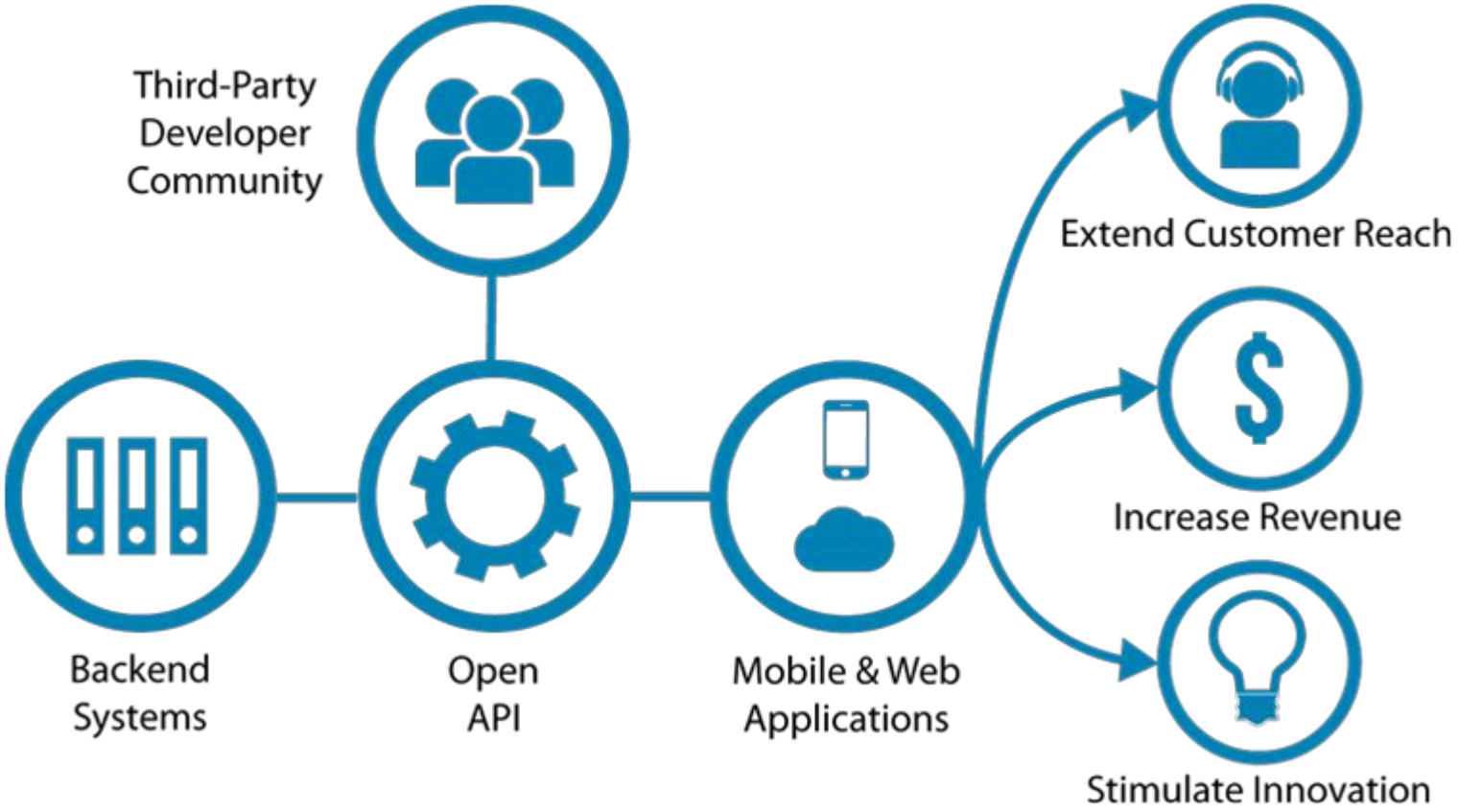
```
HTTP/1.1 202 Accepted
Location: /api/status/12345
```

Asynchronous Operations

- If the client sends a GET request to this endpoint, the response contains the current status of the request.
- Optionally, it could also include an estimated time to completion or a link to cancel the operation.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "status":"In progress",
    "link": { "rel":"cancel", "method":"delete",
"href":"/api/status/12345" }
}
```

Open API


Third-Party
Developer
Community
Backend
Systems
Open
API
Mobile & Web
Applications
Extend Customer Reach
Increase Revenue
Stimulate Innovation

Readings

- Robert C. Martin, "Clean Architecture", Chapter 25, 2017.
- Cloud Skill Challenge, "RESTful web API design,"Microsoft Blog, 2023.
- Postman Team, "Guide to API-first," Postman Blog, Last Access: December 2023.
- Michelle Bu, "Stripe's payments APIs: The first 10 years," Stripe Engineering Blog, December 15, 2020.