apigee

Best Practices in API Design

Google Cloud

Mission: The Most Important Design Influence

Always keep in mind why APIs are being implemented and who is using them?

Sample Mission:

Reduce time to market for new apps

- More than just wrapping existing services
- Development teams need:
    - Consistency
    - Sandboxes to test theories
    - Good Documentation
    - Uninterrupted Workflow
    - Stable Environments

Tips for Success

Spend 70% of your design effort on APIs that have consumers who can tell you if what you're designing is useful (or not).

Tips for Success

Spend 70% of your design effort on APIs that have consumers who can tell you if what you're designing is useful (or not).

Spend more time thinking about resources that are used, not actions that are taken. Nouns are good; verbs are bad.

apigee

Tips for Success

Spend 70% of your design effort on APIs that have consumers who can tell you if what you're designing is useful (or not).

Spend more time thinking about resources that are used, not actions that are taken. Nouns are good; verbs are bad.

Before designing a new API, first evaluate the current portfolio and see if a modification can be made to an existing API to satisfy the need.

More Tips

Non-functional requirements such as security schemes, SLA, or latency timing, while important, should not be allowed to impact the design.

Avoid the urge to implement custom anything. To the greatest extent possible, use conventions and techniques already in place.

An API is like a joke.
If you have to explain it, it's not that good.

Paraphrased Martin LeBlanc

Best Practice:

Noun-Oriented Resources

```
https://myserver/v1/dogs
https://myserver/v1/dogs/{dog-id}
```

- Keep primary resources to 2 levels
- Use plural nouns for collections
- Prefer concrete names over abstractions

Best Practice:

Force Verbs Out of URIs

Verbs in the URI cause a long list of URIs with no consistent pattern.

```
/getAllDogs
/verifyDogLocation
/isFeedNeeded
/giveDirectOrder

/getDog
/newDog
/getNewDogsSince
/getRedDogs
/setDogStateTo
/getAllLeashedDogs
/createRecurringMedication
```

```
/getHungerLevel
/getSquirrelChasingPuppies
/newDogForOwner
/transferDog
/doDirectOwnerDiscipline
/getRecurringFeedingSchedule
/isDogSick

/getDogsAtPark
/feedADog
/getSittingDogs
/checkHealth
```

Best Practice:

Use HTTP Verbs to Manage CRUD Operations

Resource	**POST** create	**GET** read	**PUT** update	**DELETE** delete
`/dogs`	Create a new dog	List matched dogs	Bulk update matched dogs	Delete all matched dogs
`/dogs/1234`	Error	Show "Fido"	If exists, update "Fido" If not, error!	Delete "Fido"

apigee

Best Practice:
Sweep Complexity Behind the ' **?** '

Relationships can be complex. Use query parameters instead of complex pathing.

- Which would you rather use?
- Which is more performant in bandwidth and compute constrained client environments?

```
GET /parks
For each park:  GET /parks/{park-id}/dogs
For each dog:   GET /dogs/{dog-id} (filter out dogs not running & not grey)
```

vs.

```
GET /dogs?color=grey&state=running&location=park
```

apigee

Thank You