API-DESIGN

DANIEL OSKARSSON

b02danos –> daniel@softwareyou.se

Theory, Challenges, Insights, Tools, DEMO, TLDR, Questions?

Theory

API

Application Programming Interface

Design

Spend time on designing your public interface (API)

I.e. gather a group of stakeholders (depending on the API preferably with different skills within the company)

Answer the question: What is the purpose of the API?

Find all resources/models/entities/classes/objects/structs (noun/verb)

For each resource decide what operations are applicable

Design (continued)

For each operation what input is required? optional?

For each operation what is the output?

How to handle input that was not expected?

How to handle errors happening inside the API?

How to handle an abnormal amount of requests?

How to handle authentication/throttling/caching/… ?

Design (continued)

Make sure that input/output/error handling is consistent

Iterate the design until everyone understands each decision

Opinionated

“Iterate the design until everyone understands each decision”

There are many many things to consider when designing an API. (I’ve just mentioned a few). Many things are highly opinionated. E.g. names in camelCase versus under_score.

A good API takes a stance in these questions so that the API is consistent, and while not everyone may agree, everyone should understand decisions before the design is complete.

Versioned

After iterating the API design it’s time to implement it.

However, the API will not stay the same forever

Assign a version to the API

- Clients will know which version they use
- Newer version of the API can be introduced while still maintaining the old one

Your API should ALSO be ...

Subject to the principle of least surprise

To the point (KISS)

Self-explanatory (hypermedia anyone?)

Consistent (enough with the consistent part, we get it!)

Service-Oriented APIs

(SOA)

SOAP

REST

Define Service-Oriented APIs

WSDL

Swagger

RAML

API Blueprint

WADL

XML Schema

JSON Schema

JSOND

REST (Normally HTTP + JSON)

Resources/Paths

Methods

Parameters (Path/Query/…)

Body

Headers

Status (codes)

Body

Headers

Challenges

Input and Output

Choosing the right level of abstraction for the input and output is often harder than it seems.

In a normal object-oriented API the options are often to either provide an object/struct/tuple, or multiple parameters as input and an an object/struct/tuple or a single return value as output.

The same challenge exists in Service-Oriented APIs.

Spend time thinking about input and output! Be consistent!

Authentication/Authorization

Depending on the type of API it may be a good idea to required clients to authenticate.

OAuth and other ways of doing this is out of scope for this presentation. In short:

Credentials (username+password) are used to generate a token, that is then provided as a header in each request.

“Authorization: Bearer ABCDEF01234567890”

Caching

If invoking the API is considered expensive it may be a good idea to supply cached responses.

The biggest challenge with caching is to find the right balance between providing up-to-date data and still off-load the actual API.

As always with caching, when to invalidate it?

Documentation

Documentation is often overlooked and thrown in as an afterthought.

Preferably the output of the iterated API-design should be the documentation itself. Keep it up to date.

There are ways to document the design so that stubs can be generated from the documentation itself. Use such features when available.

Make sure the documentation available to those who need it.

Insights

Insights

There are a few things that relatively simple

Other things are challenges (as just discussed)

Time and effort spent on API design is well invested!

Carefully review and validate the API documentation before implementation starts

Follow insights spread throughout this presentation

Tools

Tools

---

cURL

Postman (version 2 and 3)

Proxy (cannot recommend anyone in particular)

SoapUI (developed for SOAP, also does REST)

./jq (command-line json processor)

Integration Tests!

DEMO

TLDR

TLDR

Remember that the API is a Public Interface

Iterate the API design before implementing it

Carefully consider all available methods for each resource

Make sure input, output, and error handling are consistent

Assign an API version, at least on breaking changes

When applicable use authentication, throttling, and caching

Questions?

References

References

---

http://curl.haxx.se/

https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop

http://www.soapui.org/

https://stedolan.github.io/jq/

References (continued)

---

http://swagger.io/

http://raml.org/

https://apiblueprint.org/

http://json-schema.org/

http://jsond.org/