PRAGMATIC CODERS

OSEC

# Architecture decision making

...

Kamil Szymański
@kszdev

# Goal

Improve the way we take architectural decisions by showing some of the things that often get skipped when making such decisions

# Context

- SaaS
- Targeting larger enterprises
- Web & API versions of the product
- In the future there will be a mobile app
- Mostly web clients but as well few API ones
- HTTP(s) everywhere
- Microservices
- Cloud

# Disclaimer

Don’t focus on decisions that we will make, focus on how and why those decisions are made and what is missing from the picture

Whatever decisions are made here they might not be right in your context

This talk won’t simulate the real process of making such decisions, I’ll make huge simplifications but I hope you get the point

# New requirement

Add API versioning

# API versioning schemes

- Query parameter
  - URI =>  /api/customers?version=1
- Resource path
  - URI => /api/v1/customers
- Custom header
  - headers => { X-API-version: 1 }
- Media type
  - headers => { Accept: application/vnd.com.example.v1+json }

# API versioning schemes

- Query parameter
  - URI =>  /api/customers?version=2
- Resource path
  - URI => /api/v1/customers
- Custom header
  - headers => { X-API-version: 2 }
- Media type
  - headers => { Accept: application/vnd.com.example.v2+json }
- API/service domain
  - domain => api2.example.com

# Problem space first

First focus on the problem, not on the solution

What we want to achieve, what problem are we trying to solve

Knowing why should lead to a better how

Check if we adding API versioning is something we really need or just one of the possible solutions or even an unnecessary complexity added to the platform

# Why?

Our clients want a stable API

- no one wants to work on snapshots provided by 3rd party

Applications using our APIs are outside of our control

- we want to be able to evolve our APIs and have possibility to drop some of the legacy

# What?

Some APIs are exposed to clients some are be internal

# How granular?

Do we version the whole platform under a single version or do we have separate versions for separate contexts/services?

# Where?

Multiple contracts

- API clients, web app, mobile app, ....
- is there an API gateway somewhere?
- is some middle-tier solution supporting the web app?

# Testing support

Having support for testing is one thing, ease of testing is another thing

How do we support testing multiple API versions?

Does deprecating or dropping deprecated API affect tests?

How does releasing new API version affect tests?

# Tech stack support

What is supported out of the box?

What is easily supported without tons of hacks?

Quick PoC

# API versioning solutions

- ~~Query parameter~~
- Resource path
- ~~API/service domain~~
- Custom header
- Media type

# Versioning scheme

How about semantic versioning?

# Scope of versioning

Contract

- inputs, outputs, response codes

Which changes require releasing new version?

- breaking changes?
  - e.g. removing a field, making optional field mandatory
- non breaking changes?
  - accepting/returning more data (new fields)
- adding new endpoints?
- behavior changes?
  - e.g. changing business rules or adding new validations

# From client's standpoint

What is our deprecation policy?

- when do we deprecate?
- how long do we support deprecated versions?

How do we document and expose APIs to our clients?

# What we require from clients

Clients should explicitly specify the API version to be used

- if they don't shall we error out?
  - how do we error out?
  - will there be a transition period where not specifying a version will default to some version?
- what if the specified version is invalid or no longer exists?
- what if the requested version is valid but request payload does not fulfill it's contract?

# Documentation

- The decision itself
- Why THIS decision was made
  - what other solutions were evaluated
  - what criteria did we evaluate against
- How and when to release new API version
- When and how to deprecate API version
- Code examples

# Context is king

You’re not Netflix / Google / Twitter

Make research but evaluate everything in your context

Start from learning the problem space before jumping to solution space

# Impact on clients and you

Make it pleasant to use (and test)

Limit added complexity

Stay pragmatic

# Taking the best decision

What does “the best” mean?

- not only pros and cons
- short/long term gains/costs
- data-driven

# Lowering the costs of decision process

Do as much as possible asynchronously

If a meeting is needed have a clear goal and an agenda, take and publish notes

Come prepared and give people time to prepare

Document every decision and important facts

# What we touched on during API versioning evaluation

- Why?
- What?
- How granular?
- Where?
- Testing support
- Tech stack support
- Versioning scheme
- Scope of versioning
- From client's standpoint
- What we require from clients
- Documentation