THE TEN COMMANDMENTS OF API VERSIONING

Sponsored by Rex Software

ABOUT ME - JODIE

- Senior dev at REX Software working on the backend team
- I build API's, micro services, and work on infrastructure
- Big focus on establishing and exploring API standards
- twitter.com/seriouslyjodie

GENESIS

In the beginning the ARCHITECT created the perfect RESTful API

- An API is created
- To production!
- Time to make some updates though

EXODUS

“Let my API be free”

- We want to make changes
- But we have have existing clients and consumers
- People probably aren’t using that endpoint anyway
- Maybe we could just sneak in a few changes

R.I.P.
YOU

FIRST COMMANDMENT

Thou shall not break your consumers

- You have a CONTRACT with your consumers
- Yes, things change but it shouldn't be a surprise
- When things change in a backwards-incompatible manner, you need a new version

SECOND COMMANDMENT

Thou shall not commit breaking changes to your API without versioning

- You should have a good reason to update your API
- Your clients may use your API in ways you can't imagine
- If you make a breaking change, serve a new version
- Properties can be supplemented but not changed or removed

THIRD COMMANDMENT

You shall choose a versioning scheme

- Decide on a versioning scheme right from the start
- Version via URL
- Version via `Accept` header
- Version via custom header
- Implement SEMVER major.minor.patch

VERSIONING VIA URL

```
/v2/people
```

- 🙏 Easy to use - just give someone the URL
- 🙏 Bookmarking, navigable
- 🙏 Point it at a different branch
- Purists: “URL’s should represent the resource”

VERSIONING VIA ACCEPT HEADER

```
Accept:“application/vnd.myapp.v2+json"

Content-Type:“application/vnd.myapp.v2+json”
```

- 🙏 Accept already used to negotiate content
- 🙏 IETF legitimised this approach in RFC4627
- 🙏 Semantically makes sense
- Less shareable
- Send back the same `Content-Type`

VERSIONING VIA CUSTOM HEADER

```
X-Api-Version:“1.0.0”
X-Api-Version:“1,0.0”
```

- Same problems as Accept header
- Not a standard
- 🙏 More customisable (eg. Timestamp)
- Send back the same `X-Api-Version`

WHY NOT BOTH?
memegenerator.net

FOURTH COMMANDMENT

Invalid versions shall throw errors

- Don’t just assume latest version for an un-versioned request
- Give feedback
- HTTP Status Code (400)
- Application code: INVALID_API_VERSION

FIFTH COMMANDMENT

Implement semantic versioning

- SEMVER: major.minor.patch
- Bug-fixes update patch
- Non-breaking features update minor
- Breaking changes update major

SEMVER

```
Client  X-Api-Version:“1.5”
Server  X-Api-Version:“1.5.10”
```

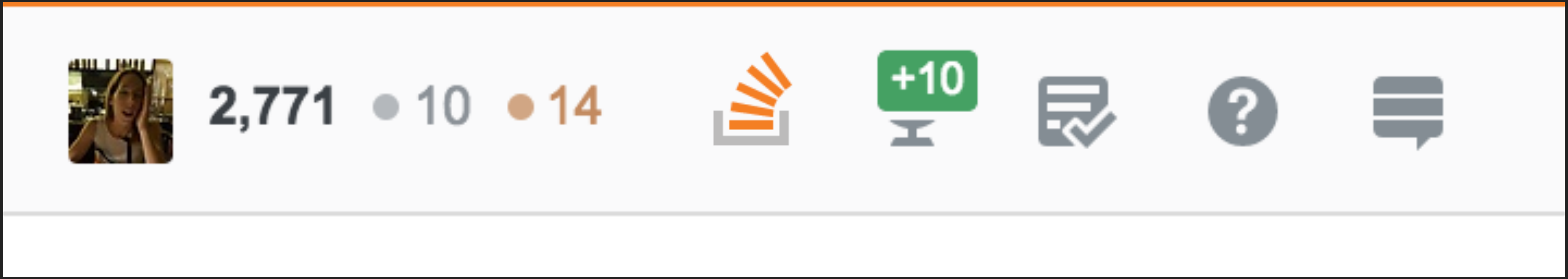
- Your clients can request a minor version
- You respond with a full version
- 🙏 Gives fine granularity
- Might be harder to maintain

SIXTH COMMANDMENT

Have a stable contract

- Whatever you do, provide stability
- Be practical
- Establish processes; breaking changes, sunsetting
- Other arguments don't really matter: RESTful, semantic, standards, URL sucks, headers are lame

That me



“Just delete your API right now, and start again.”

SEVENTH COMMANDMENT

Changes shall be well documented

- CHANGELOG
- Version announcements
- Updated API documentation; Swagger, Blueprint etc

EIGHTH COMMANDMENT

You shall steal from thy neighbour

- Who does API's well?
- Look at others for inspiration; Stripe, Twilio ~~Twitter~~
- Others have already gone through this torment before you
- Take what works and see if it applies to your API's

STRIPE

- Major versions use URL. Eg. /v1
- Server code can be remapped
- Accounts get pinned to the latest version at the time of their first request
- Versions are encapsulated via a transformer, resource types, and documentation
- Stripe-Version header is included with all responses

BUT YOU SAID ...
imgflip.com

NINETH COMMANDMENT

Transformations are godly

- Don’t return your models/entities as is
- Use a transformation layer to manage changes between versions
- Your API version can dictate what transformations are applied

TENTH COMMANDMENT

Thou shall not change your API

- … Unless you really have to
- API's evolve
- Make your changes backwards-compatible
- Versioning is difficult