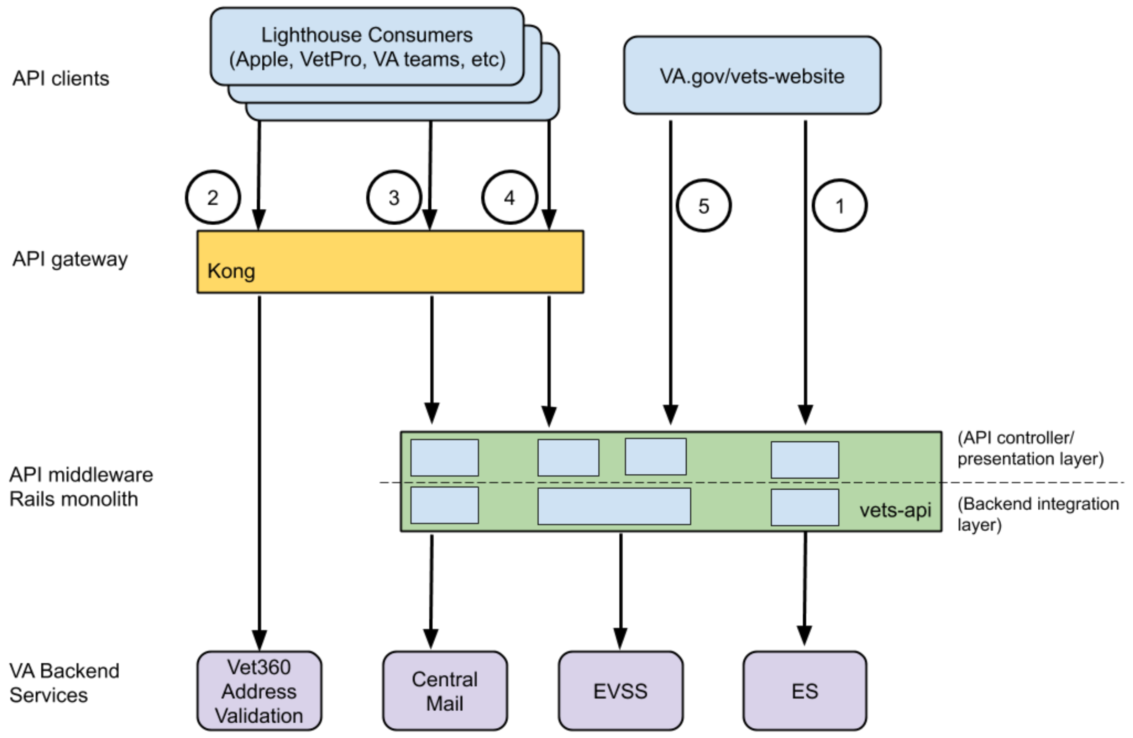
This document contains a discussion on the following topics pertinent to a VA mobile app:

- Lighthouse and VFS APIs
- Versioning
- OAuth


API clients
Lighthouse Consumers (Apple, VetPro, VA teams, etc)
VA.gov/vets-website
2
3
4
5
1
API gateway
Kong
API middleware Rails monolith
(API controller/ presentation layer)
(Backend integration layer)
vets-api
VA Backend Services
Vet360 Address Validation
Central Mail
EVSS
ES

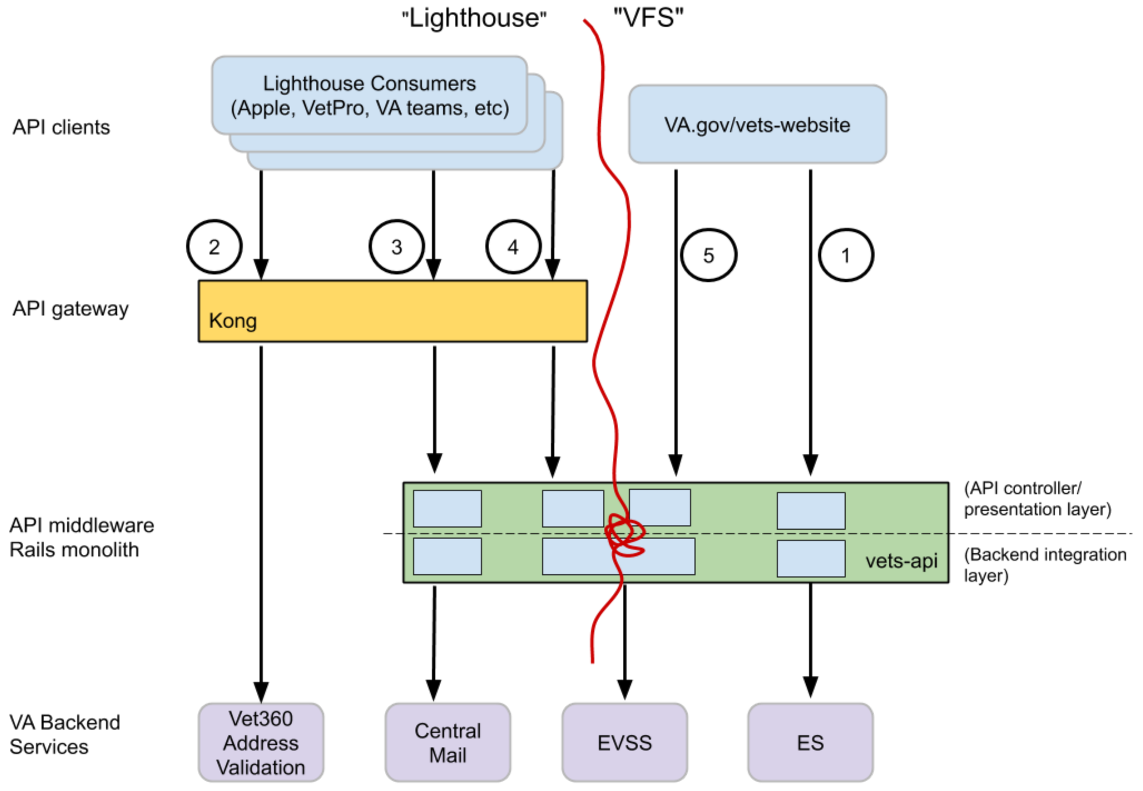
Five classes of APIs:

- **[1]** APIs that are for use by VA.gov and whose end-end implementation and operation are done by VFS teams.
  Examples: Healthcare Application Enrollment Service, MHV Rx/Secure Messaging

- **[2]** APIs that are part of Lighthouse, no connection to vets-api. Kong enforces access control and rate limiting, proxies directly to backend system.
  Examples: Address Validation, Loan Guaranty

- **[3]** APIs that are part of Lighthouse, proxied via vets-api, but are not used at all by VA.gov.
  Examples: Benefits Intake API via Central Mail gateway

- **[4 and 5]** APIs that are part of Lighthouse (4) or VA.gov (5), that serve different consumers but both proxy via vets-api and share some common implementation and talk to the same backend systems.
  Examples: Claims, Appeals

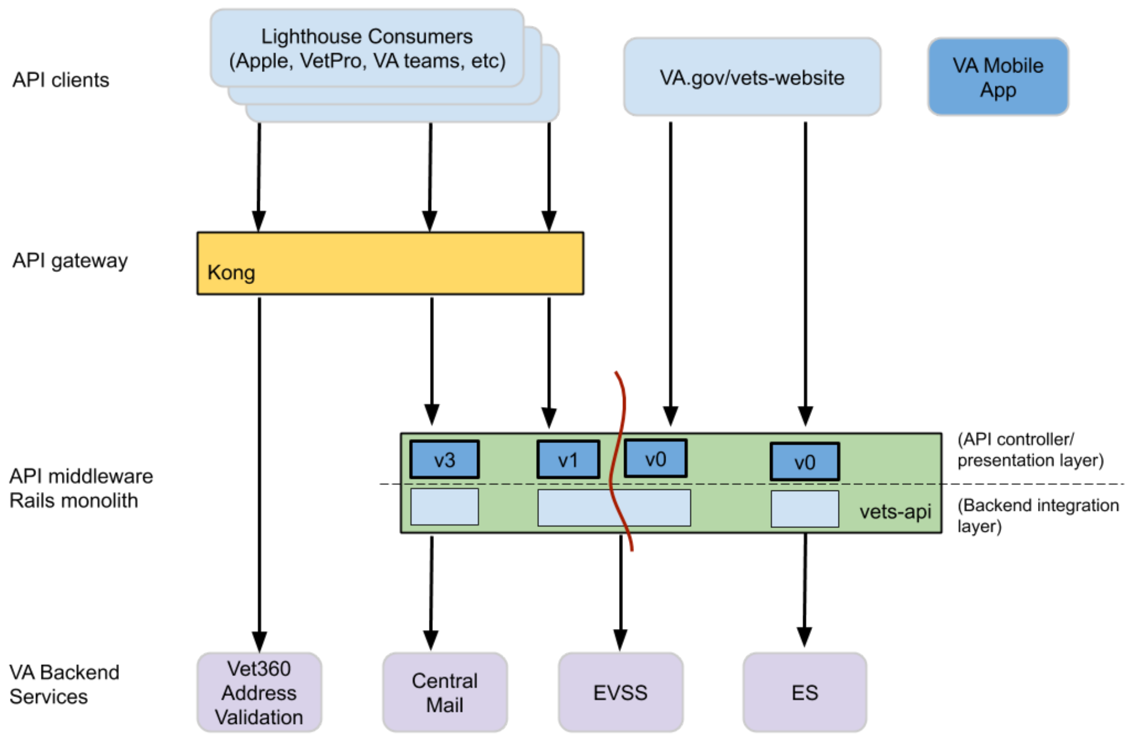

"Lighthouse"
"VFS"
API clients
Lighthouse Consumers (Apple, VetPro, VA teams, etc)
VA.gov/vets-website
2
3
4
5
1
API gateway
Kong
API middleware Rails monolith
(API controller/ presentation layer)
(Backend integration layer)
vets-api
VA Backend Services
Vet360 Address Validation
Central Mail
EVSS
ES

Lighthouse vs. VFS

- Nothing magical about "in Lighthouse" vs. "in VFS"
- Different consumers, different but overlapping sets of APIs
- Shared code base in vets-api
- Ownership/responsibility gets fuzzy in shared library code - relies on human-human communication/culture to maintain correctly.

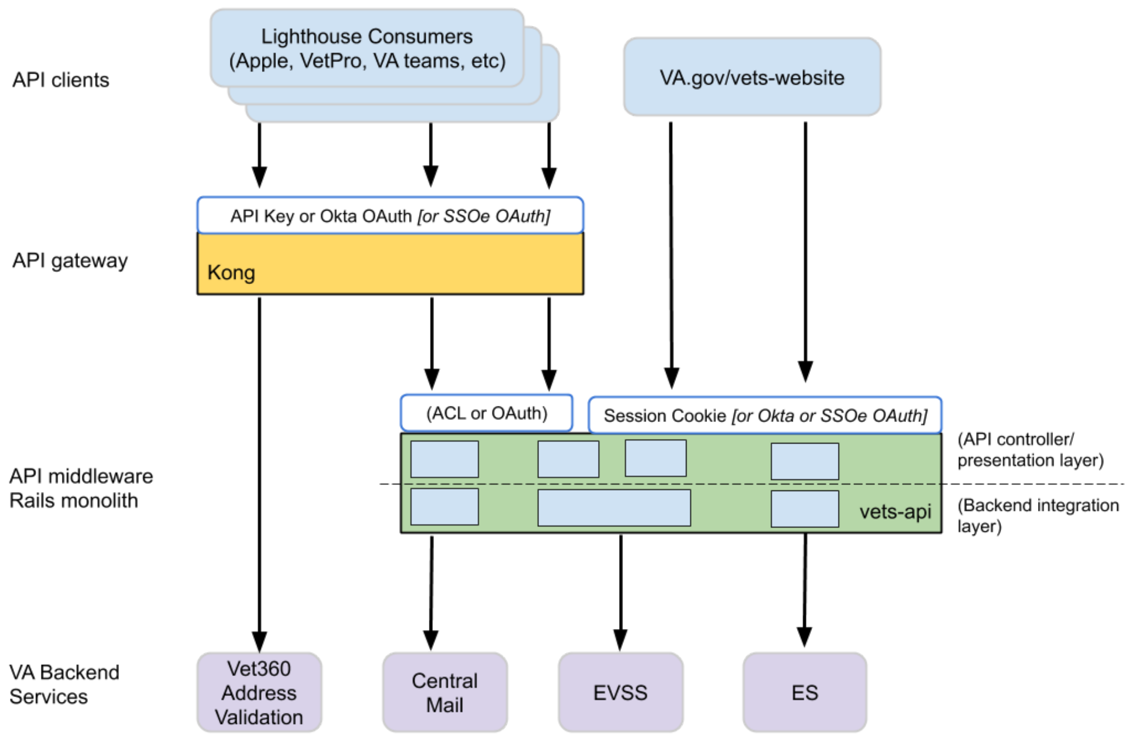
So if a mobile app needs an API, the question is not "Lighthouse vs. VA.gov". Instead:

- Does this need a Rails middleware implementation at all to make backing service usable by mobile? If yes, probably makes sense to build in vets-api. Especially if talking to one of many services that vets-api already talks to.
- Can call it "Lighthouse" or not based on desire to have other consumers use it.
- If no middleware is needed, then Lighthouse provides an easy way to make otherwise-internal APIs usable externally, and layer on access control.


API clients
Lighthouse Consumers (Apple, VetPro, VA teams, etc)
VA.gov/vets-website
VA Mobile App
API gateway
Kong
API middleware Rails monolith
v3
v1
v0
v0
vets-api
(API controller/ presentation layer)
(Backend integration layer)
VA Backend Services
Vet360 Address Validation
Central Mail
EVSS
ES

Versioning

- API versioning **at technical level** is an identical concern regardless of how you classify the API.
- Maintaining "n" versions incurs some cognitive drag on developers as APIs evolve.
- That said, there is a cultural difference:
  - Lighthouse program has the default notion that APIs are public and have many consumers, so no breaking changes are allowed
  - VFS program has the default notion that there is one API consumer - vets-website, and the developers of that are on your sprint team. If you need to make a breaking change, just coordinate with your teammate.
  - In practice, most developers are disciplined, breaking changes have been rare, and I don't think re-setting the mindset of VFS developers would be a big lift.


API clients
Lighthouse Consumers (Apple, VetPro, VA teams, etc)
VA.gov/vets-website
API gateway
API Key or Okta OAuth [or SSOe OAuth]
Kong
API middleware Rails monolith
(ACL or OAuth)
Session Cookie [or Okta or SSOe OAuth]
(API controller/ presentation layer)
vets-api
(Backend integration layer)
VA Backend Services
Vet360 Address Validation
Central Mail
EVSS
ES

Auth

Two intermingled questions:

1. Okta vs. SSOe OAuth
2. If OAuth needed for a new API, how/where/how much work?

Step 1 - Token issuance

- Strictly happens between mobile client and authorization server (Okta or SSOe)
- Implementation-wise "should be" standards-based (but devil is in the details)
- Okta vs. SSOe:
  - End user experience will differ (SSOe OAuth will have AccessVA look-and-feel)
  - Integration process will differ for dev teams (SSOe OAuth project management slog)
  - Changes/enhancements dependent on IAM dev process vs. working w/ Lighthouse program
  - This would be first external app integration with SSOe OAuth
- End result: app can request token, user authenticates/consents, auth server gives back token.

Step 2 - API interaction

- App makes an API request, and puts the token in a header.
- API endpoint (aka resource server) now has to look at the token and say "yes or no" to allowing access.
    - Is token current, or is it expired?
    - Is token issued and signed by an entity I trust?
    - Does the scope of the token match up with the operation being requested (type of data, read vs. write, etc)
- Typically this involves making a call back to the authorization server to say "is this token really valid? And by the way, can I trade this for the user's information?"
- If the overall access answer is "yes", then the API endpoint can proceed, and it has the user information needed to process the rest of the request.
- Means writing a module (say, 2 sprint's worth of work?) to enforce that access decision
- Might need to be available for both Kong and vets-api (for type 2 vs. other types of APIs)
- We've already built this for Okta OAuth, usable in both Kong and vets-api (e.g. FHIR APIs, Military Service History API)
- Building the same for SSOe OAuth would follow similar patterns but definitely new code.
- The other thing nobody has done yet is make a single API endpoint accept two different types of auth information - API key or OAuth, Cookie or OAuth, Okta OAuth or SSOe OAuth
    - Definitely doable, but not quite as simple as plugging in all the existing pieces of code.