FORGE DevCon 2017

FDC131261

Best practices when structuring an API heavy app

Galia Traub
Autodesk

Dorothy Thurston
Autodesk

Learning Objectives

- The importance of separation of concerns
- Managing your secrets
- Building a resilient application
- Caching strategies
- Considerations when testing your application

Description

What I wish I knew when I first developed an app using APIs - getting the architecture of your app right is important - and if you are building an app that consumes APIs this is can be even more critical. By using techniques to separate out the code that connects to the API from what matters most - your business logic - you can easily build testability, resiliency, and flexibility around the API you consume - while keeping your business logic free from API changes. You will leave this talk with examples of why this separation is important, specific examples of patterns to use, and what to anticipate could go wrong when good patterns are not used.

Your Forge DevCon Experts

Galia has written software for Autodesk products that heavily rely on Forge and other APIs. Through that process she has accumulated strategies for dealing with the complexities that are inherent in API-driven applications. She joined Autodesk as part of an acquisition with the goal of bringing about culture change and more technical rigor. She has always viewed coding as a social experience and strongly believes that empathy for fellow developers goes a long way to a joyful and productive coding experience.

Dorothy works on Autodesk cloud products that feature heavy API traffic from third party integrations and internal consumption. She loves implementing best practices, writing clean code, and helping people make anything.

Separation of Concerns

Clear separation of concerns is key to a well-designed codebase. This principle should be applied when consuming an API. It is important to keep code related to the API's logic separate from the business logic of your application code. Pulling this code out into a class or module with one dedicated purpose allows for easy code reuse, prevents duplication, and makes refactoring simple. Common API related logic that should be pulled into a separate section of code includes any logic related to authentication for the API, formatting of API requests, and response parsing.

Secrets Configuration & Management

Never hardcode passwords, API host URLs, keys, or secrets into your code base. Not only can this be a security concern, these values typically change depending upon environment (development, staging, production etc.). Cloud hosting solutions have strategies to manage these sensitive values. If you have an existing code base, check to see how database passwords are managed. You may already have the right tools in place to configure and manage your application secrets in a safe and simple way.

Amazon Web Services

S3 secrets bucket

Azure / Microsoft

Azure Key Vault

Other

Hashicorp Vault: https://www.vaultproject.io/

Resilient HTTP

Even the best APIs will occasionally return intermittent failures. This could be caused by a problem with the network, or with the API itself.

Retry with exponential backoff

Typically, failure is temporary and your best course of action is to simply retry. If this API call is blocking an end user, time to retry should be short by design. However, if the failing API call does not cause an end user to wait, you can choose to retry over exponentially longer intervals.

Circuit Breakers & Graceful Degration

Consider the safest and most resilient behavior your application can perform if an API that it depends upon becomes unavailable for an extended period of time. The circuit-breaker pattern can prevent your application from fruitless retries when the dependency is hard down. Martin Fowler has a good summary here: https://martinfowler.com/bliki/CircuitBreaker.html

.NET:

Polly - https://github.com/App-vNext/Polly

JavaScript

async - http://caolan.github.io/async/docs.html#retry
breaks - https://www.npmjs.com/package/brakes
Hystrix - https://www.npmjs.com/package/hystrixjs

Tech stack agnostic (proxy)

linkerd - https://linkerd.io

Java:

Hystrix - https://github.com/Netflix/Hystrix

Caching Strategies

If your dependency is slow, or has rate limits, consider implementing a local cache solution to keep a reliably accessible copy of necessary data. Some key questions to ask yourself when considering caching as a solution are:

- Is this data that needs to be real time?
- Is it data that does not change frequently, or does not need to be updated all the time to be usable?
- Do you need updated results or is your performance more important?
- How should we know to update data in the cache?

Testing your App

A test suite that depends upon an external API will be flakey; tests will occasionally fail if the network or API are having problems. Additionally, because a test depends upon at least one round-trip HTTP request, the test suite will be slow. Stubbing external HTTP calls will make your test suite fast and deterministic.

'Cassette' playback

Prerecord your test suite's HTTP interactions once and replay them during all future test suite runs rather than actually hitting an external endpoint.

.NET:

Betamax.NET - https://github.com/mfloryan/Betamax.NET
Scotch - https://github.com/mleech/scotch

JavaScript

VCR.js - https://github.com/elcuervo/vcr.js
Nock VCR - https://github.com/carbonfive/nock-vcr

Others:

See https://github.com/vcr/vcr 'Ports in Other Languages' section

Mock server
Configure your application to point to a locally running webserver acting as mock of your external dependency when in test mode. This mock dependency is tech stack agnostic, and could be implemented using any web app framework.

Questions to ask:

What is the safest & most resilient behavior when dependency goes down?
Is there data you should you cache & what strategy should you employ to refresh your cache?
Should you retry requests & at what interval?
What security compliance matters to your application?
How should you protect your secrets?
Are there places you can reduce code duplication?
Is you API specific logic separate from your business logic?