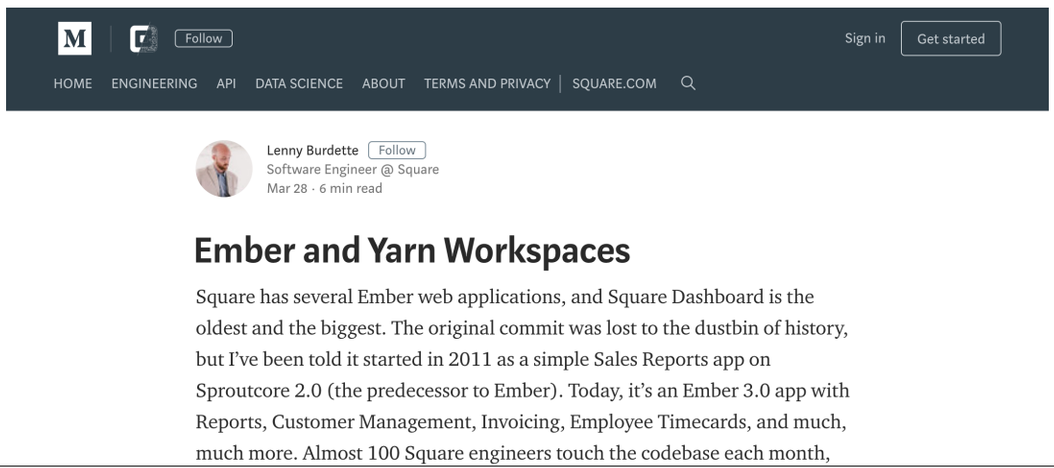
SCALING & MAINTAINING

Ember Monoliths

whoami

- github: lennyburdette
- ember discord: lennyburdette

https://medium.com/square-corner-blog/ember-and-yarn-workspaces-fca69dc5d44a



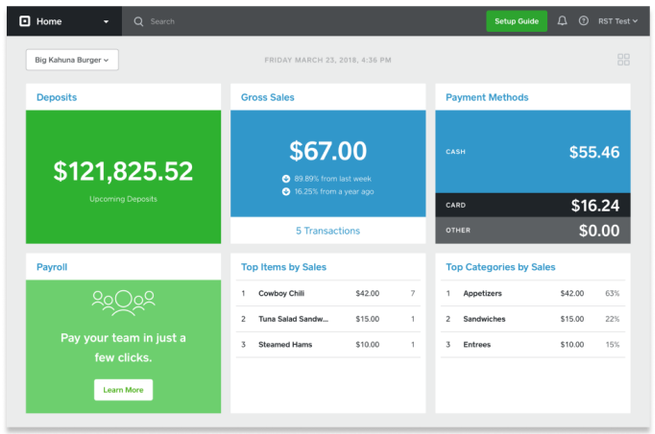
https://medium.com/square-corner-blog/ember-and-yarn-workspaces-fca69dc5d44a

Outline

- Square's Ember monoliths
- Problems we face
- Demo app walkthrough
- How to convert your app to a monorepo
- Caveats & lessons learned

Dashboard

- Public facing back-of-house business management app
- Started in 2011 on Sproutcore 2.0
- 350,000 lines of JavaScript
- 600,000 lines of test code
- 377 routes, 1514 components
- Around 80 contributors per month
- Hundreds of pull requests per month
- Daily deploys



- Not the biggest, not the oldest, but a unique combination of both
- A platform, lots of sections, many teams operating independently
- Emphasis on continuity of merchant experience
- Can’t slow down feature dev with a rewrite

Regulator

- Internal-only app for investigations and customer support
- Started in 2013 on Ember 1.0.0.rc7
- Still on Ember 1.13 & Sprockets asset build
- 50,000 lines of JavaScript
- 102 routes, 246 components, 195 views (!)

- Also a platform
- Continuity of experience isn’t as important

Regulator UI

- Started as a Yarn workspace in April 2018
- One application
- Three engines (so far)
- Handful of addons

- A chance to apply lessons learned in Dashboard to a brand new app

Problems

Technical

- Long build times in development
- Slow and brittle CI
- Monolithic deploys/rollbacks

- Big apps are slow to compile
- Lots of tests in one test suite mean more chances for pollution or flakeyness
- Can’t solve monolithic deploys/rollbacks today

Architectural

- Single namespace leads to long names, or:
- DRYing code too early, leading to:
- Tight coupling across unrelated parts of the app

- Overly specific naming conventions, but more commonly:
- Attempts to share code too early—premature abstraction
- One team's changes can affect another team

Social/Cultural

- Unclear code ownership
- Overwhelming to newcomers
- Various “best practices” exist at once
- No place for experimentation
- Long waits for code review

- Folders with hundreds of files are untenable
- Can’t refactor the whole codebase every time best practices improve
- Paralysis caused by not wanting introduce a 4th or 5th way to do something

Composition

How to make a large UI manageable? Break it down into constituent parts and compose it together. Partials are a poor unit of composition. Components are great: powerful API requires effort to learn, but flexible and composable.

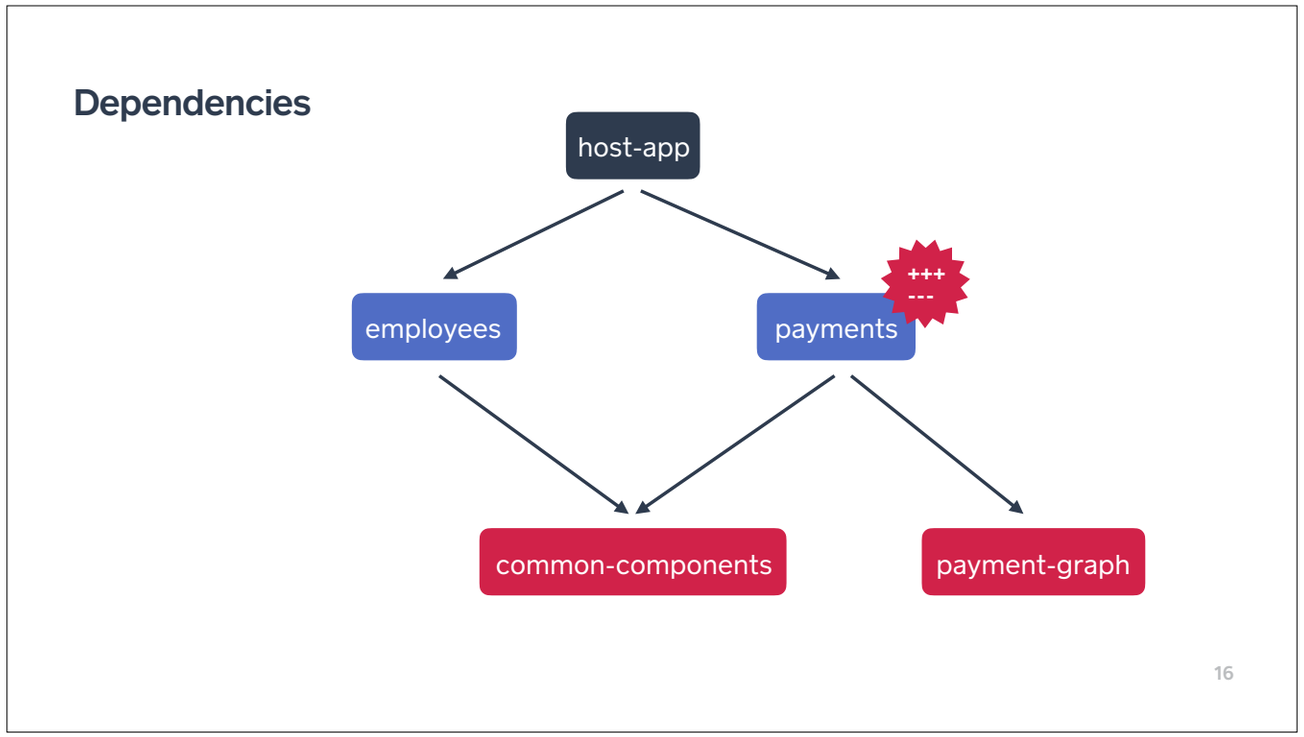
How to make a large app manageable? … Naming conventions are a poor unit of composition. Addons/engines/NPM packages are great.

Conway’s Law

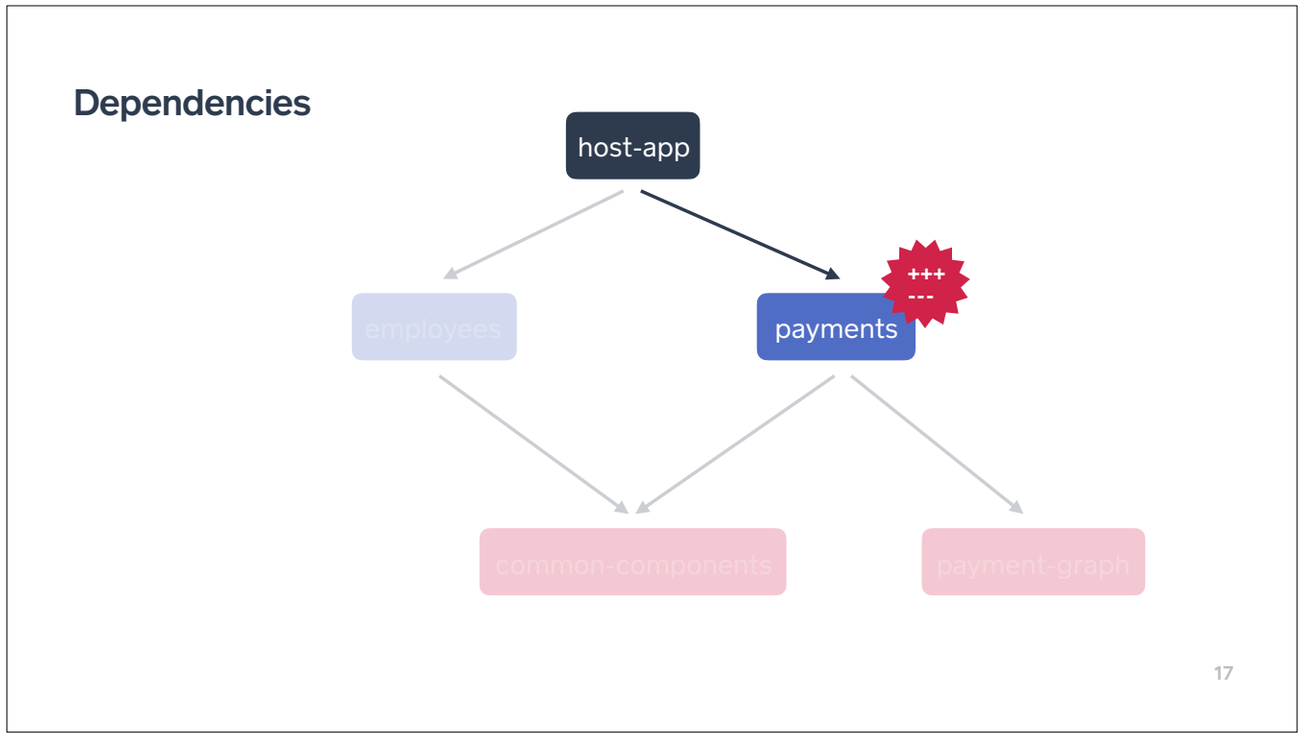
"organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."

- We should take advantage of Conway’s law and break up our apps across team lines.

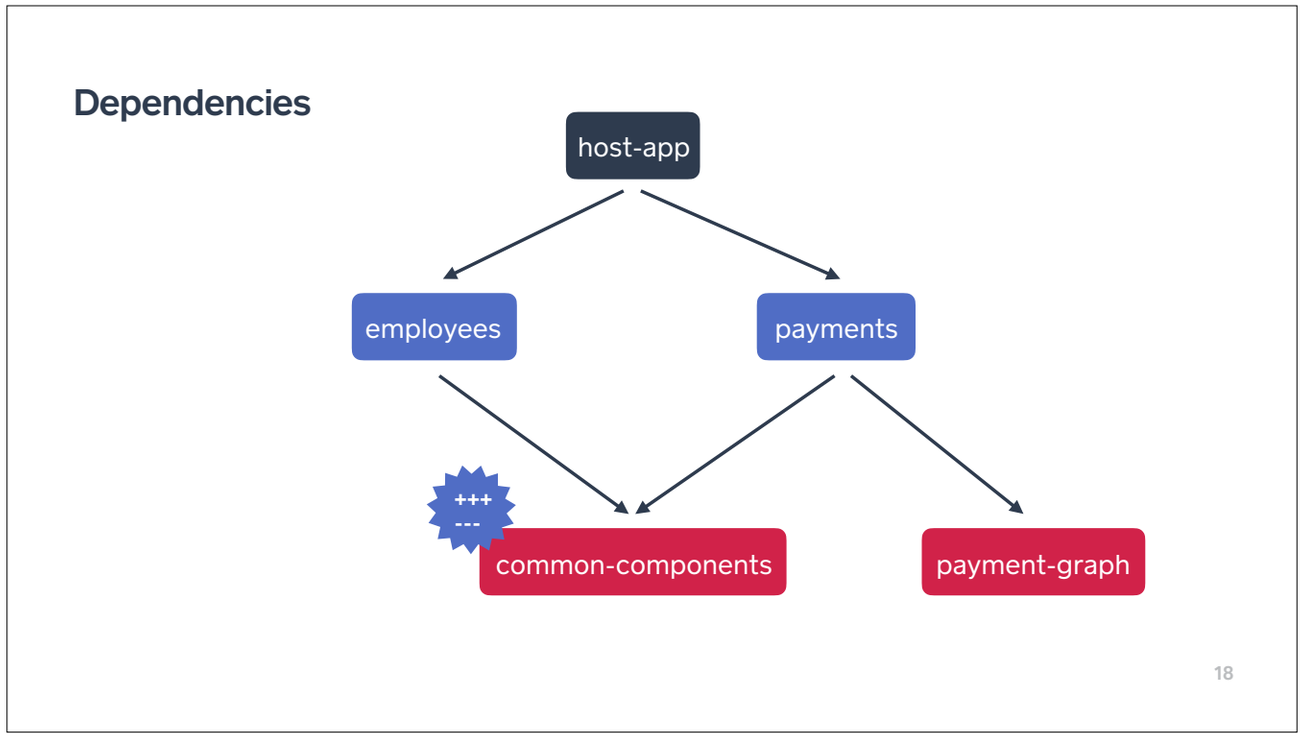
DEMO



This is the dependency graph of our packages. If we make changes only to the payments engine…

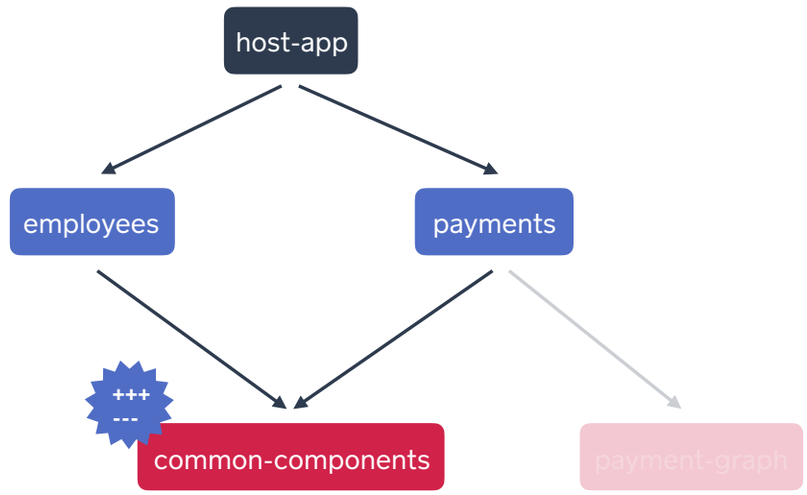


We only need to run tests for the payments engine and anything that depends on the payments engine.



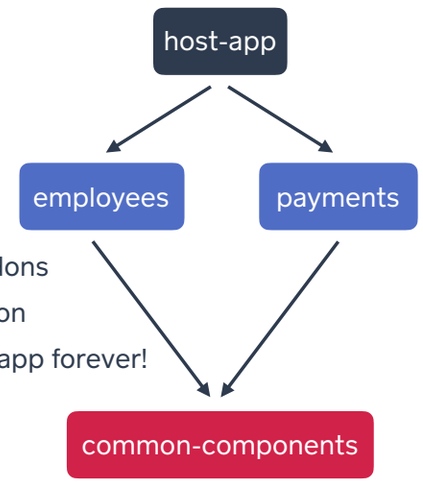
Shared addons will still run lots of tests. This is a good thing.

Dependencies



Breaking apart your monolith

- Convert your project to a Yarn workspace
- Remove unnecessary coupling (especially global state)
  - Copy-paste is not a sin
- Identify diamond dependencies
- Convert common components/services/utilities into addons
  - You can leave tests in the host app during the migration
  - You should leave (some) acceptance tests in the host app forever!
- Convert routes to engines



Caveats

- Engines still have foot-guns, especially with test APIs
- Boilerplate (dotfiles, dummy apps, etc) is repetitive and challenging to maintain
- Need ember-cli-dependency-lint to catch dependency version conflicts
- You need strategies for CSS isolation (CSS Modules?)
- ember-cli-update and ember-cli-typescript don't work in Yarn workspaces yet

Final Thoughts

- Extracted addons and engines are publishable!

I've focused on making a single application easier to manage as it grows. But by splitting up the app into packages, we have another benefit: our common code becomes sharable throughout the company. The rewritten Regulator app uses Dashboard's UI components, dramatically speeding up development.

github.com/lennyburdette/ember-monorepo-demo

square.com