GOURMET SERVICE OBJECTS™

The very thinnest of classes

MODELS VS SERVICES

MODELS

- Resources
- A collection of attributes that models some data/state
- Can touch the database — but doesn't have to (ActiveModel rocks)
- May include associations, scoping, and so on

SERVICES

- **Never** resources
- Simple!
  - No attributes, no accessors, no internal state
  - Thus no need for instances!
- “Method” object — do-ers
- Can be universal — “pure functions”

THE NATURE OF A SERVICE

- Clean (ie: short)
    - Does *one thing*
- Doesn't depend on context
    - Call from anywhere
    - Few object dependancies (if any)
        - Dependency inject if there is one (more later)
- Composable

GOURMET SERVICES

- Has only one method (`::call`)
  - De facto standard because same as Procs, methods, etc
- Takes `args={}`, or named params
- Dependancy inject all of the things!
  - i.e.: when calling outside objects, give the user the option to override with a call to a different object of the same duck type

SERVICES DIRECTORY

- It's even built into Rails!
  - app/services is automagically included
    - Models live in app/models
    - Services live in app/services
    - No need for Persistence namespace (shouldn't matter where the data comes from)
- Separates services from models (easier to read)

SERVICES LAYER

- A glance at the services directory shows all the things the app *does*
- Contextual, semantic, easy to know what it *does* from the outside
- `ScheduleAction`
- `BlockAccount`
- `SendReminder`
- `LaunchMissiles`

EXAMPLES

EXAMPLE

```ruby
# app/services/accept_invite.rb
class AcceptInvite
  def self.call(args = {})
    invite = args.fetch(:invite)
    user   = args.fetch(:user)

    invite.call(user)
    UserMailer.invite_accepted(invite).deliver
  end
end

# Somewhere else
AcceptInvite.call(user: chuck_norris, invite: party_time)
```

CONTRIVED SUPERBOLT EXAMPLE

```ruby
class Email::SendAdminSample
  def self.call(params = {})
    fail unless params[:emails].present?

    # Delegate to Advocato
    delegator = params.delete(:delegator) || Superbolt::Advocato::Enqueue
    delegator.call(params)
  end
end
```

COMPOSITION

- “has-a” rather than “is-a”
- Single responsibility (by design)
- Open/closed (from the perspective of a composed object)
- Composition moves towards concretion
  - Doesn’t inherit or depend on unused methods
  - Create hierarchies on the fly

KNOCK-ON EFFECTS

- Easy to name, because it does **one thing**
  - Adds context to code
    - via a semantic space of names and convention
- Succinct and clear
- Highly reusable (DRY)

EASY TO TEST

- Does **one** thing
- Few dependancies (by design)
- Isolated unit tests stub any calls to outside objects
- SOLID decoupling by design

USES

- Encapsulate asynchronicity
  - Workers
  - Superbolt queueing/fetching
- Hold common regexes (can compose them too)
- On the more radical end, can move all verbs into services to create a "behaviour layer"
- And more!

FURTHER READING

- Gourmet
  - http://brewhouse.io/blog/2014/04/30/gourmet-service-objects.html
  - https://gist.github.com/pcreux/9277929
  - http://www.reddit.com/r/rails/comments/24n5s2/gourmet_service_objects/
- Regular
  - http://blog.codeclimate.com/blog/2012/10/17/7-ways-to-decompose-fat-activerecord-models/
  - http://stevelorek.com/service-objects.html
  - http://jamesgolick.com/2010/3/14/crazy-heretical-and-awesome-the-way-i-write-rails-apps.html