
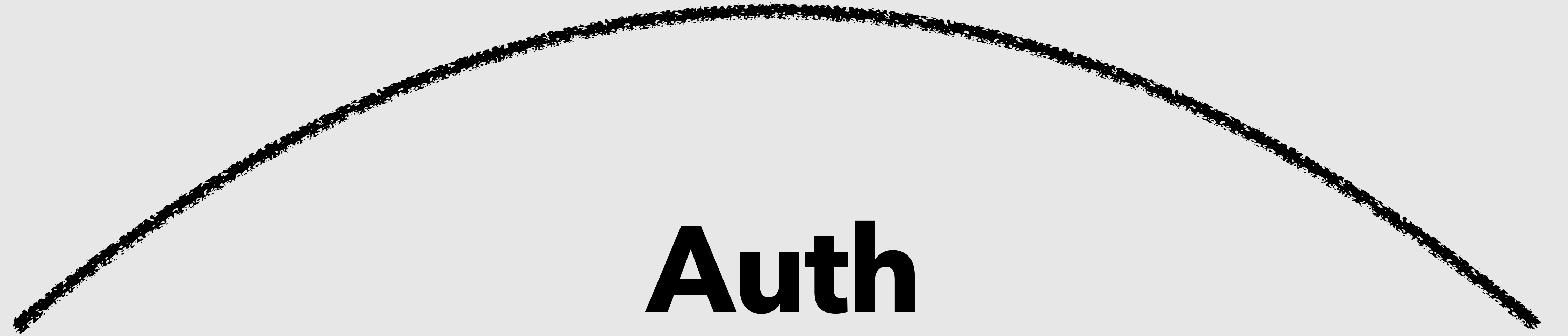


The Boring Bits Bite Back

Katie Miller, she/her, @phedinkus 

Helvetic Ruby 2024

Authorization is important.



Auth

Authentication

Authorization



Authentication

⇒ keys to the castle

Authorization

⇒ who can access certain areas

Authorization is what I do.

Authorization is boring.

Authorization is boring.

As it should be.

Authorization gets
neglected.

App complexity
grows over time.

Complexity in
Authorization
gets bitey.

Think about
Authorization
early on.

Let's build an app.

 **PawTracks** 

Manage and track
your pet's care.

~/codez

♥ 09:45

rails new paw_tracks

F.F.WD



Feature Request:

Allow multiple
people to care
for a pet.

```
class Account  
    has_many :users
```

```
class User  
    belongs_to :account
```

Feature: Allow multiple people to care for a pet.

Feature Refinement:

Some caretakers take
care of many pets.

Dog walkers, pet boarding, etc.


```
class Account  
    has_many :users
```

```
class User  
    belongs_to :account
```

```
class Account  
  has_many :users
```

```
class User  
  belongs_to :account
```

```
class Account  
    has_many :users
```

```
class User  
    has_many :accounts
```

Feature: Allow multiple people to care for a pet.

Feature Refinement:

One person is the
primary caretaker.

Authorization feature!

Can pet's caretakers all do
the same things?

Check users' roles.

user.owner? =>

account.owner_id == *user.id*

user.caretaker? =>

account

▪ *caretaker_ids*

▪ include? (*user.id*)

```
if user.owner?  
  # update pet details  
else  
  render :unauthorized  
end
```

Feature finished.

Ship it.

F.F.WD



Feature Request:

Allow support to
impersonate
customers.

K. I. S. S.

(Keep Is Simple Stupid)

user.admin? =>

user

.email

.match?(

/\@paw_tracks\.com/

)

Let the admins see
what the users see.

```
# Account access check  
account.includes?(user) ||  
user.admin?
```



F.F.WD



PawTracks is growing!



Project:
Vet Office
Integration

Vet and staff can
manage pet
health records in
PawTracks.

Add more roles.

user.veterinarian? =>

user.vet_office&.owner?

user.veterinarian_assistant? =>

user.vet_office&.assistant?

Check the new roles.

```
# PetMedicationsController#update  
if user.owner? ||  
  user.veterinarian?  
  # update pet medication  
elsif user.caretaker? ||  
  user.assistant? ||  
  user.admin?  
  render :unauthorized  
end
```

user.veterinarian?

user.assistant?

user.owner?

user.caretaker?

user.admin?

Feature Request:

Prevent updating Pet
Medication if the
account is cancelled.

*# PetMedicationsController#update
if account.cancelled?
render :unauthorized*

Lots of
if ... elif ...
Statements.

F.F.WD



Growth



Complexity



Time to tackle Tech Debt.

Policy Authorization Pattern

Can a **User**
perform an
Action on a
Resource?

user.can? (:update, medication)

Why the Policy Pattern?

No external dependencies.

Why the Policy Pattern?

Copy 

access checks

from controllers.

```
# PetMedicationPolicy  
def update?  
  user.owner? ||  
  user.veterinarian?  
end
```

Cleanup idea! 💡

```
# PetMedicationPolicy  
def update?  
  account.cancelled? ||  
  user.owner? ||  
  user.veterinarian?  
end
```

```
# PetMedicationsController#update  
if user.can?(:update?, medication)  
  # update pet medication  
else  
  render :unauthorized  
end
```




Gone bunches of
if ... elsif ...
statements!

+189 -2860



Sigh of relief.

Full speed ahead!



F.F.WD



Project:

Corporate

Vet Integration



Enterprise



Add more roles.

Customer roles

user.billing_admin?

user.business_analyst?

Internal roles

user.sales?

user.senior_admin?

Add new roles to
policy action methods.

lol



Winter 12:28 PM

Hey kt, we have a really big deal in the works. They have a feature request.

Their head of security said

We want to allow certain people to do that one thing owners can do but we don't want to make them owners

They asked us if we have RBAC (Role Based Access Control)



kt 12:29 PM

thinks good thing we refactored our auth recently

Yeah, we have RBAC. We can make new roles for them. What are we missing permissions-wise? (edited)



Winter 12:29 PM

<describes 2 new roles and fundamental changes to 3 roles>



kt 12:29 PM

Oh, that's a lot.

and I'm not sure changing the existing roles is a great idea.



Winter 12:29 PM



kt 12:30 PM

It sounds like they want to create their own roles

What is the problem
they are trying to solve?

Member Coordinator Role

Account

- create
- read
- update
- delete

Member

- create
- read
- update
- delete

Pet

- create
- read
- update
- delete

What can an **owner** do?

What can a **caretaker** do?

What can _____ do?

How will we do this?

Can we support
custom roles in
policies?

Not just checking
roles in the policy
methods.

Policy **Action** method
names are very
interesting.

User::AccountPolicy

transfer_pet_to_new_vet_office?

destroy_invites?

mark_pets_favorite_food?



kt 12:54 PM

I don't think we can do this with our current structure.

I think we need to re-write our Authorization system



Winter 12:55 PM

How long will that take?



kt 12:55 PM

I'd give the project an XL T-shirt size. 2-4 months maybe??



Winter 12:59 PM



Authorization wasn't a
requested feature...
until **Enterprise**.

Startup Law:

Thou shalt not
prematurely optimize

People use shortcuts to
avoid thinking about the
Authorization Architecture.

Startup Law:

Change is the only constant.

How could this
have been done
differently?

REWIND



Proposal 1:
Leverage
relational data
structure

```
def caretaker?  
  role == "Caretaker"  
end
```


**Special conditional for
different types of roles.**

Hard to remember how
users get a role.

Role names will change.

**Role permissions will
change.**

Roles will come and go.

```
class Member  
    belongs_to :account  
    belongs_to :user  
    belongs_to :role
```


Easily update role names.

Easily add & retire roles.

Easily see role assignments.

One day Roles can
be created by the
customer.

Proposal 2:

Don't use the

Policy

Authorization

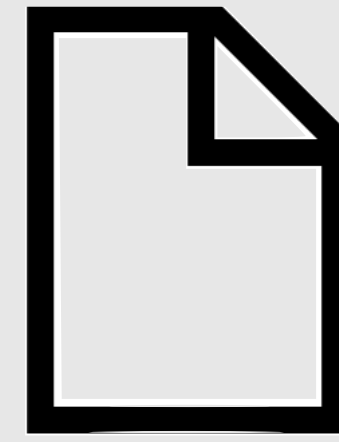
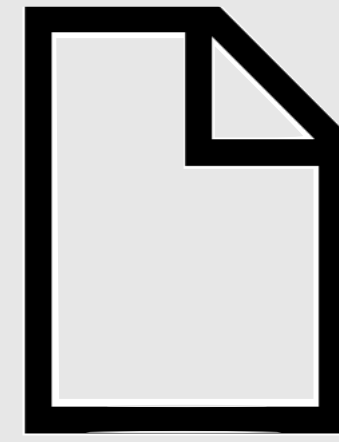
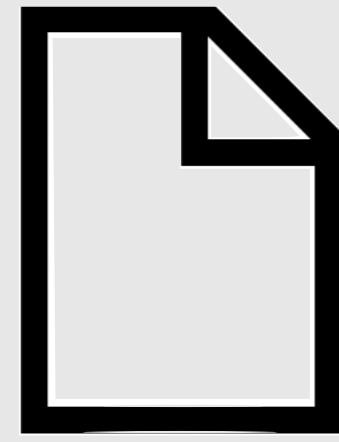
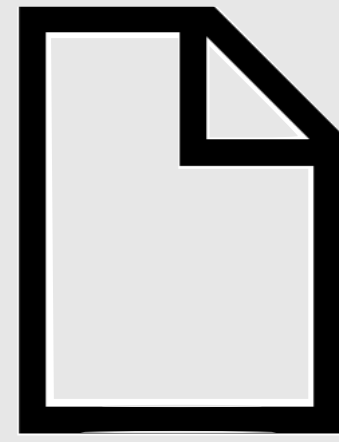
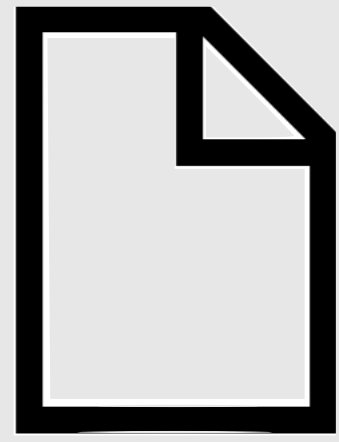
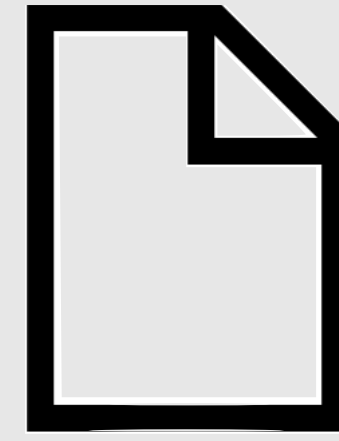
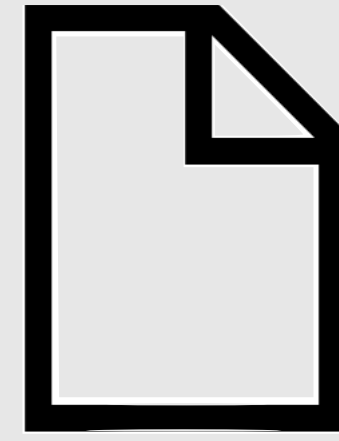
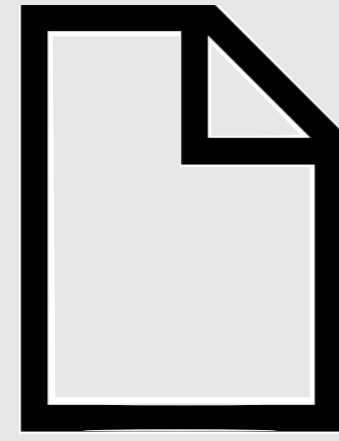
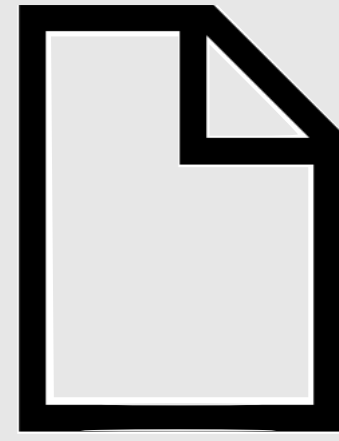
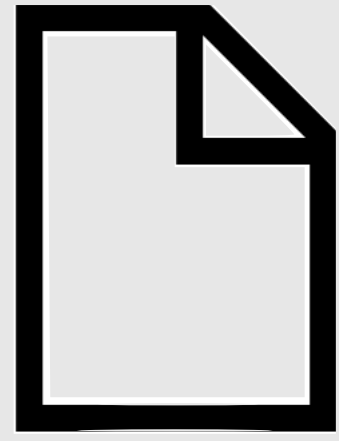
Pattern.

Policy action
methods have no
constraints.

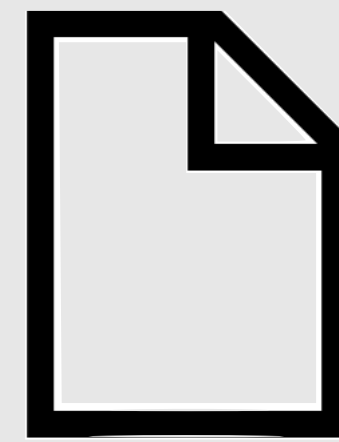
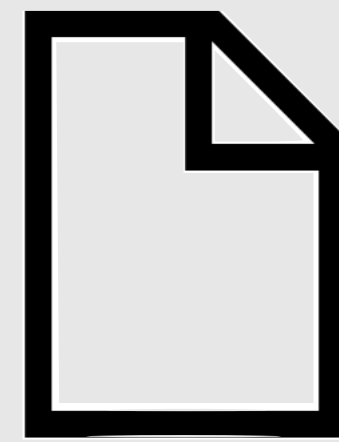
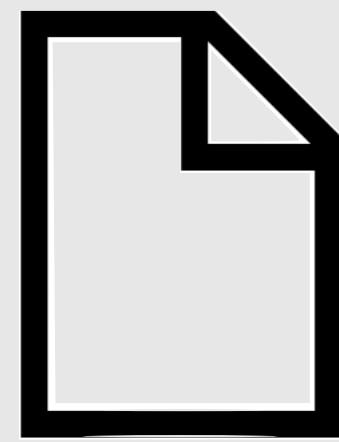
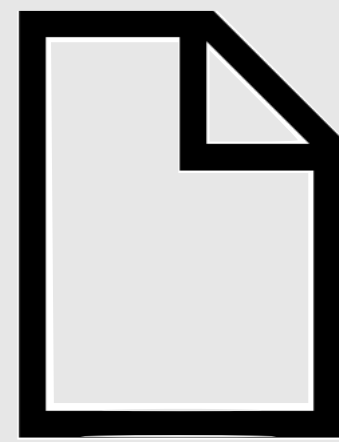
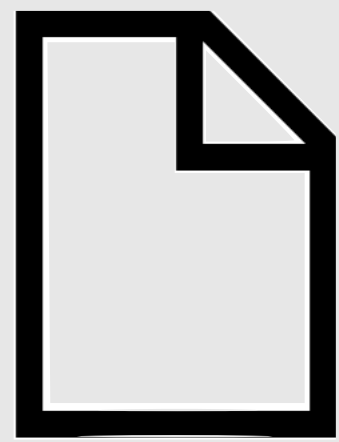
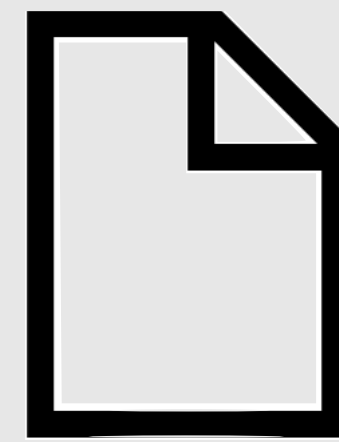
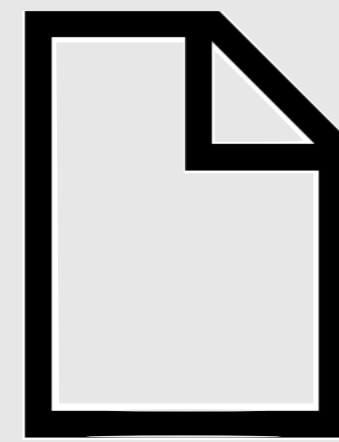
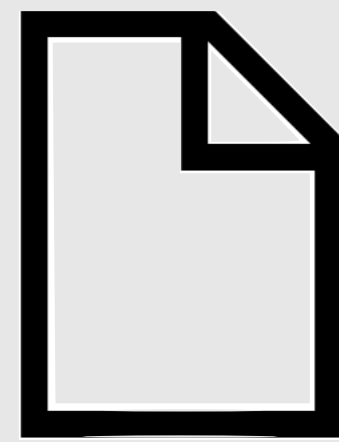
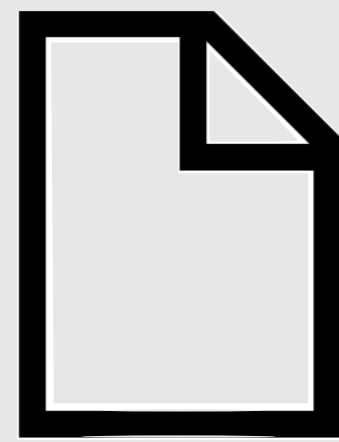
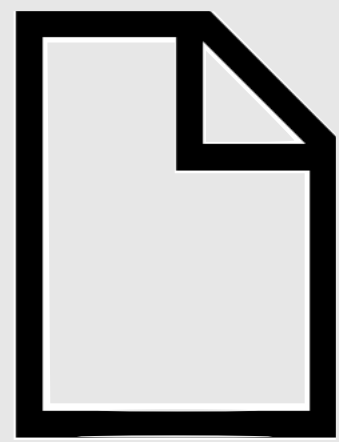
Can a **User**
perform an
Action on a
Resource?

Mixed

🚫 permission and state 🚫
checks.



Permissions not centralized.



Need a centralized, clear
way to declare
permissions per role and
resource.

Look around to see
what others are doing.

 **Gems can help.** 

CanCanCan



if user.owner?

can :manage, *Pet*

can :manage, *Account*

if user.caretaker?

can :read, *Pet*

can :update, *Pet*

```
if user.admin?  
  can :read, Pet  
  can :read, Account
```


The Role
defines the
capabilities.

Proposal 3:

Try to follow
CRUD naming.

(Create Read Update Delete)

Enforce good naming habits with:

Linters

Documentation & Education

Code Owners



- 1. Leverage Relational Data*
- 2. No to Policies*
- 3. Stick to CRUD*

Don't innovate in the
Authorization space.

You could be
building fun
features.

Give Authorization some thought.

The sooner the better.

Avoid big rewrites.

Don't let the
boring bits
bite back.

Thank you!

Katie Miller, @phedinkus 