Reform

•••

Piotr Misiurek (piotr@ragnarson.com)

RubyZG

Problem

Mess with validations in our data models

Problem

```ruby
class FooBar < ActiveRecord::Base
  [...]
  validates :email, :password, presence: true, on: :create
  validates :first_name, :last_name, :phone_number, presence: true, on: :update, if: :by_regular_user?
  validates :last_ticket_id, :support_person_id, presence: true, if: :by_support_team?
  validates :invoicing_schema, present: true, if: :paid_account?
  validates :tracking_number, format: { with: FooService::TRACKING_FORMAT }, if: :from_foo_service?
  validates :tracking_number, format: { with: BarService::TRACKING_FORMAT }, if: :from_bar_service?

  validate :account_plan, if: :by_manager?
  [...]
end
```

Piotr Misiurek

Ruby developer

Digital nomad

Blockchain enthusiast

Ragnarson

Agency

RoR specialization

Over 12 years of experience

Remote first

Teal organization

...

Teal organization

Frederic Laloux - Reinventing Organizations

Flat hierarchy

Self-management

Respect

Self-set salaries

Self-set salaries

Problem

```ruby
class FooBar < ActiveRecord::Base
  [...]
  validates :email, :password, presence: true, on: :create
  validates :first_name, :last_name, :phone_number, presence: true, on: :update, if: :by_regular_user?
  validates :last_ticket_id, :support_person_id, presence: true, if: :by_support_team?
  validates :invoicing_schema, present: true, if: :paid_account?
  validates :tracking_number, format: { with: FooService::TRACKING_FORMAT }, if: :from_foo_service?
  validates :tracking_number, format: { with: BarService::TRACKING_FORMAT }, if: :from_bar_service?

  validate :account_plan, if: :by_manager?
  [...]
end
```

Reform

Reform

THE FORM OBJECT

Form Object

Is like Service Object but for forms ;)

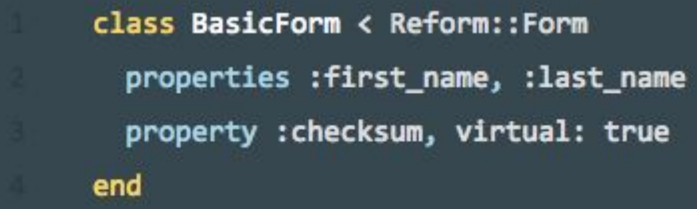
Why do we need form objects?

- Different contexts
- One form can handle fields from many data models
- Easy to change
- Easy to test

Basic form

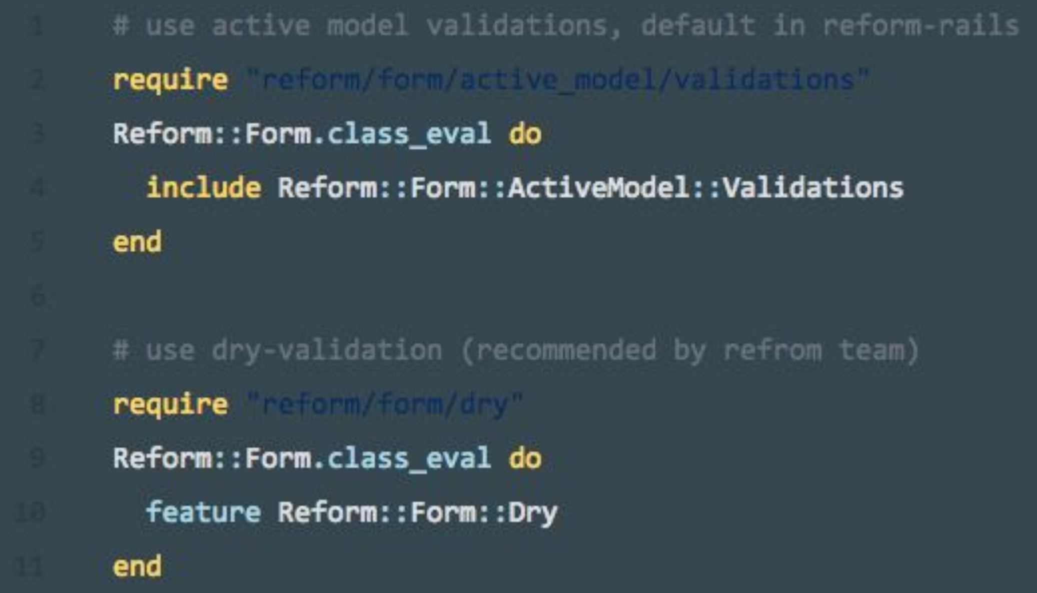
```ruby
class BasicForm < Reform::Form
  property :first_name
  property :last_name
  property :checksum, virtual: true
end
```

Basic form



```ruby
class BasicForm < Reform::Form
  properties :first_name, :last_name
  property :checksum, virtual: true
end
```

Configure validations

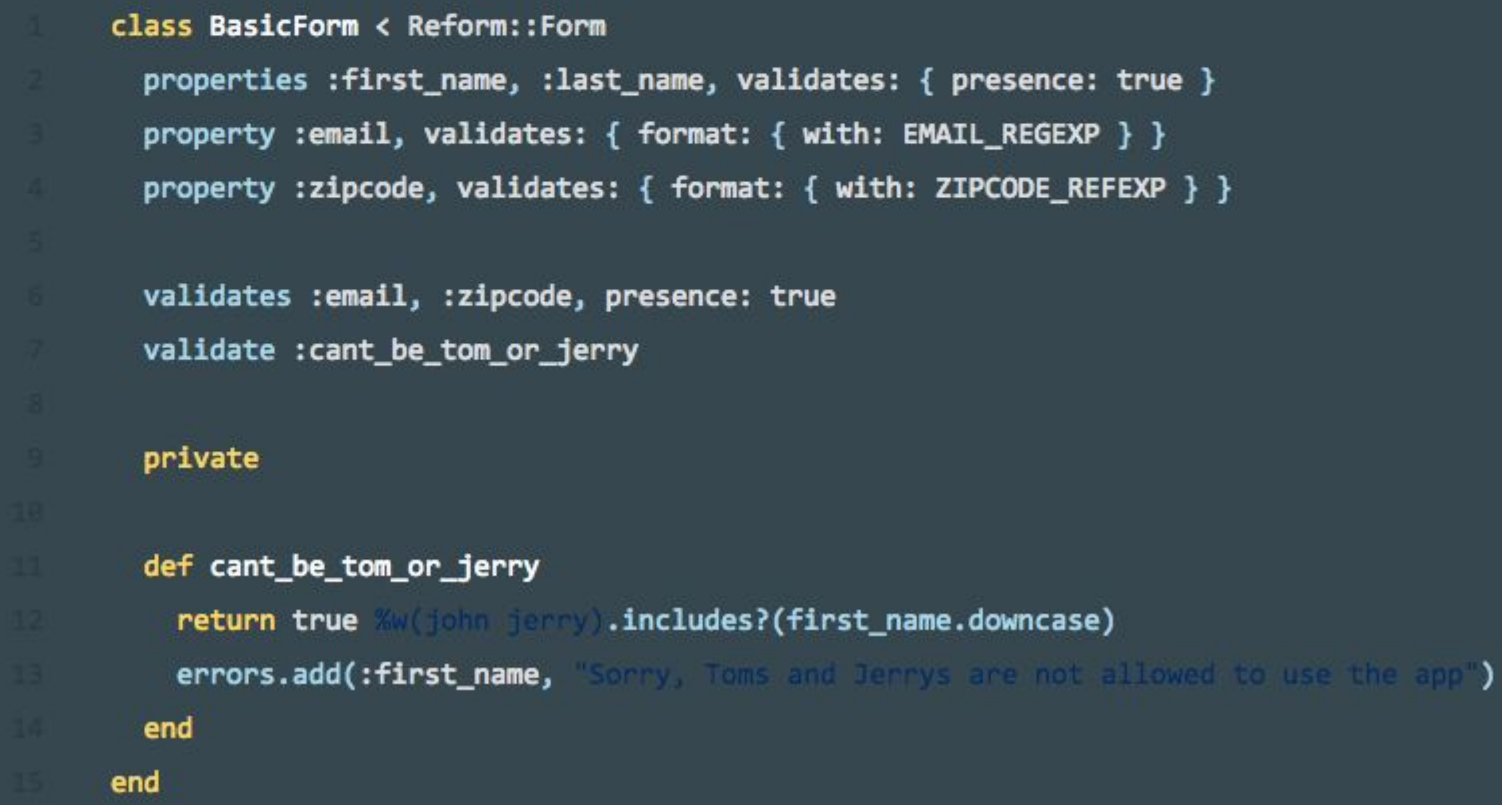


```ruby
# use active model validations, default in reform-rails
require "reform/form/active_model/validations"
Reform::Form.class_eval do
  include Reform::Form::ActiveModel::Validations
end

# use dry-validation (recommended by refrom team)
require "reform/form/dry"
Reform::Form.class_eval do
  feature Reform::Form::Dry
end
```

Validations

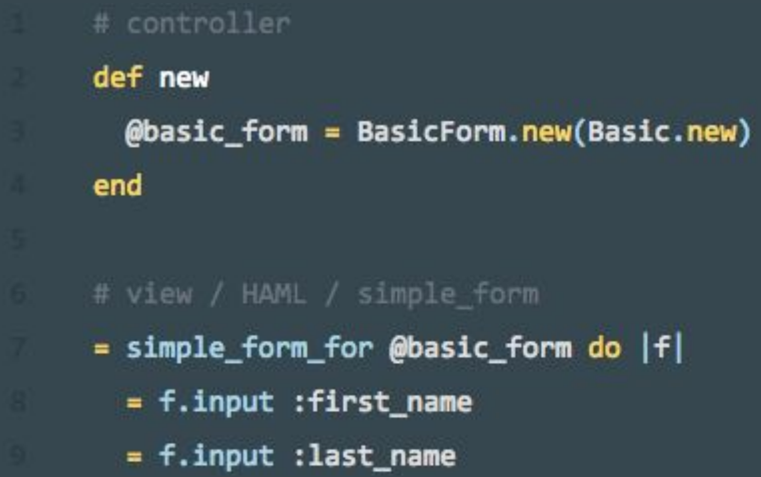
```ruby
class BasicForm < Reform::Form
  properties :first_name, :last_name, validates: { presence: true }
  property :email, validates: { format: { with: EMAIL_REGEXP } }
  property :zipcode, validates: { format: { with: ZIPCODE_REFEXP } }

  validates :email, :zipcode, presence: true
end
```

Custom validation



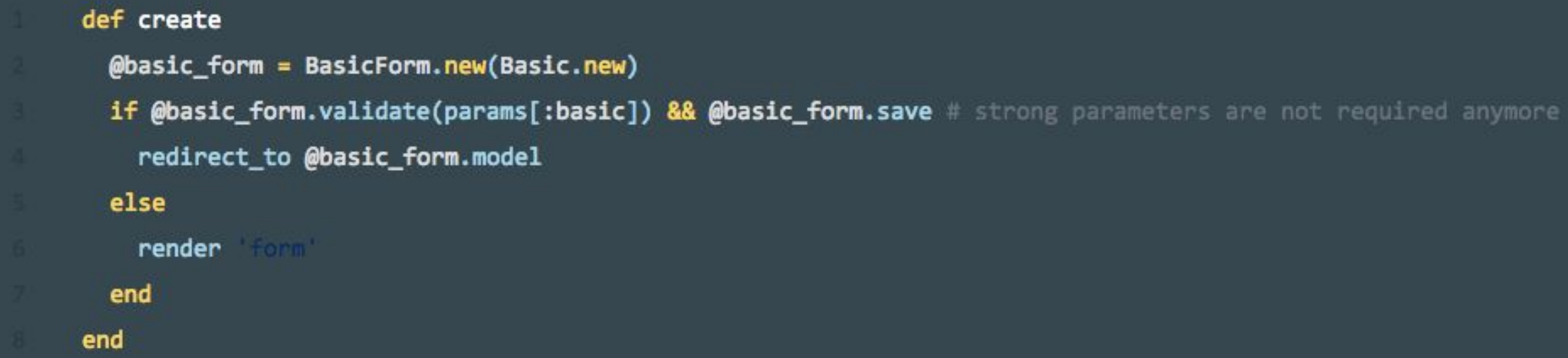
```ruby
class BasicForm < Reform::Form
  properties :first_name, :last_name, validates: { presence: true }
  property :email, validates: { format: { with: EMAIL_REGEXP } }
  property :zipcode, validates: { format: { with: ZIPCODE_REFEXP } }

  validates :email, :zipcode, presence: true
  validate :cant_be_tom_or_jerry

  private

  def cant_be_tom_or_jerry
    return true %w(john jerry).includes?(first_name.downcase)
    errors.add(:first_name, "Sorry, Toms and Jerrys are not allowed to use the app")
  end
end
```

Controller & View



```
# controller
def new
  @basic_form = BasicForm.new(Basic.new)
end

# view / HAML / simple_form
= simple_form_for @basic_form do |f|
  = f.input :first_name
  = f.input :last_name
```

Save in controller

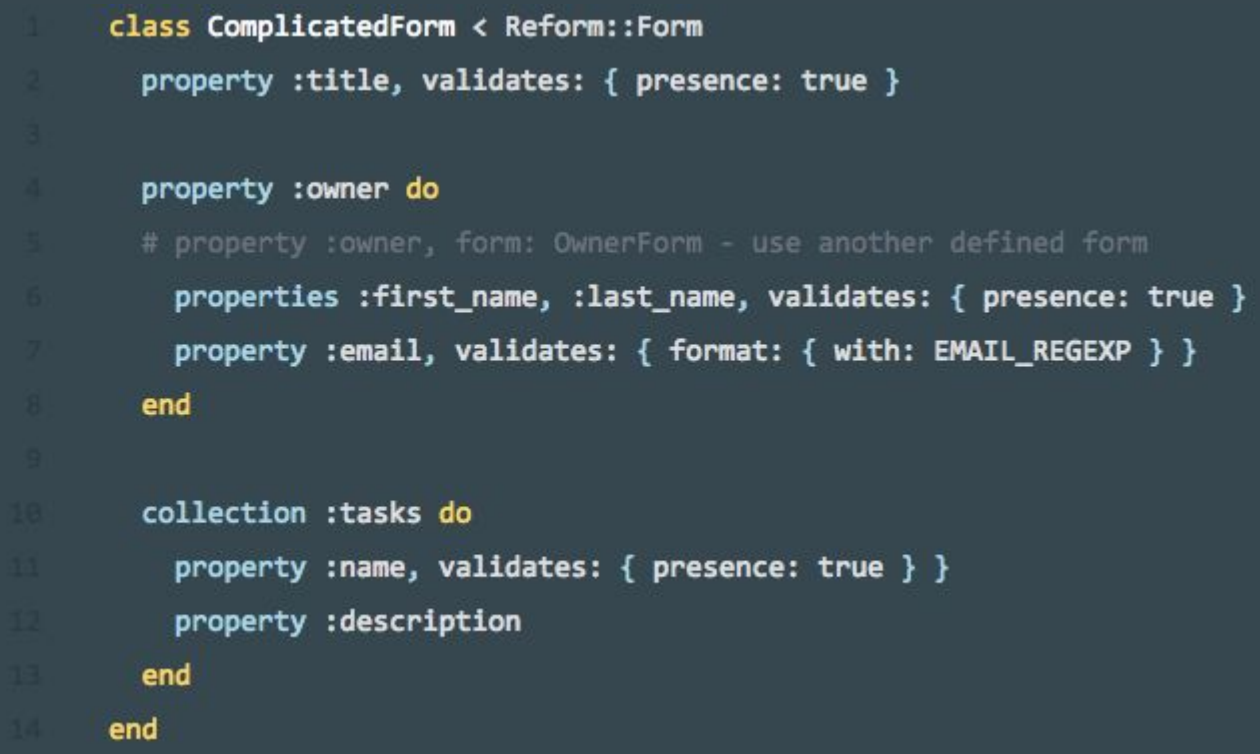


```ruby
def create
  @basic_form = BasicForm.new(Basic.new)
  if @basic_form.validate(params[:basic]) && @basic_form.save # strong parameters are not required anymore
    redirect_to @basic_form.model
  else
    render 'form'
  end
end
```

Reform flow

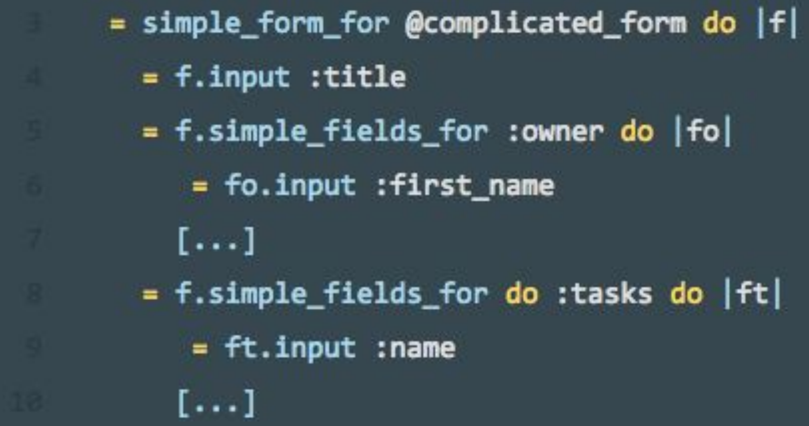
- initialize(model)
- validate(params)
- sync
- save

Nested forms



```ruby
class ComplicatedForm < Reform::Form
  property :title, validates: { presence: true }

  property :owner do
  # property :owner, form: OwnerForm - use another defined form
    properties :first_name, :last_name, validates: { presence: true }
    property :email, validates: { format: { with: EMAIL_REGEXP } }
  end

  collection :tasks do
    property :name, validates: { presence: true } }
    property :description
  end
end
```

Nested forms - view



```
= simple_form_for @complicated_form do |f|
  = f.input :title
  = f.simple_fields_for :owner do |fo|
     = fo.input :first_name
    [...]
  = f.simple_fields_for do :tasks do |ft|
     = ft.input :name
    [...]
```

Nested forms - controller

```ruby
def new
  @complicated = Complicated.new
  @complicated.build_owner
  @complicated.tasks.build
  @complicated_form = ComplicatedForm.new(@complicated)
end
```

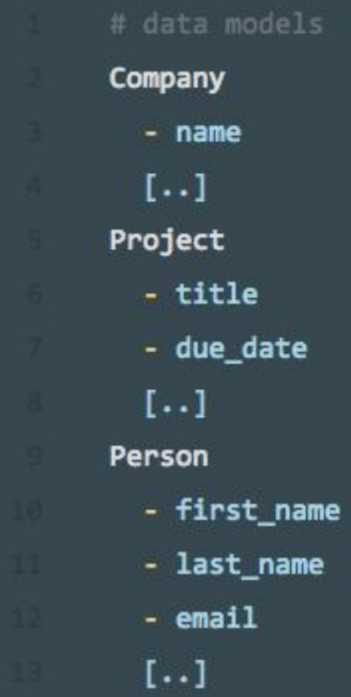
Prepopulators

```ruby
class ComplicatedForm < Reform::Form
  property :title, validates: { presence: true }

  property :owner, prepopulator: :build_owner do
    [...]
  end

  collection :task, prepopulator: :build_task do
    [...]
  end

  private

  def build_owner
    return if owner.present?
    owner = Owner.new
  end

  def build_tasks
    return if tasks.any?
    tasks < Task.new
  end
end
```

Populators

```ruby
class ComplicatedForm < Reform::Form
  property :title, validates: { presence: true }

  property :owner, prepopulator: :build_owner, populator: OwnerPopulator do
    [...]
  end

  collection :task, prepopulator: :build_task, populate_if_empty: Task do
    [...]
  end

  private

  def build_owner
    return if owner.present?
    owner = Owner.new
  end

  def build_tasks
    return if tasks.any?
    tasks < Task.new
  end
end
```

Composition - Data models

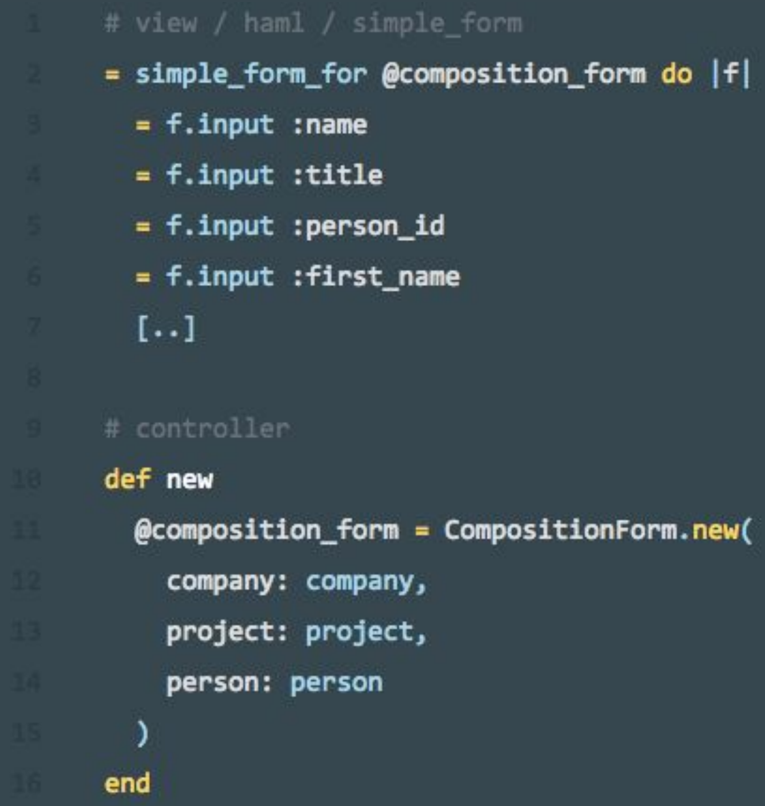


```
# data models
Company
  - name
  [..]
Project
  - title
  - due_date
  [..]
Person
  - first_name
  - last_name
  - email
  [..]
```

Composition - Form object

```ruby
class CompositionForm < Reform::Form
  include Composition

  property :company_id, on: :company, from: :id
  property :name, on: :company
  property :project_id, on: :project, from: :id
  property :title, on: :project
  property :due_date, on: :project
  property :person_id, :on: :person, from: :id
  property :first_name, on: :person
  property :last_name, on: :person
  property :email, on: :person
end
```

Composition - View & Controller



```
# view / haml / simple_form
= simple_form_for @composition_form do |f|
  = f.input :name
  = f.input :title
  = f.input :person_id
  = f.input :first_name
  [..]

# controller
def new
  @composition_form = CompositionForm.new(
    company: company,
    project: project,
    person: person
  )
end
```

Reform and form objects

- Easy to handle different contexts
- Model agnostic
- Easy to change
- Easy to test
- Easy to start

We are hiring!