Validate multiple contexts together in Rails 5

By Vijay Kumar Agrawal in Rails 5 *on April 08, 2016*

This blog is part of our Rails 5 series.

Active Record validation is a well-known and widely used functionality of Rails. Slightly lesser popular is Rails's ability to validate on custom context.

If used properly, contextual validations can result in much cleaner code. To understand validation context, we will take example of a form which is submitted in multiple steps:

```
class MultiStepForm < ActiveRecord::Base
  validate :personal_info
  validate :education, on: :create
  validate :work_experience, on: :update
  validate :final_step, on: :submission

  def personal_info
    # validation logic goes here..
  end

  # Smiliary all the validation methods go here.
end
```

Let's go through all the four validations one-by-one.

1. *personal_info* validation has no context defined (notice the absence of `on:`). Validations with no context are executed every time a model save is triggered. Please go through all the triggers here.

2. *education* validation has context of `:create`. It is executed *only* when a new object is created.

3. *work_experience* validation is in `:update` context and gets triggered for updates *only*. `:create` and `:update` are the only two pre-defined contexts.

4. *final_step* is validated using a custom context named `:submission`. Unlike above scenarios, it needs to be explicitly triggered like this:

```
form = MultiStepForm.new

# Either
form.valid?(:submission)

# Or
form.save(context: :submission)
```

`valid?` runs the validation in given context and populates `errors`. `save` would first call `valid?` in the given context and persist the changes if validations pass. Otherwise populates `errors`.

One thing to note here is that when we validate using an explicit context, Rails bypasses all other *contexts* including `:create` and `:update`.

Now that we understand validation context, we can switch our focus to *validate multiple context together* enhancement in Rails 5.

Let's change our contexts from above example to

```
class MultiStepForm < ActiveRecord::Base
  validate :personal_info, on: :personal_submission
  validate :education, on: :education_submission
  validate :work_experience, on: :work_ex_submission
  validate :final_step, on: :final_submission

  def personal_info
    # code goes here..
  end

  # Smiliary all the validation methods go here.
end
```

For each step, we would want to validate the model with all previous steps and avoid all future steps. Prior to Rails 5, this can be achieved like this:

```
class MultiStepForm < ActiveRecord::Base
  #...

  def save_personal_info
    self.save if self.valid?(:personal_submission)
  end

  def save_education
    self.save if self.valid?(:personal_submission)
              && self.valid?(:education_submission)
  end

  def save_work_experience
    self.save if self.valid?(:personal_submission)
              && self.valid?(:education_submission)
              && self.valid?(:work_ex_submission)
  end

  # And so on...
end
```

Notice that `valid?` takes only one context at a time. So we have to repeatedly call `valid?` for each context.

This gets simplified in Rails 5 by enhancing `valid?` and `invalid?` to accept an array. Our code changes to:

```ruby
class MultiStepForm < ActiveRecord::Base
  #...

  def save_personal_info
    self.save if self.valid?(:personal_submission)
  end

  def save_education
    self.save if self.valid?([:personal_submission,
                              :education_submission])
  end

  def save_work_experience
    self.save if self.valid?([:personal_submission,
                              :education_submission,
                              :work_ex_submission])
  end
end
```

A tad bit cleaner I would say.

By Vijay Kumar Agrawal In Rails 5
on April 08, 2016