Model-View-Controller
MVC

Web Programming Paradigm

Model

1. Classes
    1. Objects/Data
    2. Business Logic

2. Database
    1. Interacts with Database

Keyword: Data

43 lines (38 sloc) | 1.21 KB

Raw Bl

```ruby
class Wiki < ActiveRecord::Base
  belongs_to :user
  has_many :users, through: :collaborators
  has_many :collaborators

  validates :title, length: { minimum: 5, maximum: 255 }, presence: true
  validates :body, length: { minimum: 20 }, presence: true
  validates :user, presence: true

  extend FriendlyId
  friendly_id :title, use: [:slugged, :history]

  def user_collaborators
    users = []
    collaborators = Collaborator.includes(:user).where(wiki_id: self).all
    collaborators.each do |collaborator|
      users.push(collaborator.user)
    end
    users
  end

  def self.visible_to(user) # rubocop:disable Metrics/AbcSize
    wikis = []
    if user
      if user.role == 'admin'
        wikis = Wiki.all
      else
        wikis = Wiki.where('user_id=? OR private=?', user.id, false)
        collaborators = Collaborator.includes(:wiki).where(user_id: user).all
        collaborators.each do |collaborator|
          unless wikis.include?(collaborator.wiki) # rubocop:disable Style/IfUnlessModifier
            wikis.push(collaborator.wiki)
          end
        end
      end
    else
      wikis = Wiki.where(private: false)
    end
    wikis
    # wikis.uniq # uniq method removes private wiki when user is standard
  end
```

View

1. Web Page
    1. HTML/CSS
    2. Embedded JavaScript

2. User Interaction
    1. Clicking Links
    2. Data Entry

    Keyword: Presentation

12 lines (9 sloc) | 263 Bytes

```
.row
  .col-md-4
    %p Guidelines for wikis
    %ul
      %li Titles have to be at least 5 characters
      %li Bodys have to be big, at least 20 characters
      %li Nothing mean
  .col-md-8
    = render partial: 'wikis/form_new', locals: { wiki: @wiki }


```

14 lines (14 sloc) | 510 Bytes

```
= form_for wiki do |f|
  .form-group
    = f.label :title
    = f.text_field :title, class: 'form-control', placeholder: 'Enter title'
  .form-group
    = f.label :body
    = f.text_area :body, rows: 8, class: 'form-control', placeholder: 'Enter your wiki post'
  - if current_user.role == 'admin' || (current_user.role == 'premium')
    .form-group
      = f.label :private, class: 'checkbox' do
        = f.check_box :private
        Private wiki
  .form-group
    = f.submit 'Save', class: 'btn btn-success'
```

Controller

1. Routes and processes
    1. Requests
    2. Responses

2. Takes/sends data from/to view
    1. CRUD operations/HTTP Verbs

Keyword: Routing

```ruby
class WikisController < ApplicationController
  before_action :authenticate_user!, except: [:index, :show]

  def index
    @wikis = Wiki.visible_to(current_user)
    authorize @wikis
  end

  def show
    @wiki = Wiki.friendly.find(params[:id])
    unless request.path == wiki_path(@wiki)
      redirect_to @wiki, status: :moved_permanently
    end
    authorize @wiki
  end

  def new
    @wiki = Wiki.new
    authorize @wiki
  end

  def create
    @wiki = current_user.wikis.build(wiki_params)
    authorize @wiki
    if @wiki.save
      flash[:notice] = 'Your wiki was saved'
      redirect_to @wiki
    else
      flash[:error] = 'Your wiki failed to save'
      render :new
    end
  end

  def edit
    @wiki = Wiki.friendly.find(params[:id])
    authorize @wiki
  end

  def update
    @wiki = Wiki.friendly.find(params[:id])
    authorize @wiki
    @wiki.slug = nil
    if @wiki.update_attributes(wiki_params)
```

Separation of Concerns

1. Representation separate from data
    1. Custom Interfaces -> Views
    2. APIs -> Controllers
    3. Stateless|Asynchronous|RESTful

Supports Convention

Models

Style

Follow Django's defined conventions for model code.

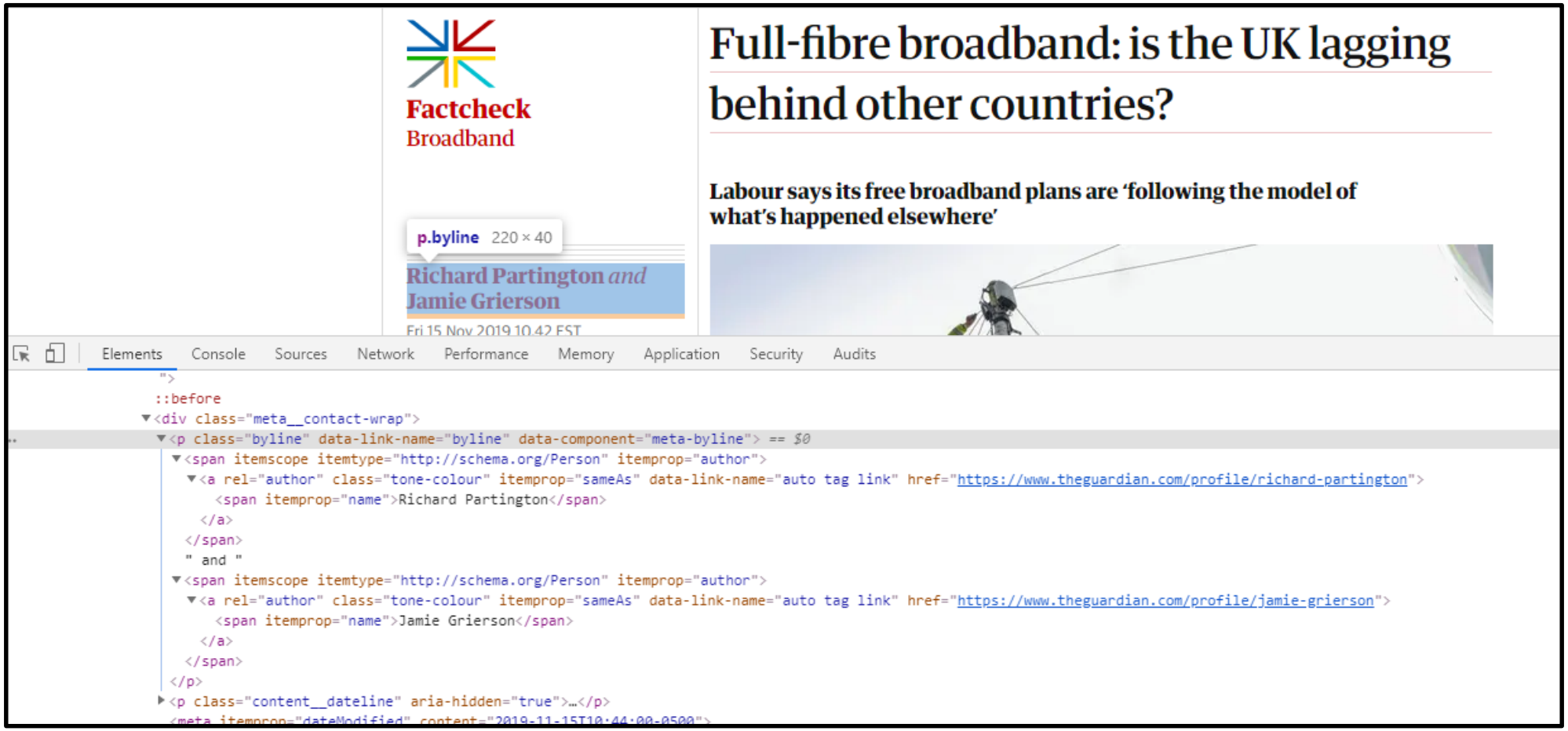
Make 'em Fat

A common pattern in MVC-style programming is to build thick/fat models and thin controllers. For Django this translates to building models with lots of small methods attached to them and views which use those methods to keep their logic as minimal as possible. There are lots of benefits to this approach.

1. **DRY**: Rather than repeating the same logic in multiple views, it is defined once on the model.
2. **Testable**: Breaking up logic into small methods on the model makes your code easier to unit test.
3. **Readable**: By giving your methods friendly names, you can abstract ugly logic into something that is easily readable and understandable.

For a good example of a fat model in Django, look at the definition of `django.contrib.auth.models.User`.

Representation



Data


root: {} 1 key
response: {} 4 keys
status: "ok"
userTier: "developer"
total: 1
content: {} 12 keys
id: "politics/2019/nov/15/labour-full-fibre-broadband-is-the-uk-lagging-behind-other-countries"
type: "article"
sectionId: "politics"
sectionName: "Politics"
webPublicationDate: "2019-11-15T15:42:26Z"
webTitle: "Full-fibre broadband: is the UK lagging behind other countries?"
webUrl: "https://www.theguardian.com/politics/2019/nov/15/labour-full-fibre-broadband-is-the-uk-lagging-behind-other-countries"
apiUrl: "https://content.guardianapis.com/politics/2019/nov/15/labour-full-fibre-broadband-is-the-uk-lagging-behind-other-countries"
tags: [] 2 items
0: {} 12 keys
id: "profile/richard-partington"
type: "contributor"
webTitle: "Richard Partington"
webUrl: "https://www.theguardian.com/profile/richard-partington"
apiUrl: "https://content.guardianapis.com/profile/richard-partington"
references: [] 0 items
bio: " Richard Partington is the Guardian's economics correspondent. Twitter @rjpartington "
bylineImageUrl: "https://uploads.guim.co.uk/2017/12/27/Richard-Partington.jpg"
bylineLargeImageUrl: "https://uploads.guim.co.uk/2017/12/27/Richard_Partington,_R.png"
firstName: "Richard"
lastName: "Partington"
twitterHandle: "RJPartington"
1: {} 12 keys
id: "profile/jamie-grierson"
type: "contributor"
webTitle: "Jamie Grierson"

Maintainable

1. Separate updates to separate layers
    1. Representation (new HTML/CSS: mobile compatible)
    2. Controller (Authentication methods)
    3. Models (Changing Data Requirements: soft deletes, archiving)

2. Specialized attention
    1. Web Design
    2. UX
    3. Programming

Testable

1. Representation/Features -> Clicks, URLs
2. Controllers -> [Mock] Requests
3. Models -> Data Retrieval, Storage, Creation

Extensible

Search : Representation → Data