iangreenleaf / **gist:b206d09c587e8fc6399e**

Rails naming conventions

gistfile1.md

Rails naming conventions

General Ruby conventions

Class names are `CamelCase`.

Methods and variables are `snake_case`.

Methods with a `?` suffix will return a boolean.

Methods with a `!` suffix mean one of two things: either the method operates destructively in some fashion, or it will raise and exception instead of failing (such as Rails models' `#save!` vs. `#save`).

In documentation, `::method_name` denotes a *class method*, while `#method_name` denotes a *instance method*.

Database

Database tables use `snake_case`. Table names are **plural**.

Column names in the database use `snake_case`, but are generally **singular**.

Example:

```
+--------------------------+
| bigfoot_sightings        |
+------------+-------------+
| id         | ID          |
| sighted_at | DATETIME    |
| location   | STRING      |
| profile_id | FOREIGN KEY |
+------------+-------------+

+------------------------------+
| profiles                     |
+---------------------+--------+
| id                  | ID     |
| name                | STRING |
| years_of_experience | INT    |
+---------------------+--------+
```

Model

Model *class names* use `CamelCase`. These are **singular**, and will map automatically to the plural database table name.

Model *attributes* and *methods* use `snake_case` and match the column names in the database.

Model files go in `app/models/#{singular_model_name}.rb`.

Example:

```ruby
# app/models/bigfoot_sighting.rb
class BigfootSighting < ActiveRecord::Base
  # This class will have these attributes: id, sighted_at, location
end
```

```ruby
# app/models/profile.rb
class Profile < ActiveRecord::Base
  # Methods follow the same conventions as attributes
  def veteran_hunter?
    years_of_experience > 2
  end
end
```

Relations in models

Relations use `snake_case` and follow the type of relation, so `has_one` and `belongs_to` are **singular** while `has_many` is **plural**.

Rails expects foreign keys in the database to have an `_id` suffix, and will map relations to those keys automatically if the names line up.

Example:

```
# app/models/bigfoot_sighting.rb
class BigfootSighting < ActiveRecord::Base
  # This knows to use the profile_id field in the database
  belongs_to :profile
end



# app/models/profile.rb
class Profile < ActiveRecord::Base
  # This knows to look at the BigfootSighting class and find the foreign key
  has_many :bigfoot_sightings
end
```

Controllers

Controller *class names* use `CamelCase` and have `Controller` as a suffix. The `Controller` suffix is always singular. The name of the resource is usually **plural**.

Controller *actions* use `snake_case` and usually match the standard route names Rails defines (`index` , `show` , `new` , `create` , `edit` , `update` , `delete`).

Controller files go in `app/controllers/#{resource_name}_controller.rb` .

Example:

```
# app/controllers/bigfoot_sightings_controller.rb
BigfootSightingsController < ApplicationController
  def index
    # ...
  end
  def show
    # ...
  end
```

```
  # etc
end


# app/controllers/profiles_controller.rb
ProfilesController < ApplicationController
  def show
    # ...
  end
  # etc
end
```

Routes

Route names are `snake_case`, and usually match the controller. Most of the time routes are **plural** and use the plural `resources`.

Singular routes are a special case. These use the singular `resource` and a singular resource name. However, they still map to a plural controller by default!

Example:

```
resources :bigfoot_sightings
# Users can only see their own profiles, so we'll use `/profile` instead
# of putting an id in the URL.
resource :profile
```

Views

View file names, by default, match the controller and action that they are tied to.

Views go in `app/views/#{resource_name}/#{action_name}.html.erb`.

Examples:

- `app/views/bigfoot_sightings/index.html.erb`
- `app/views/bigfoot_sightings/show.html.erb`
- `app/views/profile/show.html.erb`

More resources