Routes

1. ../rails/info/routes

2. rails routes

Routes and paths

1. <%= link_to('Styleguide', styles_atoms_path) %>

2. get 'styleguide', to: 'styles#atoms'
   get 'search_results', to: 'ideas#index'

3. root to: 'home#index'

List information for models

1. Model.inspect
2. db/schema.rb
3. Model.column_names

4. Model.columns_hash

5. Model.columns
6. For instance variables/attributes in an AR object

a. object.attribute_names

b. object.attribute_present?

c. object.attributes

10. For instance methods without inheritance from super class

a. Model.instance_methods(false)

Model methods

1. Model.all
2. Model.where(id: 1)
3. Model.where('title LIKE ?', search_term)
4. Model.where('title LIKE ?', '%#{search_term}%')

Ordering

1. Idea.all.order(created_at: :desc)
2. Idea.all.order(title: :asc)
3. Idea.all.order(done_count: :desc)

Maximum records

1. Idea.all.limit(5)

Combining

1. Idea.all.order(created_at: :desc).limit(3)
2. results = Idea.where('title LIKE ?', "%#{search_term}%").or(Idea.where('description LIKE ?', "%#{search_term}%"))

Methods that produce an array of results

1. .all
2. .find
3. .where
4. .order
5. .limit
6. .or

Creating associated models

1. rails generate model Comment body:string idea:references
2. class Comment < ApplicationRecord belongs_to :idea end
3. associated_idea = Idea.first comment = Comment.new(body: 'This is great!', idea: associated_idea) comment.save comment.idea.title => 'A weekend in Dublin'
4. class Idea < ApplicationRecord has_many :comments end
5. idea = Idea.first first_comment = Comment.new(body: 'This is great!') idea.comments << first_comment second_comment = Comment.new(body: "Yes, I'd love to do this!") idea.comments << second_comment idea.comments.count => 2 idea.comments.first.body => 'This is great!'

6. idea.comments.order(created_at: :desc) idea.comments.order(created_at: :desc).limit(3)

Interacting with an associated model

1. in router: resources :ideas
2. resources :ideas do
   resources :comments
   end

3. idea = Idea.find(23) Rails.application.routes.url_helper.idea_comments_path(idea) => "/idea/23/comments" Rails.application.routes.url_helper.new_idea_comment_path(23) => "/idea/23/comments/new" Rails.application.routes.url_helper.idea_comment_path(idea, 212) => "/idea/23/comments/212"
4. <%= form_for([@idea, @comment]) do |form| %> … <% end %>
5. class IdeasController < ApplicationRecord

def show @idea = Idea.find(params[:id]) @comment = Comment.new end

end 6. class CommentsController

def create @idea = Idea.find(params[:idea_id]) @comment = Comment.new(comment_params) @comment.idea = @idea @comment.save! redirect_to(idea_path(@idea)) end

end

Creating another associated model

1. rails generate model Comment body:string idea:references
2. idea:references
3. rails generate model User email:string
4. rails generate migration AddUserToComment user:references
5. rails db:migrate
6. class Comment < ApplicationRecord

belongs_to :idea
belongs_to :user

end
7. class User < ApplicationRecord
has_many :comments
end

A sign up and log in form

1. resources :users
   resources :ideas do resources :comments end

2. rails generate controller Users

Associations in more detail

1. `rails generate migration AddUserToComment user:references`
   What this actually does is to create a new column in the comments table called user_id.

Create the style guide and homepage

1. rails generate controller styles atoms molecules organisms

2. rails generate controller –help

Create some partials

1. <%= render(partial: 'idea_card') %>

2. or if in the same folder:
   <%= render(partial: 'header') %>

3. Partials are a small fragment of HTML in an ERB file, and can be referenced in views or layouts using <%= render(partial: 'partial_name') %>.

4. Partials must always be created with a leading underscore in the filename, and must always be created within a subfolder of the app/views/ folder.

5. Views can reference partials in different folders by including the folder name, <%= render(partial: 'folder_name/partial_name') %>.

6. Commonly used partials can be placed in the app/views/application/ folder, which means that the folder name can be omitted when referencing the partial within a view.

CSS, Sass and front end assets

1. The first thing that we need to do is replace the default app/assets/stylesheets/application.css file with a Sass equivalent, app/assets/stylesheets/application.scss

2. Beyond the application.scss file, the structure of Sass files within the app/assets/stylesheets folder is entirely at a developer's discretion.

3. include inside
   tag
   <%= stylesheet_link_tag 'application', media: 'all' %>

4. Images are put within the app/assets/images/

5. reference images from css file
   .some-class {
   background-image: url(../images/mountain-lake.jpg);
   }

6. or, using the asset pipeline
   .some-class {
   background-image: asset-url('mountain-lake.jpg');
   }

Dynamic elements in views

1. <%= 'Atoms' %>

2. <%= link_to('Atoms', '/styles/atoms') %>

3. <%= link_to('Text for anchor', '/relative/path') %>

Parameters for partials (use of locals)

1. <%= render(partial: 'button', locals: { label: 'Accept' }) %>

2. <%= render(partial: 'button', locals: { label: 'Go!',
   extra_css_class: 'action' }) %>

3. <%= button_tag(label, class: "button #{extra_css_class}") %>

Iterating in views

1. loops
   <% [1, 2, 3, 4].each do %>

   One step through the array.
   <% end %>

Form Helpers

0. https://guides.rubyonrails.org/form_helpers.html#helpers-for-generating-form-elements
1. https://api.rubyonrails.org/classes/ActionView/Helpers/FormTagHelper.html#method-i-text_field_tag
2. <%= form_tag(ideas_index_path, method: :get, class: 'form full') do %>
   <%= text_field_tag('q', nil, class: 'input') %>
   <%= button_tag('Search', class: 'button primary') %>
   <% end %>

3. <%= text_field_tag('q', nil, class: 'input') %>

4. <%= button_tag('Search', class: 'button primary') %>

   Search

5. <%= text_area_tag(:message, "Hi, nice site", size: "24x6") %>
   <%= password_field_tag(:password) %>
   <%= hidden_field_tag(:parent_id, "5") %>
   <%= search_field(:user, :name) %>
   <%= telephone_field(:user, :phone) %>
   <%= date_field(:user, :born_on) %>
   <%= datetime_local_field(:user, :graduation_day) %>
   <%= month_field(:user, :birthday_month) %>
   <%= week_field(:user, :birthday_week) %>
   <%= url_field(:user, :homepage) %>
   <%= email_field(:user, :address) %>
   <%= color_field(:user, :favorite_color) %>
   <%= time_field(:task, :started_at) %>
   <%= number_field(:product, :price, in: 1.0..20.0, step: 0.5) %>
   <%= range_field(:product, :discount, in: 1..100) %>
   <%= radio_button_tag(:age, "child") %>

<%= label_tag(:age_child, “I am younger than 21”) %>
<%= check_box_tag(:pet_dog) %>
<%= label_tag(:pet_dog, “I own a dog”) %>

Models

1. rails generate MyNewModel attribute1:string attribute2:integer

2. rails db:migrate

3. idea = Idea.new

idea.title = ‘Stand up paddle boarding in Annecy’
idea.done_count = 24
idea.photo_url = ‘http://exts.epfl.ch/images/sup.jpg’

idea.save!

all_ideas = Idea.all
all_ideas.first == idea

idea.destroy!
Idea.all.empty?

Managing models

1. `ideas_index GET  /ideas/index(.:format) ideas#index`

   new_idea_index GET /ideas/new ideas#new
   create_idea_index POST /ideas/create ideas#create

Extending an existing model

1. rails generate migration AddAttributeNameToModelName attribute_name:string

2.

adds the attribute ‘photo_url’, which is a String, to the Idea model

rails generate migration AddPhotoUrlToIdea photo_url:string

3.

adds the attribute price, which is a Decimal, to the Product model

rails generate migration AddPriceToProduct price:decimal

4.

adds nothing to the Address Model, as there is no attribute argument

rails generate migration AddPostCodeToAddress

5.

adds the attributes age and email to the User model

age is an Integer and email is a String
rails generate migration AddAgeToUser age:integer email:string

Reversing a migration

1. rails db:rollback
2. Delete the migration file that was created by the generate migration command

3. There is a second, more forceful way to reverse a migration, but this involves deleting all of the data in the existing database. To this, it is necessary to first delete the migration file that was just created, then also deleting the development database file db/development.sqlite3 and the test database file db/test.sqlite3. Then the database can be recreated using the command rails db:setup. Only do this if you are willing to delete the existing data in your databases.

Updating a record

1. class IdeasController < ApplicationController def update # extract the identifier from the params Hash id = params[:id]

```
# retrieve the instance using the identifier
idea = Idea.find(id)

idea.title = params[:title]
idea.save!
redirect_to(ideas_index_path)
```

end end

Automated testing

1. sleep(…)
2. visit(…)
3. fill_in(…)
4. click_on(…)
5. test 'capybara works' do visit('http://www.google.com') sleep(5.seconds) fill_in('q', with: 'Ruby on Rails') sleep(5.seconds) click_on('Search') click_on('Ruby on Rails', match: :first) assert page.has_content?('Contribute on GitHub') end
6. rails test:system
7. assert page.has_content?('Cycle across Australia')
8. assert_equal 2, page.all('').count

Building the show idea view

1. get 'ideas/:id', to: 'ideas#show', as: 'show_idea'
2. class IdeasController < ApplicationController

def show @idea = Idea.find(params[:id]) end

end 3. app/views/ideas/show.html.erb 4. rails generate system_test test_name

Search

1. search_term = ‘The complete title’ //exact match ideas = Idea.where(‘title LIKE ?’, search_term)
2. search_term = ‘match’ // all matches ideas = Idea.where(‘title LIKE ?’, “%#{search_term}%”)
3. class Idea < ApplicationRecord def self.search(search_term) where(‘title LIKE ?’, “%#{search_term}%”) end end
4. class IdeasController < ApplicationController

def index @search_term = params[:q] @ideas = Idea.search(@search_term) end

end

Results

1. Idea.all.order(created_at: :desc)
2. Idea.all.order(title: :asc)
3. Idea.all.order(done_count: :desc)
4. Idea.all.limit(5)
5. Idea.all.order(created_at: :desc).limit(3)

More complex search

1. search_term = ‘France’ Idea.where(‘title LIKE ?’, “%#{search_term}%”)
2. search_term = ‘France’ Idea.where(‘description LIKE ?’, “%#{search_term}%”)
3. search_term = ‘France’

results = Idea.where(‘title LIKE ?’, “%#{search_term}%”).or(Idea.where(‘description LIKE ?’, “%#{search_term}%”)) results.count 4. class Idea < ApplicationRecord def self.search(search_term) where(‘title LIKE ?’, “%#{search_term}%”) end end 5. class Idea < ApplicationRecord def self.search(search_term) where(‘title LIKE ?’, “%#{search_term}%”).or(where(‘description LIKE ?’, “%#{search_term}%”)) end end 6. .all 7. .find 8. .where 9. .order 10. .limit 11. .or

The database

1. hash = { title: ‘Island hop around Greece’, description: ‘A great way to experience Greek islands’, photo_url: ‘http://some.host/new_image.jpg’ } idea = Idea.new(hash) idea.title => ‘Island hop around Greece’ idea.save!
2. def update idea = Idea.find(params[:id]) idea.title = params[:title] idea.done_count = params[:done_count] idea.photo_url = params[:photo_url] idea.save! redirect_to account_ideas_path end
3. hash = { title: ‘Island hop around Greece - update’, description: ‘The best way to see Greek islands - update’, photo_url: ‘http://some.host/new_image.jpg’ } idea = Idea.last() idea.update(hash) idea.title => ‘Island hop around Greece - update’

Parameters in controllers

1. class IdeasController < ApplicationController def create end

def update end end 2.