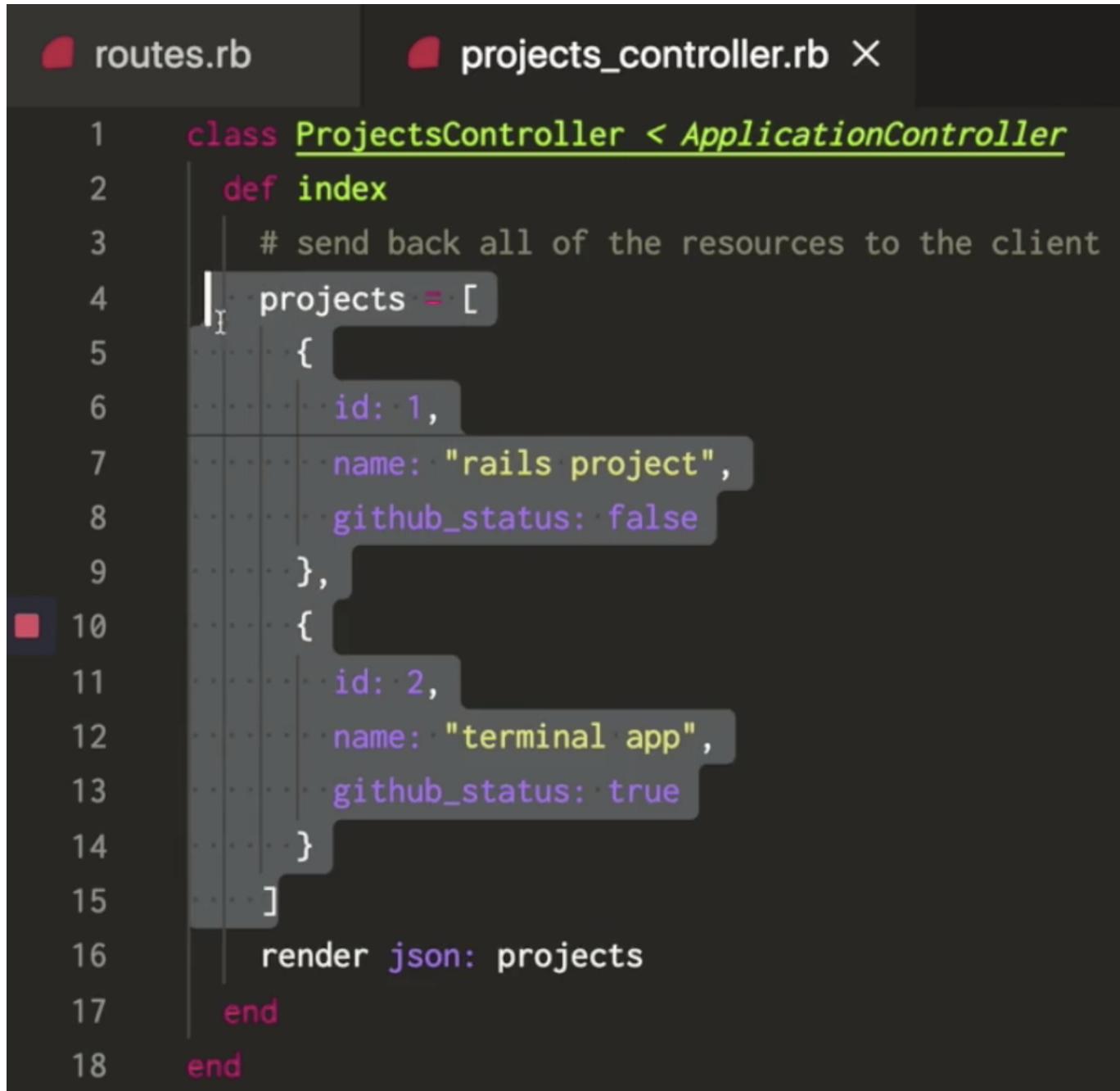


Rails Controllers - Show:

Link to lesson

- A link to the lesson can be found [here](#).
- We saw in the previous lesson that we were sending back all of our projects.
- Basically we were storing all of our projects in an array



The screenshot shows a code editor with two tabs: "routes.rb" and "projects_controller.rb". The "projects_controller.rb" tab is active and displays the following code:

```
1 class ProjectsController < ApplicationController
2   def index
3     # send back all of the resources to the client
4     projects = [
5       {
6         id: 1,
7         name: "rails project",
8         github_status: false
9       },
10      {
11        id: 2,
12        name: "terminal app",
13        github_status: true
14      }
15    ]
16    render json: projects
17  end
18end
```

The code defines a controller named "ProjectsController" that inherits from "ApplicationController". The "index" action sends back all of the resources to the client. It does this by creating an array called "projects" containing two project objects. Each project object has an "id", a "name", and a "github_status". The first project has an id of 1, a name of "rails project", and a false github status. The second project has an id of 2, a name of "terminal app", and a true github status. Finally, it renders a JSON response with the "projects" array.

- Then we were sending back a response to the client (see image below)

```
[  
]  
    render json: projects  
end  
end
```

- The response was formatted as JSON; meaning we were sending back a JSON data structure or a JSON format back to the client side.
- So when we typed into our URL

```
localhost: 3000/projects
```

- We saw our array of hashes

```
*[{"id":1,"name":"rails project","github_status":false}, {"id":2,"name":"terminal app","github_status":true}]
```

Sending back one project at a time (see image below)

```
[{"id":1,"name":"rails project","github_status":false}, {"id":2,"name":"terminal app","github_status":true}]
```

- The way in which we can achieve this is by:
 - Defining a new route and by having something particular in that route.

Route URL

- What we want the route to look like is something like this:

```
localhost: 3000/projects/(a number)
```

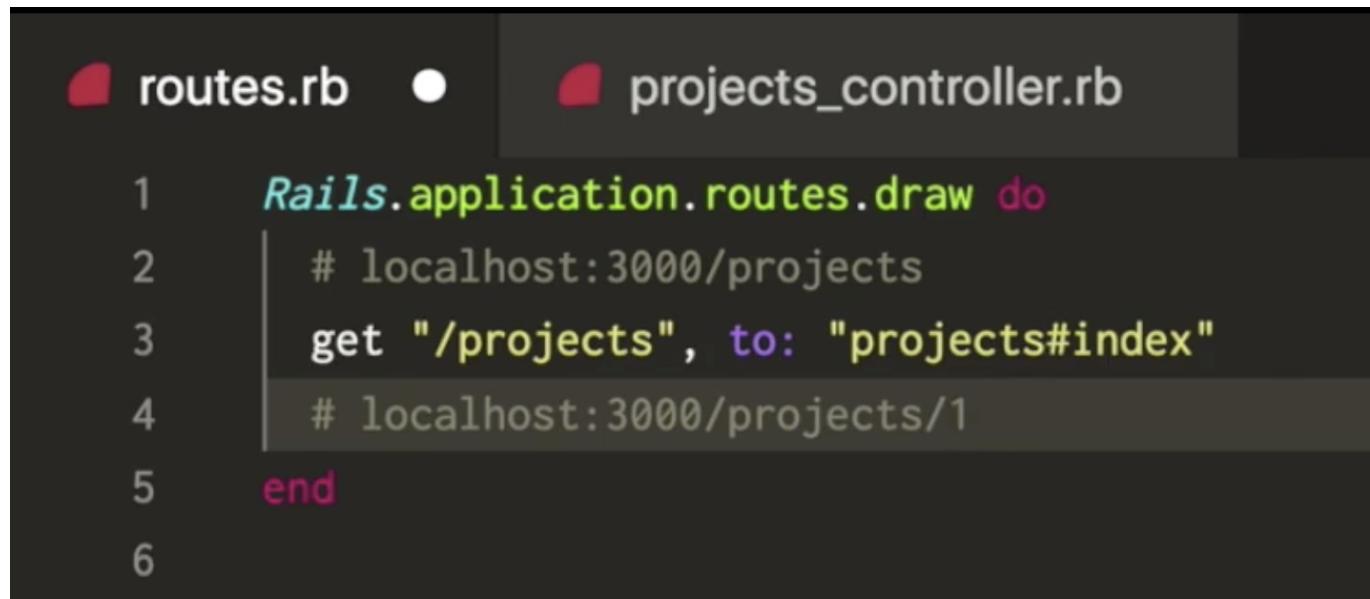
For example:

```
localhost: 3000/projects/1
```

OR

```
localhost: 3000/projects/2
```

ETC



The screenshot shows a terminal window with two files open: `routes.rb` and `projects_controller.rb`. The `routes.rb` file contains the following code:

```
1 Rails.application.routes.draw do
2   # localhost:3000/projects
3   get "/projects", to: "projects#index"
4   # localhost:3000/projects/1
5 end
6
```

The `get "/projects", to: "projects#index"` line is highlighted in yellow, and the `# localhost:3000/projects/1` line is also highlighted in yellow, indicating they are part of the same route definition.

- This number is going to correlate to an id (image below highlights id 1)

The screenshot shows a code editor with two tabs: 'routes.rb' and 'projects_controller.rb'. The 'projects_controller.rb' tab is active, displaying the following Ruby code:

```
1 class ProjectsController < ApplicationController
2   def index
3     # send back all of the resources to the client
4     projects = [
5       {
6         id: 1,
7         name: "rails project",
8         github_status: false
9       },
10      {
11        id: 2,
12        name: "terminal app",
13        github_status: true
14      }
15    ]
16    render json: projects
17  end
18end
```

- Therefore, if we type in the number 1 into our url:

```
localhost: 3000/projects/1
```

- We should be able to fetch our correcting id from our projects_controllers file (the particular hash in the image below):

```
routes.rb • projects_controller.rb X

1  ✓ class ProjectsController < ApplicationController
2  ✓   def index
3    # send back all of the resources to the client
4  ✓     projects = [
5  ✓       {
6        id: 1,
7        name: "rails project",
8        github_status: false
9      },
10  ✓   {
11        id: 2,
12        name: "terminal app",
13        github_status: true
14      }
15    ]
16    render json: projects
17  end
18 end
```

- The same goes if we were to type in the number 2 into our url:

```
localhost: 3000/projects/2
```

- We should be able to fetch the id 2 from our projects_controller file (see image below):

```
routes.rb • projects_controller.rb X

1  class ProjectsController < ApplicationController
2    def index
3      # send back all of the resources to the client
4      projects = [
5        {
6          id: 1,
7          name: "rails project",
8          github_status: false
9        },
10     {
11       id: 2,
12       name: "terminal app",
13       github_status: true
14     }
15   ]
16   render json: projects
17 end
18 end
```

Define the Route

- To define the route head to the routes.rb file
- This will use a get route (get request)
- Then we can add the name of the route.

```
get "/projects/"
```

- Then the number could be anything as it refers to the particular project we want to fetch.

Passing a placeholder

- This will represent the part of the route that can change.
- The notation for passing a placeholder is:
 - A colon and then the placeholder name for this particular route (id in this example)

```
get "/projects/:id"
```

The screenshot shows a code editor with two tabs: 'routes.rb' and 'projects_controller.rb'. The 'routes.rb' tab is active, displaying the following code:

```
1  Rails.application.routes.draw do
2      # localhost:3000/projects
3      get "/projects", to: "projects#index"
4      # localhost:3000/projects/5
5      get "/projects/:id"
6  end
```

The ':id' placeholder in the fifth line is highlighted with a blue rectangle. The 'projects_controller.rb' tab is visible in the background.

- id will represent some kind of number (could be 1 could be 2 could be 10 etc)
- Let's now define our key value pair
- This will go to the projects folder
- remembering that the first part of the to: is the name of the controller (projects in this example)
- The second part is the name of the controller action - separated with a #
- In this example we are using show as the controller action (a convention in Rails)

```
get "/projects/:id", to: "projects#show"
```

The screenshot shows a code editor with two tabs: "routes.rb" and "projects_controller.rb". The "routes.rb" tab is active, displaying the following code:

```
1  Rails.application.routes.draw do
2      # localhost:3000/projects
3      get "/projects", to: "projects#index"
4      # localhost:3000/projects/5
5      get "/projects/:id", to: "projects#show"
6  end
```

Show action

- The show action is for sending back only one particular resource (one project in this example)

Checking show action

- Let's type out our url (see images below) into our browser and see what we get.

The screenshot shows a code editor with two tabs: "routes.rb" and "projects_controller.rb". The "routes.rb" tab is active, displaying the following code:

```
1  Rails.application.routes.draw do
2      # localhost:3000/projects
3      get "/projects", to: "projects#index"
4      # localhost:3000/projects/1
5      get "/projects/:id", to: "projects#show"
6  end
```

- We are getting an error:

The screenshot shows a web browser window with two tabs: "Copy of CA Template - Google" and "Action Controller: Exception ca". The address bar indicates the URL is "localhost:3000/projects/1". The main content area has a red header with the text "Unknown action" and a red body with the text "The action 'show' could not be found for ProjectsController".

- The error is pretty clear and is an easy fix.
- First lets go to our projects_controller.rb file

The screenshot shows a code editor with two tabs: "routes.rb" and "projects_controller.rb". The "projects_controller.rb" tab is active and contains the following code:

```
1 class ProjectsController < ApplicationController
2   def index
3     # send back all of the resources to the client
4     projects = [
5       {
6         id: 1,
7         name: "rails project",
8         github_status: false
9       },
10      {
11        id: 2,
12        name: "terminal app",
13        github_status: true
14      }
15    ]
16    render json: projects
17  end
18
19
20 end
```

- All we have to do is define the show action

```
def show
end
```

```
4  projects = [
5    {
6      id: 1,
7      name: "rails project",
8      github_status: false
9    },
10   {
11     id: 2,
12     name: "terminal app",
13     github_status: true
14   }
15 ]
16 render json: projects
17 end
18
19 def show
20 end
21 end
```

- Now let's refresh our browser.
- Add we see a different error (see image below):

The screenshot shows a Chrome browser window. The address bar says 'localhost:3000/projects/1'. The main content area has a red header bar with the text 'No template for interactive request'. Below it, a message reads 'ProjectsController#show is missing a template for request formats: text/html'. A note below states: 'Unless told otherwise, Rails expects an action to render a template with the same name, contained in a folder named after its controller. If this controller is an API responding with 204 (No Content), which does not require a template, then this error will occur when trying to access it via browser, since we expect an HTML template to be rendered for such requests. If that's the case, carry on.' There is a bullet point list below this note.

NOTE!

Unless told otherwise, Rails expects an action to render a template with the same name, contained in a folder named after its controller. If this controller is an API responding with 204 (No Content), which does not require a template, then this error will occur when trying to access it via browser, since we expect an HTML template to be rendered for such requests. If that's the case, carry on.

- Now it's saying that the show action is not sending back a response. (that's what missing a template refers to).
- We can fix this by using the render key word and add some plain text to see it's working

```
def show
  render plain: "sending back 1 project"
end
```

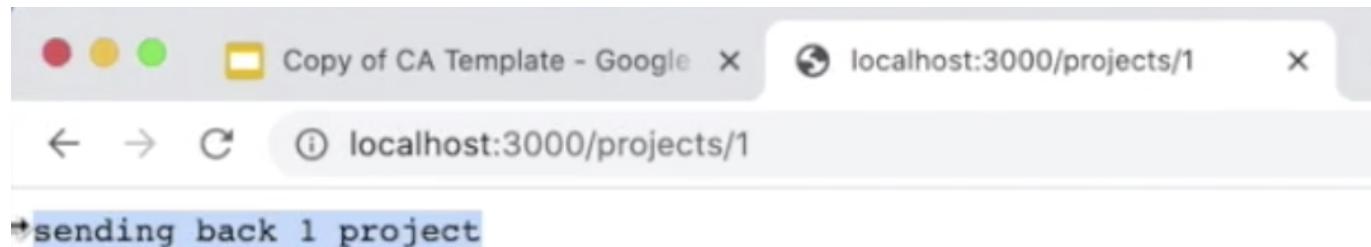
The screenshot shows a terminal window with the following code:

```
def show
  render plain: "sending back 1 project"
end
```

Below the code, the terminal shows the output of running the application:

```
abc project
abc projects
abc ProjectsController
```

- Now let's refresh the browser to see if this renders.



- Although it is working, it is not very useful given we are just sending back plain text and not being back a project from our array of hashes.

The image shows two tabs in a code editor: 'routes.rb' and 'projects_controller.rb'. The 'projects_controller.rb' tab is active and contains the following code:

```
3   # send back all of the resources to the client
4   projects = [
5     {
6       id: 1,
7       name: "rails project",
8       github_status: false
9     },
10    {
11      id: 2,
12      name: "terminal app",
13      github_status: true
14    }
15  ]
16  render json: projects
17 end
18
19 def show
20   render plain: "sending back 1 project"
21 end
22 end
```

Finding item

- First step is to grab our array of hashes.
- This won't be very DRY but we can refactor later.



The screenshot shows a code editor with two tabs: "routes.rb" and "projects_controller.rb". The "projects_controller.rb" tab is active, displaying the following code:

```
19 def show
20   projects = [
21     {
22       id: 1,
23       name: "rails project",
24       github_status: false
25     },
26     {
27       id: 2,
28       name: "terminal app",
29       github_status: true
30     }
31   ]
32   render plain: "sending back 1 project"
33 end
34 end
35
```

NOTE: show is for sending back only one resource.

- A method we can use to find one item in an array is:

```
.find
```

- We can call the dot find method on an array (in this case it's our projects array)

```
projects.find
```

- Find then takes a block:

```
projects.find do |project|  
  end
```

- Within the do block we are going to find a project based on its id
- Because each project is a hash, we will use the hash notation:

```
projects.find do |project|  
  project[:id]  
end
```

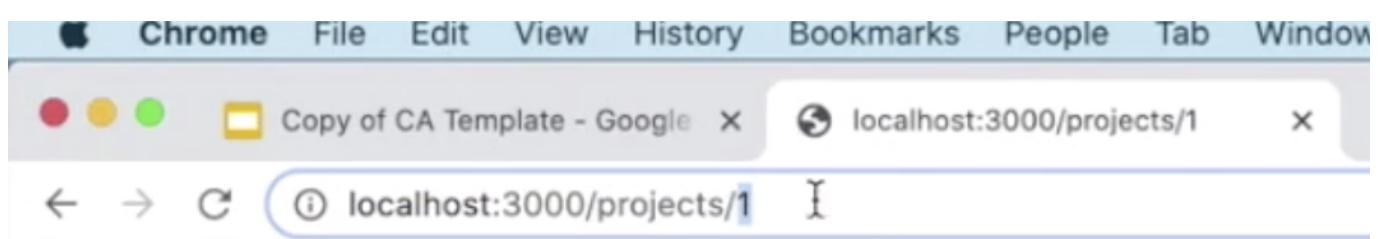
- We want to find the one that is equal to something

```
projects.find do |project|  
  project[:id] ==something  
end
```

The screenshot shows two tabs open in a code editor: 'routes.rb' and 'projects_controller.rb'. The 'routes.rb' file contains the following code:24 name: 'rails project',
25 github_status: false
26 },
27 {
28 id: 2,
29 name: "terminal app",
30 github_status: true
31 }
32]
33
34 projects.find do |project|
35 | project[:id] == something|
36 end
37
38 render plain: "sending back 1 project"
39
40 end
41The 'projects_controller.rb' file contains the following code:

```
name: 'rails project',  
github_status: false  
},  
{  
id: 2,  
name: "terminal app",  
github_status: true  
}  
]  
  
projects.find do |project|  
| project[:id] == something|  
end  
  
render plain: "sending back 1 project"  
end  
end  
  
The line '| project[:id] == something|' is highlighted with a red rectangle.
```

- The "something" is going to represent the number in the url (see image below):



Accessing URL path in controller action

- We can use a special method params
- If we comment out the block of code we just did and print params

The screenshot shows two code files side-by-side in a code editor:

- routes.rb**: Contains a single route definition:

```
get "/projects/:id", to: "projects#show", id: 1
```

- projects_controller.rb**: Contains the following code:

```
class ProjectsController < ApplicationController
  def show
    @project = Project.find(params[:id])
    p params
    # project = Project.find(params[:id])
    # render plain: "sending back 1 project"
  end
end
```

A tooltip "p params" is shown over the `p params` line in the controller code.

- We will be able to see that p statement in our terminal.
- Lets restart our rails server

```
rails s
```

- Then we will go to the route:

```
localhost: 3000/projects/1
```

- This will still work
- Then what we can see is our print statement (see image below):

```
Parameters: {"id"=>"1"}  
<ActionController::Parameters {"controller"=>  
"projects", "action"=>"show", "id"=>"1"} pe  
rmitted: false>  
Rendering text template  
Rendered text template (Duration: 0.0ms |  
Allocations: 3)  
Completed 200 OK in 6ms (Views: 1.8ms | Acti  
veRecord: 0.0ms | Allocations: 1031)
```

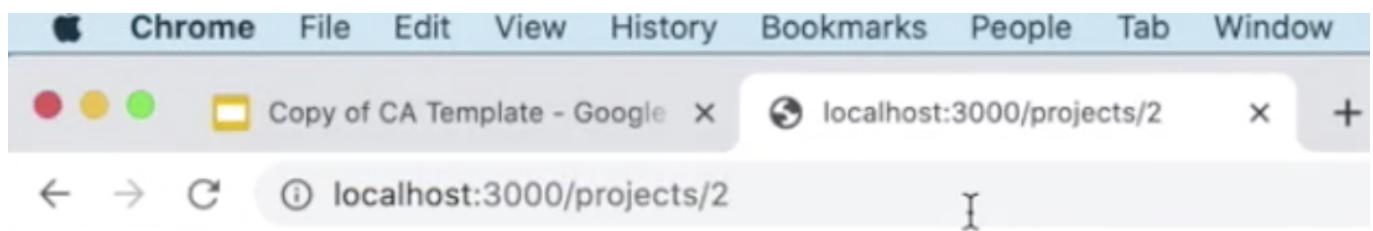
- We printed out params
 - params is a special object in which you can access everything in it using the hash notation.
- In this special params object, you can see that there is an id key:

```
Parameters: {"id"=>"1"}  
<ActionController::Parameters {"controller"=>  
"projects", "action"=>"show", "id"=>"1"} pe  
rmitted: false>  
Rendering text template  
Rendered text template (Duration: 0.0ms |  
Allocations: 3)  
Completed 200 OK in 6ms (Views: 1.8ms | Acti  
veRecord: 0.0ms | Allocations: 1031)
```

- And its value

```
Parameters: {"id=>"1"}  
<ActionController::Parameters {"controller"=>"projects", "action"=>"show", "id=>"1"} permitted: false>  
  Rendering text template  
  Rendered text template (Duration: 0.0ms | Allocations: 3)  
Completed 200 OK in 6ms (Views: 1.8ms | ActiveRecord: 0.0ms | Allocations: 1031)
```

- Through this params method we can access the placeholder or the number that we pass into the URL.
- Therefore in this case we pass in 2



- You will see that the id is the key and 2 is now the value.

```
Started GET "/projects/2" for ::1 at 2020-09-11 10:02:29 +1000  
Processing by ProjectsController#show as HTML  
  Parameters: {"id=>"2"}  
<ActionController::Parameters {"controller"=>"projects", "action"=>"show", "id=>"2"} permitted: false>  
  Rendering text template  
  Rendered text template (Duration: 0.0ms | Allocations: 2)  
Completed 200 OK in 1ms (Views: 0.3ms | Acti
```

Accessing id value

- All we would need to do is use the hash notation (square brackets and name of the key):

```
p params[:id]
```

- This will access the value
- Show if we save this file

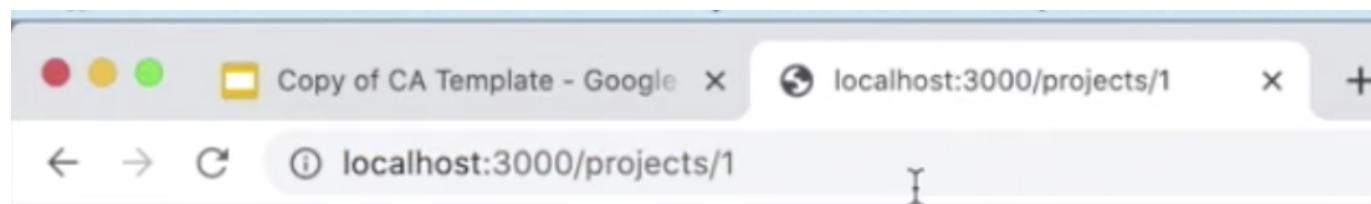
The screenshot shows two code files side-by-side in a code editor:

- routes.rb**: Contains a single route definition:

```
get "/projects/:id", to: "projects#show"
```
- projects_controller.rb**: Contains the controller logic:

```
class ProjectsController < ApplicationController
  def show
    @project = Project.find(params[:id])
    p params[:id]
    # projects.find do |project|
    #   project[:id] ==
    # end
    render plain: "sending back 1 project"
  end
end
```

- Then change our route back to 1



- now we are just accessing 1 in our terminal

```
Started GET "/projects/1" for ::1 at 2020-09  
-11 10:03:23 +1000  
  (0.1ms)  SELECT sqlite_version(*)  
Processing by ProjectsController#show as HTM  
L  
  Parameters: {"id"=>"1"}  
  1"  
  Rendering text template  
  Rendered text template (Duration: 0.0ms |  
Allocations: 2)  
Completed 200 OK in 2ms (Views: 1.5ms | Acti
```

Coming back to our find block

```
p params[:id]  
  
  # projects.find do |project|  
  #   project[:id] ==  
  # end  
  
  render plain: "sending back 1 project"  
end  
end
```

- We are finding a project based on its id in the hash.

```
routes.rb
18
19 def show
20   # show is for sending back only 1 resource
21   projects = [
22     {
23       id: 1,
24       name: "rails project",
25       github_status: false
26     },
27   ]
```

```
projects_controller.rb
18
19 def show
20   # show is for sending back only 1 resource
21   projects = [
22     {
23       id: 1,
24       name: "rails project",
25       github_status: false
26     },
27     {
28       id: 2,
29       name: "terminal app",
30       github_status: true
31     }
32   ]
```

```
routes.rb
18
19 def show
20   # show is for sending back only 1 resource
21   projects = [
22     {
23       id: 1,
24       name: "rails project",
25       github_status: false
26     },
27     {
28       id: 2,
29       name: "terminal app",
30       github_status: true
31     }
32   ]
```

```
projects_controller.rb
18
19 def show
20   # show is for sending back only 1 resource
21   projects = [
22     {
23       id: 1,
24       name: "rails project",
25       github_status: false
26     },
27     {
28       id: 2,
29       name: "terminal app",
30       github_status: true
31     }
32   ]
```

- Then we are going to find a particular project:

```
projects.find do |project|
  project[:id] ==
end
```

- Based on the param that we are working we:

```
p params[:id] |
```



```
projects.find do |project|
  project[:id] ==
end
```

- So we can copy this value and paste into our find (see image below):

The screenshot shows two tabs open in a code editor: 'routes.rb' and 'projects_controller.rb'. The 'routes.rb' tab is active, displaying the following code:

```
25     github_status: false
26   },
27   {
28     id: 2,
29     name: "terminal app",
30     github_status: true
31   }
32 ]
33
34 projects.find do |project|
35   project[:id] == params[:id]
36 end
37
38 render plain: "sending back 1 project"
39 end
40 end
41
```

The 'projects_controller.rb' tab is visible in the background. The code editor has a dark theme with syntax highlighting.

- We can then store the result of the find in a variable (found_project in this example):

```
found_project = projects.find do |project|
  project[:id] == params[:id]
end
```

- Then we can print found_project

```
p found_project
```

- Send the request again:

```
localhost:3000/projects/1
```

- And at the moment the return of our found_project is nil

```
Started GET "/projects/1" for ::1 at 2020-09  
-11 10:04:47 +1000  
  (0.1ms)  SELECT sqlite_version()  
Processing by ProjectsController#show as HTM  
L  
  Parameters: {"id"=>"1"}  
nil  
  Rendering text template  
  Rendered text template (Duration: 0.0ms |  
Allocations: 1)  
Completed 200 OK in 2ms (Views: 1.2ms | Acti
```

- The reason that it is nil is because:
- The param we are working with:

```
found_project = projects.find do |project|  
  project[:id] == params[:id]  
end
```

- This id that we are pulling out of params is actually a string.
- Whereas the project[:id]

```
projects = [
  {
    id: 1,
    name: "rails project",
    github_status: false
},
```

- that we are getting from the hash:
- is an integer.
- So we are comparing an integer to a string.
- In this case we have to do some explicit type casting and converting the string to an integer.

```
found_project = projects.find do |project|
  project[:id] == params[:id].to_i
end
```

- Then we can save our file and refresh our browser
- Now we are pulling out one single hash.

```
Parameters: {"id"=>"1"}  
{:id=>1, :name=>"rails project", :github_status=>false}  
Rendering text template  
Rendered text template (Duration: 0.0ms |  
Allocations: 1)  
Completed 200 OK in 1ms (Views: 0.7ms | Acti
```

- We can confirm this by sending back that hash to the client.

Sending back hash to client:

- Instead of plain text we can render JSON and send back the found_project to the client side.

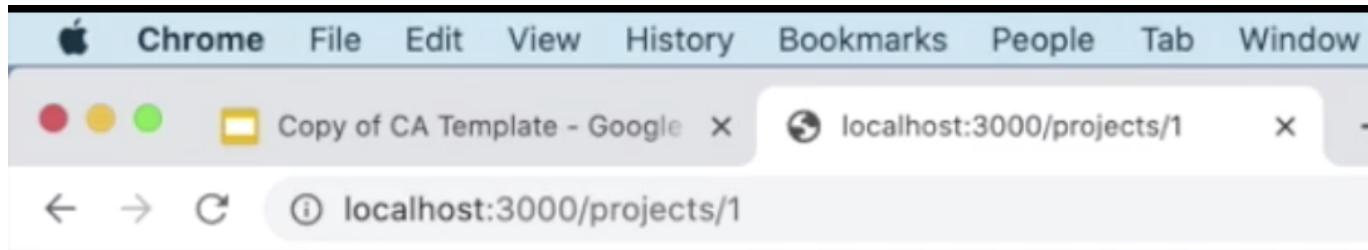
The screenshot shows a code editor with two tabs open: 'routes.rb' and 'projects_controller.rb'. The 'routes.rb' file contains a single line: 'resources :projects'. The 'projects_controller.rb' file contains the following code:

```
20 # show is for sending back only 1 resource
21 projects = [
22   {
23     id: 1,
24     name: "rails project",
25     github_status: false
26   },
27   {
28     id: 2,
29     name: "terminal app",
30     github_status: true
31   }
32 ]
33
34 found_project = projects.find do |project|
35   project[:id] == params[:id].to_i
36 end
37
38 render json: found_project
39
40 end
```

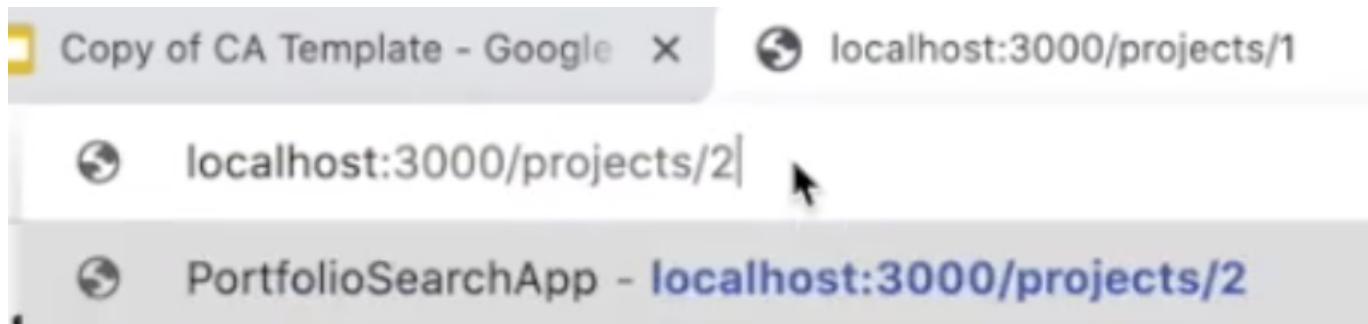
- Lets restart our server:

```
rails s
```

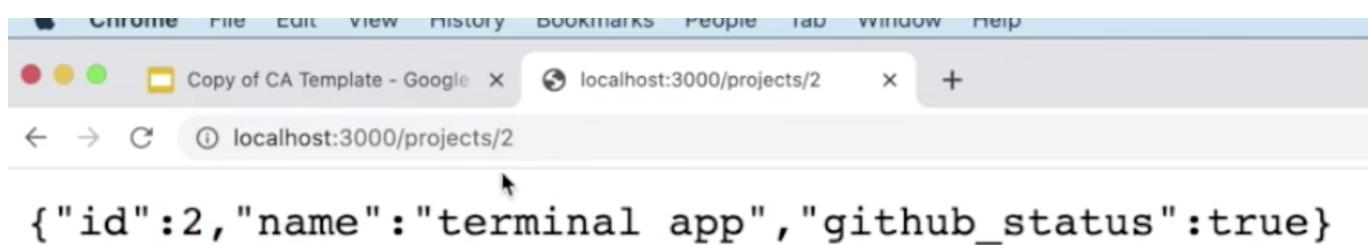
- refresh our browser



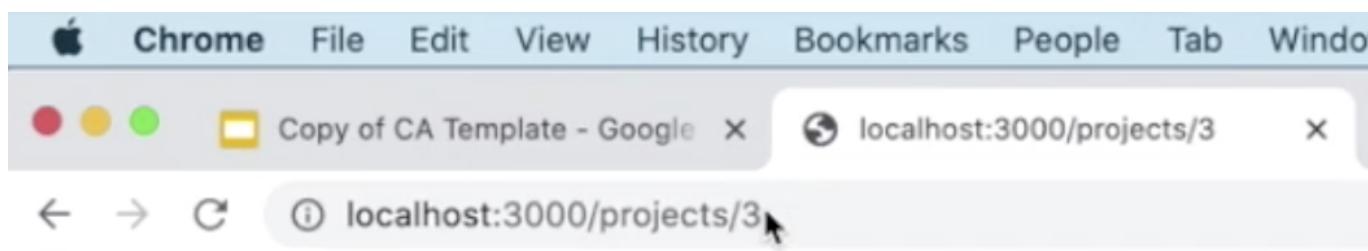
- And now we are getting just the first project of id 1.
- If we pass in 2 the browser:



- We will get the second project of id 2:



- If we pass in 3 we will get a return of null:



- This is because we don't have a project with an id of 3.
- We only have projects with an id of 1 and 2.

The screenshot shows a code editor with two tabs open: 'routes.rb' and 'projects_controller.rb'. The 'routes.rb' tab is on the left, and the 'projects_controller.rb' tab is on the right. The code in 'routes.rb' contains a single line: 'resources :projects'. The code in 'projects_controller.rb' is as follows:

```
18
19  def show
20    # show is for sending back only 1 resource
21    projects = [
22      {
23        id: 1,
24        name: "rails project",
25        github_status: false
26      },
27      {
28        id: 2,
29        name: "terminal app",
30        github_status: true
31      }
32    ]
33
34    found_project = projects.find do |project|
35      project[:id] == params[:id].to_i
36    end
37
38    render json: found_project
```

Params

```
found_project = projects.find do |project|
  project[:id] == params[:id].to_i
end
```

- Params is a special method that we have access to in each controller action
- The id represents a key in our params.

```
found_project = projects.find do |project|
  project[:id] == params[:id].to_i
end
```

- The name of this key depends on what we have named it in our routes.rb folder.

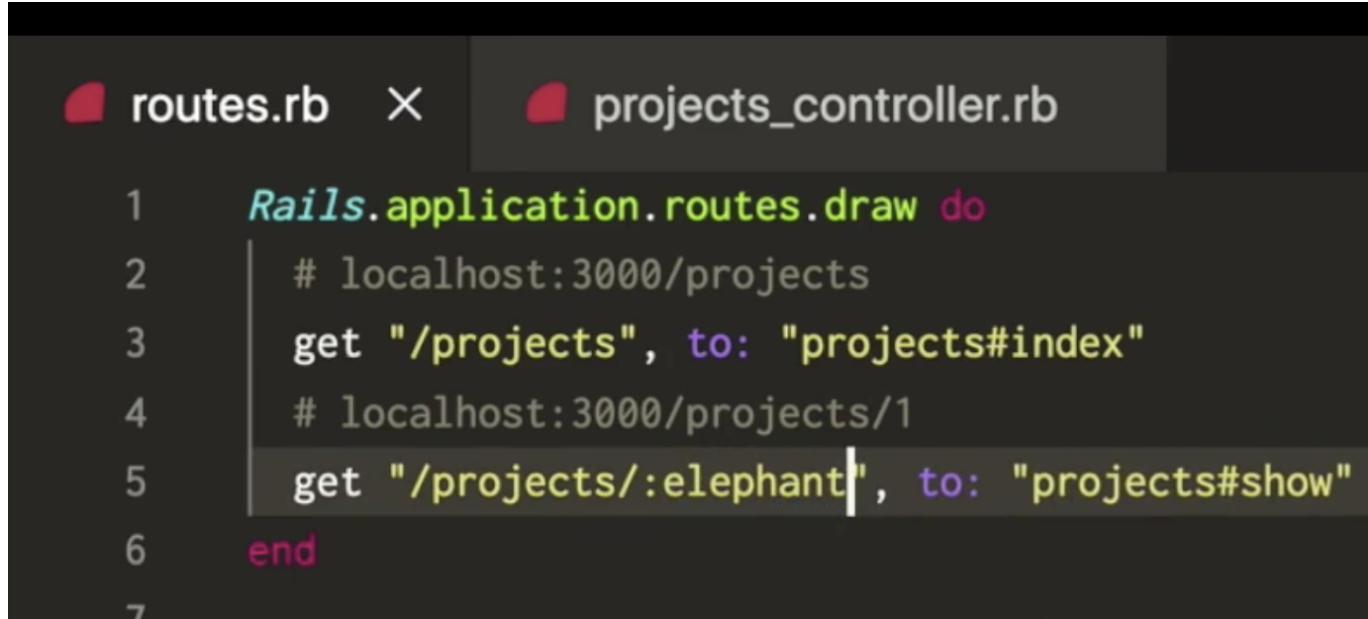
The screenshot shows a code editor with two tabs: "routes.rb" and "projects_controller.rb". The "routes.rb" tab is active, displaying the following code:

```
1 Rails.application.routes.draw do
2   # localhost:3000/projects
3   get "/projects", to: "projects#index"
4   # localhost:3000/projects/1
5   get "/projects/:id", to: "projects#show"
6 end
```

- Therefore the id in our routes folder is very important because:
 - Whatever we have in our params depends on whatever name we use in our route.

For example:

- We could name it elephant:



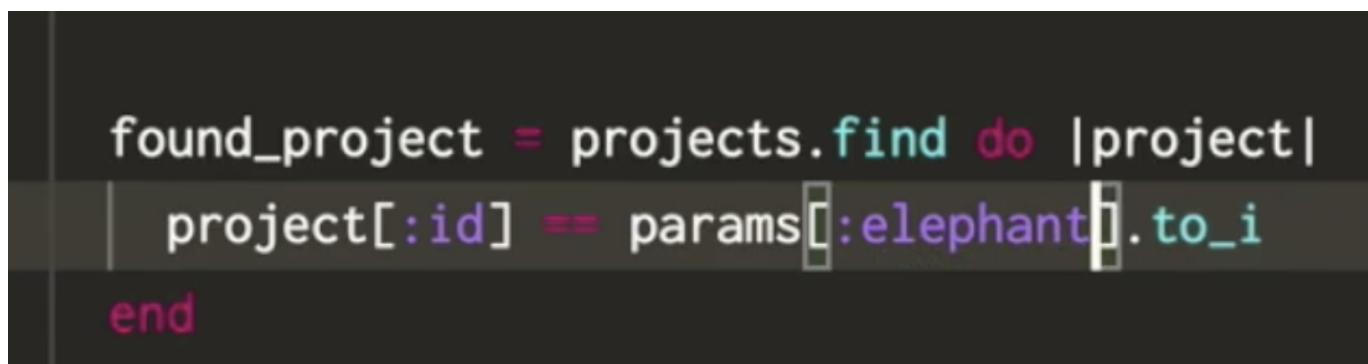
```

routes.rb  X  projects_controller.rb

1   Rails.application.routes.draw do
2     # localhost:3000/projects
3     get "/projects", to: "projects#index"
4     # localhost:3000/projects/1
5     get "/projects/:elephant", to: "projects#show"
6   end
7

```

- We then change the name in our projects_controller.rb folder:



```

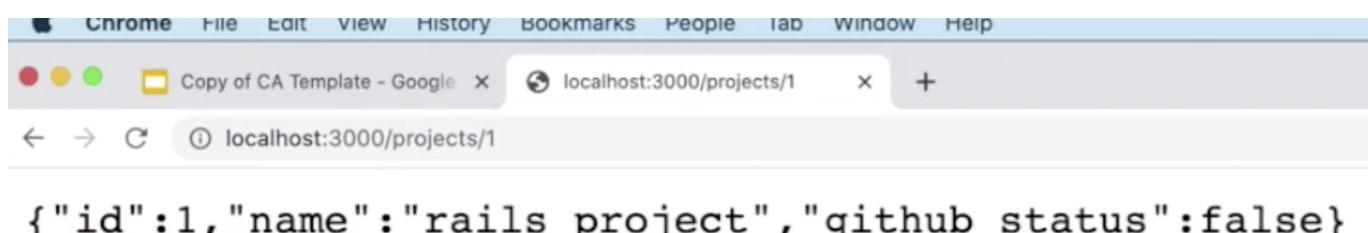
found_project = projects.find do |project|
  project[:id] == params[:elephant].to_i
end

```

- We can then run our rails server:

```
rails s
```

- Then send a request again with a valid project id (1 in this case)
- And we can see that things still work.



```
{"id":1,"name":"rails project","github_status":false}
```

- We can see this is updated in our terminal too:

```
(0.8ms) SELECT sqlite_version(*)  
Processing by ProjectsController#show as HTML  
L  
Parameters: {"elephant"=>"1"}  
Completed 200 OK in 5ms (Views: 0.3ms | ActiveRecord: 0.0ms | Allocations: 350)
```

- Therefore the key that we have access to in params is dependant on the name that we use in our routes.