



RESKILL



# RUBY ON RAILS INTERVIEW QUESTIONS



RESKILL

We gathered top interview questions that will help you land the ruby on rails position you desire.

for more visit: [ReSkill.me](https://reskill.me)





HOW DOES A SYMBOL  
DIFFER FROM A  
STRING?




# HOW DOES A SYMBOL DIFFER FROM A STRING?


Short answer: symbols are immutable and reusable, retaining the same object\_id.

Be prepared to discuss the benefits of using symbols vs. strings, the effect on memory usage, and in which situations you would use one over the other.





WHAT IS A MODULE? CAN  
YOU TELL ME THE  
DIFFERENCE BETWEEN  
CLASSES AND MODULES?



# WHAT IS A MODULE? CAN YOU TELL ME THE DIFFERENCE BETWEEN CLASSES AND MODULES?

Modules serve as a mechanism for namespaces.

```
module ANamespace
  class AClass
    def initialize
      puts "Another object, coming right up!"
    end
  end
end
```

```
ANamespace::AClass.new
#=> Another object, coming right up!
```

More on the next slide...



# WHAT IS A MODULE? CAN YOU TELL ME THE DIFFERENCE BETWEEN CLASSES AND MODULES?

Also, modules provide as a mechanism for multiple inheritance via mixins and cannot be instantiated like classes can.

```
module AMixin
  def who_am_i?
    puts "An existentialist, that's who."
  end
end

# String is already the parent class
class DeepString < String
  # extend adds instance methods from AMixin as class methods
  extend AMixin
end

DeepString.who_am_i?
#=> An existentialist, that's who.

AMixin.new
#=> NoMethodError: undefined method 'new' for AMixin:Module
```



EXPLAIN THIS  
RUBY IDIOM:  $A \parallel = B$

*EXPLAIN THIS RUBY IDIOM: A ||= B*

a = b when a == false

Otherwise a remains unchanged.



WHAT'S THE ISSUE WITH  
THE CONTROLLER CODE  
BELOW? HOW WOULD YOU  
FIX IT?





# WHAT'S THE ISSUE WITH THE CONTROLLER CODE BELOW? HOW WOULD YOU FIX IT?

```
class CommentsController < ApplicationController
  def users_comments
    posts = Post.all
    comments = posts.map(&:comments).flatten
    @user_comments = comments.select do |comment|
      comment.author.username == params[:username]
    end
  end
end
```

This is a classic example of the "n+1" bug. The first line will retrieve all of the Post objects from the database, but then the very next line will make an additional request for each Post to retrieve the corresponding Comment objects. To make matters worse, this code is then making even more database requests in order to retrieve the Author of each Comment. This can all be avoided by changing the first line in the method to:

```
posts = Post.includes(comments:
[:author]).all
```

This tells ActiveRecord to retrieve the corresponding Comment and Author records from the database immediately after the initial request for all Posts, thereby reducing the number of database requests to just three.





WHAT PATHS (HTTP VERB AND  
URL) WILL BE DEFINED BY THE  
FOLLOWING SNIPPET IN CONFIG/  
ROUTES.RB?



```
resources :posts do
  member do
    get 'comments'
  end
  collection do
    post 'bulk_upload'
  end
end
```

## WHAT PATHS (HTTP VERB AND URL) WILL BE DEFINED BY THE FOLLOWING SNIPPET IN CONFIG/ROUTES.RB?

Using the resource method to define routes will automatically generate routes for the standard seven restful actions:

- GET /posts
- POST /posts
- GET /posts/new
- GET /posts/:id/edit
- GET /posts/:id
- PATCH/PUT /posts/:id
- DELETE /posts/:id
- GET /posts/:id/comments
- POST /posts/bulk\_upload





HOW WOULD YOU CREATE  
GETTER AND SETTER  
METHODS IN RUBY?



# HOW WOULD YOU CREATE GETTER AND SETTER METHODS IN RUBY?

Setter and getter methods in Ruby are generated with the `attr_accessor` method. `attr_accessor` is used to generate instance variables for data that's not stored in your database column.

You can also take the long route and create them manually.



EXPLAIN A  
POLYMORPHIC  
ASSOCIATION:



# EXPLAIN A POLYMORPHIC ASSOCIATION:

employees	
Model: <b>Employee</b> has_many :pictures, :as => :imageable	
id	integer
name	string

products	
Model: <b>Product</b> has_many :pictures, :as => :imageable	
id	integer
name	string

pictures	
Model: <b>Picture</b> belongs_to :imageable, :polymorphic => true	
id	integer
name	string
imageable_id	integer
imageable_type	string


Polymorphic associations allow a model to belong to more than one other model through a single association.



```
class Picture < ActiveRecord::Base
  belongs_to :imageable, polymorphic: true
end

class Employee < ActiveRecord::Base
  has_many :pictures, as: :imageable
end

class Product < ActiveRecord::Base
  has_many :pictures, as: :imageable
end
```

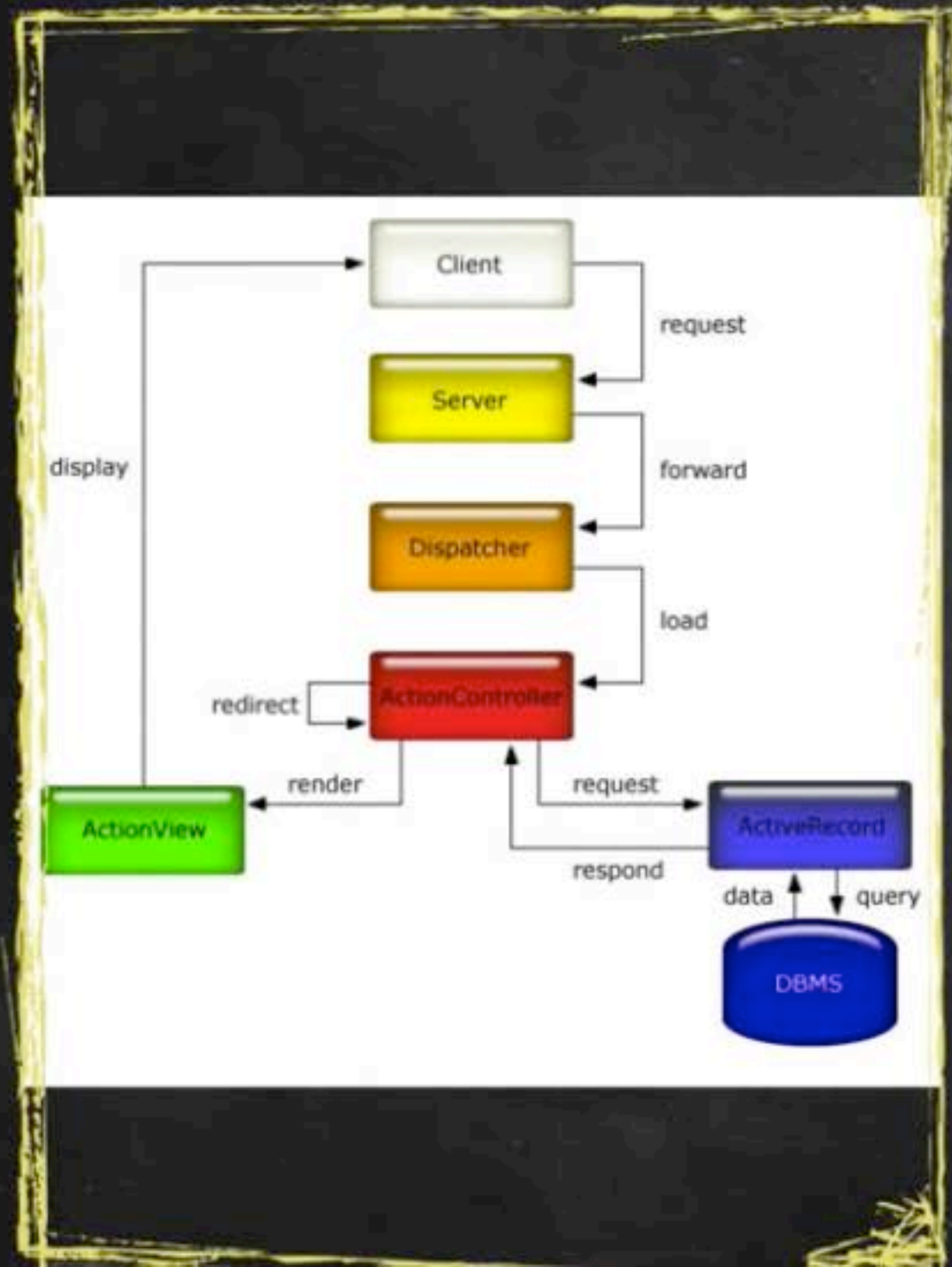


Here, the class Picture belongs\_to both Employee and Product, but does so through a single association rather than through multiple.

Be sure to know an appropriate situation to create a polymorphic association, such as creating a comment model associated with multiple other models (articles, photos, etc.). The advantage of using polymorphic here is that it allows you to create a single comment model, rather than separate models for each one (PhotoComment model, ArticleComment model, etc.)

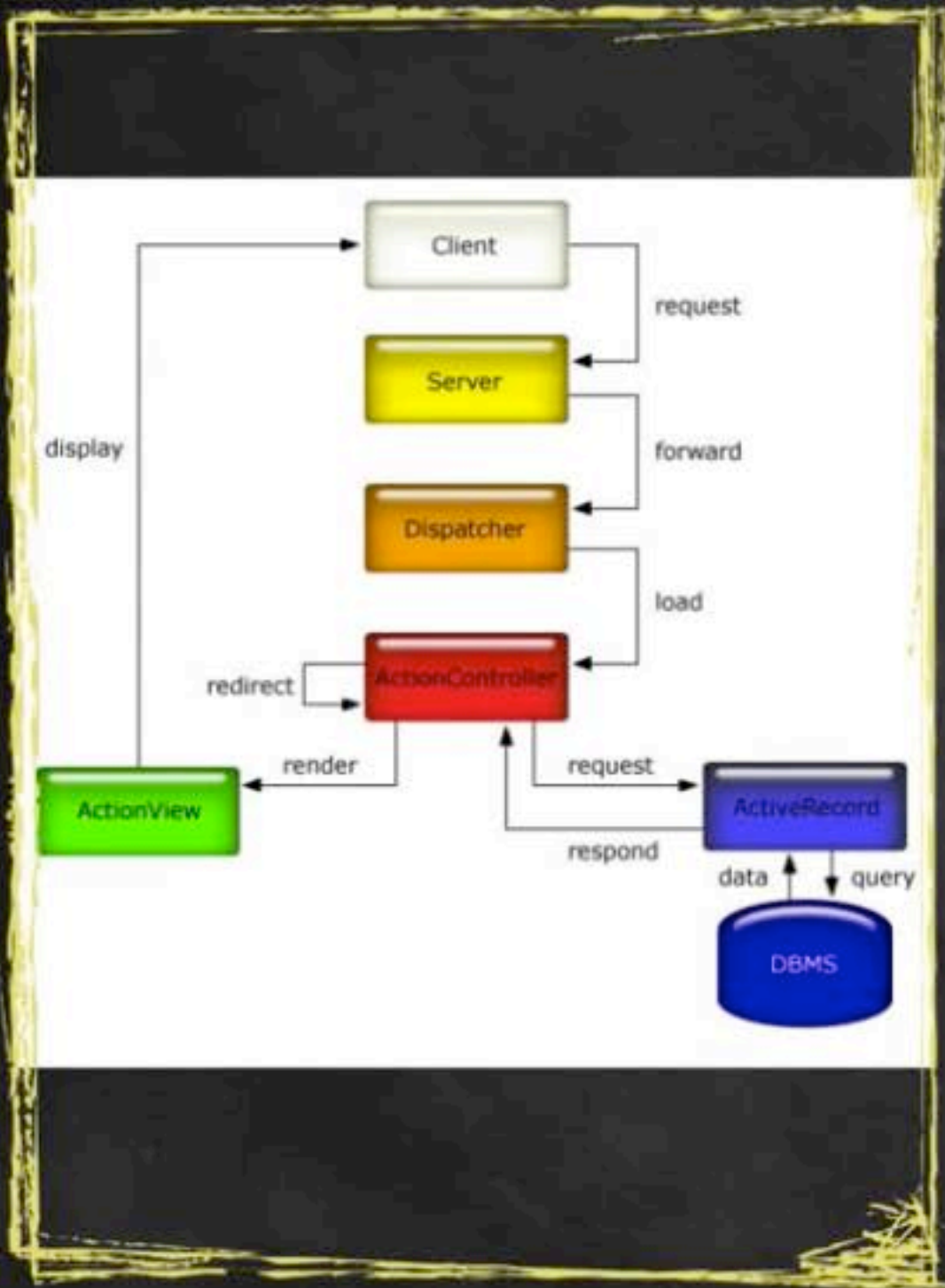


# Define the Rails MVC implementation using an example.



- **Model (ActiveRecord)** – Maintains the relationship between Object and Database and handles validation, association, transactions, and more. This layer provides an interface and binding between the tables in a relational database and the Ruby program code that manipulates database records. Ruby method names are automatically generated from the field names of database tables, and so on.





- Controller (ActionController) – The facility within the application that directs traffic, on the one hand querying the models for specific data, and on the other hand organizing that data (searching, sorting, massaging it) into a form that fits the needs of a given view.
- View (ActionView) – A presentation of data in a particular format, triggered by a controller's decision to present the data. Every Web connection to a Rails application results in the displaying of a view.



WHAT IS A FILTER ?  
WHEN IT IS CALLED?





# WHAT IS A FILTER ? WHEN IT IS CALLED?

Filters are methods that are called either before/after a controller action is called.

Say a user requests a controller action such as `userdashboard/index`. In such a case a filter can be setup so that the `UserDashboard/index` page is only accessible to loggedin users by adding the following lines towards the beginning of the page:

```
class UserDashboardController < ApplicationController
  before_filter :confirm_logged_in, :except => [:login, :attempt_login, :logout]
  def index
    ....
  end

  def login
    ....
  end

  def attempt_login
    ....
  end

  def logout
    ....
  end
end
```



# RESKILL, REACH YOUR DREAM JOB



## COURSE RECOMMENDATIONS

Curated courses to help you to acquire your missing skills.



## INDUSTRY UPDATES

Automatically stay up to date with the latest articles and industry updates for your field.



## PERSONALIZED JOB SEARCH

Discover relevant job opportunities that you are currently qualified for and apply right away.



RESKILL, BECOME THE  
TALENT EMPLOYERS ARE  
LOOKING FOR!

