**Rails Important Material**

HTTP vs HTTPs

UDP vs TCP

UDP (user datagram protocol)

TCP is a connection-oriented protocol, whereas UDP is a connectionless protocol. A key difference between TCP and UDP is speed, as TCP is comparatively slower than UDP. Overall, UDP is a much faster, simpler, and efficient protocol, however, retransmission of lost data packets is only possible with TCP.

Rails components

In addition to that, Rails also comes with:

* [Action Mailer](https://github.com/rails/rails/blob/main/actionmailer/README.rdoc), a library to generate and send emails
* [Action Mailbox](https://github.com/rails/rails/blob/main/actionmailbox/README.md), a library to receive emails within a Rails application
* [Active Job](https://github.com/rails/rails/blob/main/activejob/README.md), a framework for declaring jobs and making them run on a variety of queuing backends
* [Action Cable](https://github.com/rails/rails/blob/main/actioncable/README.md), a framework to integrate WebSockets with a Rails application
* [Active Storage](https://github.com/rails/rails/blob/main/activestorage/README.md), a library to attach cloud and local files to Rails applications
* [Action Text](https://github.com/rails/rails/blob/main/actiontext/README.md), a library to handle rich text content
* [Active Support](https://github.com/rails/rails/blob/main/activesupport/README.rdoc), a collection of utility classes and standard library extensions that are useful for Rails, and may also be used independently outside Rails

**Rails main components**

The framework’s main components are ActionPack, ActiveSupport, ActiveModel, ActiveRecord, ActiveResource, and ActionMailer. Essentially, Rails is a framework composed of other frameworks that can be used independently.

* **Action Pack:** Handles requests and responses. This framework is part of Rails’s Model View Controller pattern, which serves the Web request, handling, routing, and view generation. To provide the response, it defines controllers that implement actions in order to render views.
* **Active Model:** Provide the interfaces for the Model part of Model View Controller. This component is new in Rails 3.0. In previous versions, the model layer was based on ActiveRecord. In the current version, you can use any ruby class as a Model. This is very important because you can use your own persistence layer and glue it into Rails.
* **Active Record:** This is the [Object Relational Mapping](http://en.wikipedia.org/wiki/Object-relational_mapping) (ORM) component of Rails, with a very nice zero-configuration feature. Naming and convention is the key to maintaining simple and minimal code to define classes that will be persisted in the database tables.
* **Active Support:** a collection of utility classes and standard library of extensions that are useful in Rails. We can find this extensions useful for several Ruby projects.
* **Active Resource:** Connects business objects and Representational State Transfer ([REST](http://tomayko.com/writings/rest-to-my-wife)) Web services. With ActiveResource, you can easily use REST to expose your ActiveRecord models with just a small amount of code. This is a useful way to create an API without much effort.
* **Action Mailer:** This framework provides the email service layer, helping out to send the forgot password emails, registration emails, invoices for billing, etc. This class wraps ActionController from ActionPack to render the emails as page views, with the same render and templates like pages.

Assets pipeline:

Main features:

1. Fingerprinting: Fingerprinting is a technique that makes the name of a file dependent on the contents of the file. When the file contents change, the filename is also changed. For content that is static or infrequently changed, this provides an easy way to tell whether two versions of a file are identical, even across different servers or deployment dates.
2. Minification: concatenate and minify or compress JavaScript and CSS assets.
3. All JS files are concatenated into a single .js file and all CSS files are concatenated into a single .css file. This allows the browser to make fewer requests so our pages load more quickly.
4. Precompilation: The third feature of the asset pipeline is it allows coding assets via a higher-level language, with precompilation down to the actual assets. Supported languages include Sass for CSS, CoffeeScript for JavaScript, and ERB for both by default.

HashWithIndifferentAccess

Implements a hash where keys :foo and "foo" are considered to be the same.

rgb = ActiveSupport::HashWithIndifferentAccess.new

rgb[:black] = '#000000'

rgb[:black] *# => '#000000'*

rgb['black'] *# => '#000000'*

You can set the :autosave option on a [has\_one](https://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html" \l "method-i-has_one), [belongs\_to](https://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html" \l "method-i-belongs_to), [has\_many](https://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html" \l "method-i-has_many), or [has\_and\_belongs\_to\_many](https://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html" \l "method-i-has_and_belongs_to_many) association. Setting it to true will *always* save the members, whereas setting it to false will *never* save the members. More details about :autosave option is available at [AutosaveAssociation](https://api.rubyonrails.org/classes/ActiveRecord/AutosaveAssociation.html).

***Transpilation****:* conversion of high level language to high level language

*Coffe to js*

**Minification** is the process of minimizing code and markup in your web pages and script files. It's one of the main methods used to **reduce load times and bandwidth usage on websites**. Minification dramatically improves site speed and accessibility, directly translating into a better user experience.

Form with:

Url , method if no url then ‘/’ root, model

Form for: only accept model

**Minitest vs RSpec in rails**

[https://www.honeybadger.io/blog/minitest-rspec-rails/#:~:text=RSpec%20has%20the%20same%20goals,(BDD)%20and%20specification%20writing.](https://www.honeybadger.io/blog/minitest-rspec-rails/" \l ":~:text=RSpec has the same goals,(BDD) and specification writing.)

**rails dbconsole**

**Rails Important Question**exists? vs present?

services vs concerns

A service object in Rails is a plain old Ruby object created for a specific business action. It usually contains code that doesn't fit in the model or view layer, e.g., actions via a third-party API like posting a tweet.

dependent options (destroy, delete, nullify, raise\_with\_exception, raise\_with\_error)

actiondispatch vs actioncontroller

architectural components of rails

framework vs library

what happens when you delete: gemfile, gemfile.lock

devise customization

udp vs tcp

http vs https

application server vs web server

where are static files located

rake tasks what, why, how

custom rake tasks what, why, how (give example usecases to make custom rake tasks)

webpacker

minification

scss vs sass

delete vs destroy

dirty module

fat vs slim controllers

has\_and\_belongs\_to\_many vs has\_many through

form\_with vs form\_tag vs form\_for

ajax, what

remote form submission, what

backend vs frontend pagination

polymorphic associations, what, why, how

n+1 query, what, how to solve

includes vs eager\_load vs preload

action mailer, what, why

devise, what, why

background jobs, what, why

manage secrets/keys

minitest vs rspec

what is rack

Scope vs class methods

String vs Symbol

How ROR implements Ajax

Mixens

Rails Observers

Garbage Collection

Polymorphic Association

super vs super call

Proc vs lambda

include vs extend

load vs require

Exception Handling

What is cron job?

gem vs plugin?

Revise Rails directory structure

How to write custom rake task?

precompile vs live compile?

HashWithIndifferentAccess? Turbolink? Transpilation?

Dirty module in active model?

How to pass/fail custom validation?

Does before commit exists in rails?

How to use around callback?

up vs down vs change vs irreversible migration?

polymorphic association? STI?

Has and belongs to many vs has many through

groups in Gemfile?

scope routes? shallow nesting?

Enum? built in methods of enum?

mount in routes?

How to pass data from controller to view? From view to partial? View to layout?

How to do remote form submission?

respond\_to in rails?

Does asset pipeline work in mailer view?

What is layout?

How includes solve n+1 query? eager\_load vs includes?

Helper vs service vs concern?

HTTP Verbs/methods

Form submission with GET request

PUT vs PATCH in rails

Dirty module

practice gemfile delete

require in gemfile

Db seed

rake task example

scss

JQuery

turbo links store i18n

after\_commit

rake db:midrate down

setup reset db migrate

js type request

scope vs method

pagination without gem

eager loading with

find\_each vs find\_in\_batches

manifest or rspec ka directory structure

jobs callbacks

constraint of dependant destroy in migration

Turbo Links

ORM

Filter vs Map

HashWithInDifferentAccess

Model Validations

How to skip Validations

arrount callbacks

database validation

middle wares

can we write query in migration file

join vs include

resource vs resources

can rails support multiple inheritance

how to check which table are migrated in rails

Meta programming

dependent nullify and other option

content\_for pass data from view to layout...

return type if no data found in class vs scope

enums db storage

gemfile require false

find vs find\_by

minification things

select vs where

update and update\_attribute

Migration (Rails version) [5.7] why?

Custom foreign key..

Rack Apps (Rails injection)

dependent (types) options

present? exists?

members vs. collection

Send data from view to layout

Scope and class methods.

Lazy/pre/eager loading.

enums

gem version (require ‘false’)

limit, take

permit, permit!

mailes view VS controller views

minification

find\_or\_create, find\_or\_create\_by

Select, where

Jobs, Transactions

DOM update /update /attributes do seret, db.drop

what if we del gemfile will it effect gemfile.lock

which module goes to controller to model?am,ar

assests in directory folder?

Pass data from view to layout?

Scopes vs class methods (models)

test commit changes to db after\_save

functions of enums/store only integer value to db or character

solutions of n+1 problems

chaining in scopes(models)

.take->get any random model from

length-> all record fetch then return count

update and update column same as put vs patch

send data to partials using locals

what if we del gemfile will it effect gemfile.lock

no effect on gemfile.lock

* which module goes to controller to model?ar
* assets folder in directory?5.2  
  Vendor ,lib ,app,tmp,
* Pass data from view to layout?  
  locals  
  content\_for in view and yeild in layout
* Scopes vs class methods (models)
* test commit changes to db after\_save  
  Registers a callback to be called after a record is saved.
* explore enums
* functions of enums/store only integer value to db or character?

**You can't have an enum with an underlying type of string** . The underlying type can be any integral type except char . If you want to translate a string to your enum then you'll probably need to use the Parse or TryParse methods.

The short answer is no. You will need to use a gem (such as [magic-enum](https://github.com/kovyrin/magic-enum)) if you want to do anything but store integers.

* solutions of n+1 problems,,
* Length VS Count  
  length will include the number of objects you've built but haven't saved, . count will report only the count from the database.  
  Length

size checks if the data was loaded (i.e. already in rails) if so, then just count it, otherwise it calls count. (plus the pitfalls, already mentioned in other entries).

* job.events.length – This method simply loads the data of the association into the memory and then returns the number of elements loaded.
* If all entries are already loaded in the memory (Job.all), then use length to avoid another database query.

But, a major point to NOTE here is that this method won’t force an update to previously loaded association, if associated records we created through another way, like Job.all instead of job.events.create

### count

* job.events.count – This method determines the number of elements with an SQL COUNT query.
* If you do not have anything loaded in the memory, use count to make a query on database.
* You can also give some specific condition to get the count of a subset of associated elements. For example: job.events.where("events.start\_date\_time > ?", Time.now)
* NOTE here that if you set up a counter cache on the association, count will return that cached value instead of executing a new query.

### size

* job.events.size – This method works as a combination of both the above mentioned methods i.e. if the collection has already been loaded in the memory, it will return the length same as calling length. If it has not been loaded yet, it is like calling count.
* If no consideration bothers you, use size that will adapt to any situation.
* But NOTE that size is a bit unsafe command to use as it is automated and sometimes you do want to query the database again.­
* Put VS Patch
* Update\_column VS Update vs update\_attribute  
  **update**(id, attributes) *public*

Updates an object (or multiple objects) and saves it to the database, if validations pass. The resulting object is returned whether the object was saved successfully to the database or not.

#### Parameters

* [id](http://apidock.com/rails/ActiveRecord/Base/id) – This should be the [id](http://apidock.com/rails/ActiveRecord/Base/id) or an array of ids to be updated.

[attributes](http://apidock.com/rails/ActiveRecord/Base/attributes) – This should be a [hash](http://apidock.com/rails/ActiveRecord/Base/hash) of [attributes](http://apidock.com/rails/ActiveRecord/Base/attributes) to be set on the object, or an array of hashes.  
 # Updating one record:

[Person](http://apidock.com/rails/Person).[update](http://apidock.com/rails/ActiveRecord/Base/update)(15, :user\_name => 'Samuel', :group => 'expert')

# Updating multiple records:

people = { 1 => { "first\_name" => "David" }, 2 => { "first\_name" => "Jeremy" } }

* [Person](http://apidock.com/rails/Person).[update](http://apidock.com/rails/ActiveRecord/Base/update)(people.keys, people.values)
* **update\_columns**(attributes) *public*

Updates the attributes directly in the database issuing an UPDATE SQL statement and sets them in the receiver:

user.[update\_columns](http://apidock.com/rails/ActiveRecord/Persistence/update_columns)(last\_request\_at: [Time](http://apidock.com/rails/Time).current)

This is the fastest way to [update](http://apidock.com/rails/ActiveRecord/Persistence/update) attributes because it goes straight to the database, but take into account that in consequence the regular [update](http://apidock.com/rails/ActiveRecord/Persistence/update) procedures are totally bypassed. In particular:

* [Validations](http://apidock.com/rails/ActiveRecord/Validations) are skipped.
* [Callbacks](http://apidock.com/rails/ActiveRecord/Callbacks) are skipped.
* updated\_at/updated\_on are not updated.

This method raises an +[ActiveRecord::ActiveRecordError+](http://apidock.com/rails/ActiveRecord/ActiveRecordError) when called on new objects, or when at least one of the attributes is marked as readonly

* **update\_attributes**(attributes) *public*

Updates [all](http://apidock.com/rails/ActiveRecord/Base/all/class) the [attributes](http://apidock.com/rails/ActiveRecord/Base/attributes) from the passed-in [Hash](http://apidock.com/rails/Hash) and saves the record. If the object is invalid, the saving will fail and false will be returned.

* Namespaces VS Scopes
* .touch
* Destroy Vs Delete  
  delete will only delete current object record from db but not its associated children records from db.

destroy will delete current object record from db and also its associated children record from db.

* Chaining
* Running queries and what differences includes, preload and eager\_load have.

[.statusses,

.first.status,

?,

!] -> functions of enums

* library vs framework
* how to skip validation model level
* database validation
* up vs down vs change
* fat vs slim controller
* where does the foreign key store
* how csrf token works
* resource vs resources
* named helper in routes
* collection vs member routes
* pre loading, eager loading, includes
* exists vs presents
* select vs pluck
* form\_tag, form\_for