Table of Contents

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, a library to generate and send emails
- Action Mailbox, a library to receive emails within a Rails application
- Active Job, a framework for declaring jobs and making them run on a variety of queuing backends
- Action Cable, a framework to integrate WebSockets with a Rails application
- Active Storage, a library to attach cloud and local files to Rails applications
- Action Text, a library to handle rich text content
- Active Support, a collection of utility classes and standard library extensions that are useful for Rails, and may also be used independently outside Rails

Web Socket:

WebSocket is bidirectional, a full-duplex protocol that is used in the same scenario of client-server communication, unlike HTTP it starts from ws:// or wss://. It is a stateful protocol, which means the connection between client and server will keep alive until it is terminated by either party (client or server). After closing the connection by either of the client and server, the connection is terminated from both ends.

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.. It provides mechanisms for routing (mapping request URLs to actions), defining controllers that implement actions, and generating responses by rendering views, which are templates of various formats.
- **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 (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) 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:

The asset pipeline provides a framework to concatenate and minify or compress JavaScript and CSS assets. It also adds the ability to write these assets in other languages and pre-processors such as

CoffeeScript, Sass, and ERB. It allows assets in your application to be automatically combined with assets from other gems.

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`, `belongs_to`, `has_many`, or `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`.

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.

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.

csrf

There are two components to CSRF. First, a unique token is embedded in your site’s HTML. That same token is also stored in the session cookie. When a user makes a POST request, the CSRF token from the HTML gets sent with that request. Rails compares the token from the page with the token from the session cookie to ensure they match.

CSRF (Cross-Site Request Forgery) is a method of attack that “works by including malicious code or a link in a page that accesses a web application the user is believed to have authenticated.

Briefly, Cross-Site Request Forgery (CSRF) is an attack that allows a malicious user to spoof legitimate requests to your server, masquerading as an authenticated user.

Credensials

The master key:

When you create a new rails app a file called credentials.yml.enc is added to the config directory. This file will be decrypted in a production environment using a key stored either on a RAILS_MASTER_KEY environment variable or a master.key

master.key is needed everywhere, and we should make sure that our team members also get this master.key file. When we want to deploy to the server, we should put what is inside the master.key to the environment variable. According to rubyonrails.org, Rails uses config/master.key or alternatively looks for the environment variable ENV["RAILS_MASTER_KEY"] to encrypt the credentials file.

Protocols

SSL stands for Secure Sockets Layer, and it refers to a protocol for encrypting, securing, and authenticating communications that take place on the Internet.

Handshake:

The handshake protocol uses the public key infrastructure (PKI) and establishes a shared symmetric key between the parties to ensure confidentiality and integrity of the communicated data. The handshake involves three phases, with one or more messages exchanged between client and server: 1.

Step by step :

1. The client sends a request to the server for a secure session. The server responds by sending its X.509 digital certificate containing server’s public key to the client.
2. The client receives the server's X.509 digital certificate.
3. The client authenticates the server, using a list of known certificate authorities.
4. The client generates a random symmetric key and encrypts it using server's public key.
5. The client and server now both know the symmetric key and can use the SSL encryption process to encrypt and decrypt the information contained in the client request and the server response.

Scaffolding

Rails *scaffolding* is a quick way to generate some of the major pieces of an application. If you want to create the models, views, and controllers for a new resource in a single operation, scaffolding is the tool for the job.

Load path

Ruby by default has a list of directories it can look through when you ask it to load a specific file. This is stored in a variable: $: This is the load path. It initially includes the libdir, archdir, sitedir, vendordir and some others and is information Ruby holds about itself. If you type ruby -e 'puts $LOAD_PATH'

ActiveModel:

Active Model is a library containing various modules used in developing classes that need some features present on Active Record. So ActiveModel includes things like validations.

1) When including ActiveModel::API you get some features like:

model name introspection

conversions

translations

validations

initialize object

2) The ActiveModel::AttributeMethods module can add custom prefixes and suffixes on methods of a class. It is used by defining the prefixes and suffixes and which methods on the object will use them.

3) ActiveModel::Callbacks gives Active Record style callbacks. This provides an ability to define callbacks which run at appropriate times. After defining callbacks, you can wrap them with before, after, and around custom methods.

4) An object becomes dirty when it has gone through one or more changes to its attributes and has not been saved. ActiveModel::Dirty gives the ability to check whether an object has been changed or not. It also has attribute-based accessor methods.

5) The ActiveModel::Validations module adds the ability to validate objects like in Active Record.

6) ActiveModel::Model allows implementing models similar to ActiveRecord::Base

7) ActiveModel::Serialization provides basic serialization for your object. You need to declare an attributes Hash which contains the attributes you want to serialize. Attributes must be strings, not symbols.

8) ActiveModel::SecurePassword provides a way to securely store any password in an encrypted form. When you include this module, a has_secure_password class method is provided which defines a password accessor with certain validations on it.

ActiveRecord:

ActiveRecord is an ORM. It's a layer of Ruby code that runs between your database and your logic code.

Active Record facilitates the creation and use of business objects whose data requires persistent storage to a database. It is an implementation of the Active Record pattern which itself is a description of an Object Relational Mapping system.

This is the component that associates a class to the database. This will give the class functionality such as methods that make it easy to pull records from the database (An example is the find method).

ActiveResource:

Similar to ActiveRecord. However, instead of being backed by a database, an ActiveResource object is backed by another application through a web service API. More information: http://ofps.oreilly.com/titles/9780596521424/activeresource_id59243.html

“ORM:

An object that wraps a row in a database table or view, encapsulates

the database access, and adds domain logic on that data.”

Active Record callbacks

Creating an Object

1. before_validation
2. after_validation
3. before_save
4. around_save
5. before_create
6. around_create
7. after_create
8. after_save
9. after_commit/after_rollback

Updating an Object

1. before_validation
2. after_validation
3. before_save
4. around_save
5. before_update
6. around_update
7. after_update
8. after_save

9. after_commit/after_rollback

Destroying an Object

1. before_destroy
2. around_destroy
3. after_destroy
4. after_commit/after_rollback

The **after_initialize** callback will be called whenever an Active Record object is instantiated, either by directly using new or when a record is loaded from the database.

The **after_find** callback will be called whenever Active Record loads a record from the database. after_find is called before after_initialize if both are defined.

The **after_touch** callback will be called whenever an Active Record object is touched.

What is serialization?

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database or file. Its main purpose is to save the state of an object in order to be able to recreate it when needed.

Groups in Gemfile:

https://stackoverflow.com/questions/3467177/gemfile-group-is-used-for

"group" method is used for specify gems for specific environments.

```
group :development, :test do
   gem "rspec-rails", ">= 2.0.0.beta.19"
   gem "cucumber-rails", ">= 0.3.2"
   gem "webrat", ">= 0.7.2.beta.1"
end
```

Select vs pluck:

https://findnerd.com/list/view/Select-Vs-Pluck-in-Rails/19258/

select is used to fetch records with specific attributes. It returns an ActiveRecord::Relation object.

pluck can be used the same way select is used, however it returns an array of selected attributes.

Select takes much time on building ActiveRecord models. On large data_set pluck is much faster than select.

Pluck should be used when only column values are required. Select should be used when object is required. pluck is an eager method and it hits the database immediately when called.

application server and web server:

https://www.educative.io/answers/web-server-vs-application-server

https://www.ibm.com/cloud/learn/web-server-vs-application-server

A web server delivers static web content—e.g., HTML pages, files, images, video—primarily in response to hypertext transfer protocol (HTTP) requests from a web browser.

An application server typically can deliver web content too, but its primary job is to enable interaction between end-user clients and server-side application code—the code representing what is often called business logic—to generate and deliver dynamic content, such as transaction results, decision support, or real-time analytics.

HTTP headers

An HTTP header is a field of an HTTP request or response that passes additional context and metadata about the request or response.

Rake vs Rack

Rake is a task runner.

Rack helps Ruby servers & frameworks work together.

Custom Rake tasks

Create file in lib/tasks say apple.rake

task :apple do

puts "Eat more apples!"
end

run in console:
rake apple

Render Partial:

https://stackoverflow.com/questions/13426618/whats-the-difference-between-using-object-collection-and-locals-in-partial

<% @user.comments.each do |comment| %>

<%= render partial: 'comments/comment', locals: { comment: comment } %>

<% end %>

****

```
<h2>Comments:</h2>
<!-- This will now render the partial in app/views/comments/_comment.html.erb
once for each comment that is in the @article -->
<div id="comments">
<%= render @article.comments %>
</div>

<hr>

<!-- rendering all comments manually -->
<!-- <% @article.comments.each do |comment| %>
<p>
<strong>Commenter:</strong>
<%= comment.commenter %>
</p>

<p>
<strong>Comment:</strong>
```

```
<%= comment.body %>
</p>
<% end %> -->
```

resources vs resource

Notice the differences between the two by looking at what actions (or routes) they have:

resources: **Index**, new, create, show, edit, update, destroy
resource: new, create, show, edit, update, destroy

Action Dispatcher

Action Dispatch routes requests to controllers.

find vs find_by with array values

Article.find([4,5])

get record with id 4 and 5

give error if record not found: ActiveRecord::RecordNotFound

Article.find_by(id:[4,5])

get only single record

do not give error if record not found (return nil)

Services vs concerns

https://dev.to/joker666/ruby-on-rails-pattern-service-objects-b19

https://stackoverflow.com/questions/46457227/concerns-vs-services#:~:text=Concerns%20and%20services%20are%20very,Concerns%20are%20mixins.

Concerns and services are very different abstraction patterns used for entirely different purposes.

Services are operations/functions turned into classes. (add features in service, not related to controller)

Concerns are mixins. (common code of controllers)

Service objects are used primarily to wrap a method in an object. These are useful in splitting logic up into small reusable components.

What Is A Transaction?

Transactions are typically used when you need to ensure data integrity, even if your web app crashes in the middle of a request.

Transactions are protective blocks where SQL statements are only permanent if they can all succeed as one atomic action. The classic example is a transfer between two accounts where you can only have a deposit if the withdrawal succeeded and vice versa. Transactions enforce the integrity of the database and guard the data against program errors or database break-downs. Basically, you should use transaction blocks whenever you have a number of statements that must be executed together, or not at all.

Why do we write custom rake tasks?

What is a task?

- Making a backup of your database
- Running your tests
- Gathering & reporting stats

These are small tasks that without Rake would be scattered all over your project on different files. Rake centralizes access to your tasks.
Rake also makes a few things easier, like finding files that match a certain pattern & that have been modified recently.
Rake is a popular task runner for Ruby and Rails applications. For example, Rails provides the predefined Rake tasks for creating databases, running migrations, and performing tests. You can also create custom tasks to automate specific actions - run code analysis tools, backup databases, and so on.

any difference b/w gems and library:

Gem is just a fancy term for a packaged Ruby library.

A library is just a bit of code providing a set of functionalities to anyone who integrates it in its code.

The only thing you need to know to understand the difference with an engine is that a gem is pure Ruby code.

Gems in Rails are libraries that allow any Ruby on Rails developer to add functionalities without writing code.

gems vs Plugins:

A Rails plugin is either an extension or a modification of the core framework.

A gem is a packaged Ruby application or library.

When you install a plugin in a project it can be used only in the respective project. But if u install a gem, it can be used by every project. This is the main difference of Gem & Plugins.

So, the biggest difference between the 2 is that Rails plugins are specifically made for use within Ruby on Rails applications, whereas gems aren't.

The basic difference is a gem is something that needs to be installed on the system running your Rails application, whereas a plugin is deployed along with your application. More specifically, plugins live in vendor/plugins whereas gems need to be install using rake gem install gem_name.

As for when to use each, gems tend to be easier to keep up to date, but more specifically, some gems use native C code and are compiled specifically for a given operating system (such as Nokogiri). These need to be installed as gems as they will not work when moved to another system. Whereas some things like

acts_as_commentable use straight ruby code and can be moved from system to system.

Assets pipeline

The asset pipeline provides a framework to concatenate and minify or compress JavaScript and CSS assets. It also adds the ability to write these assets in other languages and pre-processors such as CoffeeScript, Sass, and ERB. It allows assets in your application to be automatically combined with assets from other gems.

Sprocket:

The asset pipeline is implemented by the sprockets-rails gem,

Minification:

compress files, removing tabs and spaces

Concatination:

Sprockets concatenates all JavaScript files into one master .js file and all CSS files into one master .css file.

Precompilation

Precompile - happens before you send your code to production using rake assets:precompile. It makes sure your code is compilable (made up word) and put them in public/assets

We use rake assets:precompile to precompile our assets before pushing code to production. This command precompiles assets and places them under the public/assets directory in our Rails application.

In production, Rails precompiles these files to public/assets by default. The precompiled copies are then served as static assets by the web server. The files in app/assets are never served directly in production.

Preprocessing

Preprocess - a process that transforms your code into either css, html or js before you sent it to your browser. This is usually when you use (redundant here) preprocessors such as coffeescript, erb, scss, etc. I must say that the order is important when using preprocess.

Transpilation:

High to high language transformation

Need to revise how to rollback multiple migrations at once.

https://stackoverflow.com/questions/3647685/how-to-rollback-a-specific-migration

rake db:migrate:status – to know status of migrations

rake db:rollback STEP=2
this will step down last two migrations

rake db:migrate VERSION=20100905201547
this will rollback up to this migration (this specified version file will remain up)

In order to rollback ONLY ONE specific migration (OUT OF ORDER) use:rake db:migrate:down VERSION=20100905201547

And if you ever want to migrate a single migration
rake db:migrate:up VERSION=20100905201547

Which migrations are not reversible through change action in migration

https://api.rubyonrails.org/classes/ActiveRecord/IrreversibleMigration.html

https://guides.rubyonrails.org/v5.2/active_record_migrations.html#using-the-change-method

The `change` method is the primary way of writing migrations. It works for the majority of cases, where Active Record knows how to reverse the migration automatically. Currently, the `change` method supports only these migration definitions:

add_column
add_foreign_key
add_index
add_reference
add_timestamps
change_column_default (must supply a :from and :to option)

```
change_column_default :products, :approved, from: true, to: false
```

change_column_null
create_join_table
create_table
disable_extension
drop_join_table
drop_table (must supply a block)
enable_extension
remove_column (must supply a type)
remove_foreign_key (must supply a second table)
remove_index
remove_reference
remove_timestamps
rename_column
rename_index
rename_table

`change_table` is also reversible, as long as the block does not call `change`,`change_default` or `remove`.

`remove_column` is reversible if you supply the column type as the third argument. Provide the original column options too, otherwise Rails can't recreate the column exactly when rolling back

What is require false means in gemfile ?

You use :require => false when you want the gem to be installed but not "required".

Specify :require => false to prevent bundler from requiring the gem, but still install it and maintain dependencies.

*****

If a gem's main file is different than the gem name, specify how to require it

gem 'rack-cache', :require => 'rack/cache'

What is shallow nesting of routes?

https://medium.com/@jaredrayjohnson1/4-things-i-learned-toying-around-with-nested-resources-in-rails-fed6d761e924

```
resources :articles, shallow: true do
  resources :comments

  resources :quotes

  resources :drafts
end
```

****

```
resources :artists, shallow: true do
  resources :songs
end
```

Shallow nesting Adding the ‘shallow: true’ parameter to your nested resource will define routes at the base level, ‘/songs/’, for four of our RESTful routes — show, edit, update and destroy, while leaving the remaining routes — index, new, create— at the nested level. So how is this useful to us?

In my model, with artists and songs, we are now able to use the routes starting with ‘/songs/:id’ to view or modify existing songs on an individual basis. Recall that songs belong to artists, so upon creation, we still need to initialize a song with an artist relationship. However, once a song is created, it doesn’t matter to artist relationship if we would like to edit or delete it. And so, we don’t need to follow a lengthy route, such as ‘/artists/:artist_id/songs/:id/edit’ to make our changes.

exists? vs present?

Present initialize all found records (suppose we find humza name is present or not , and it will initialize all the objects and return it)

exist check if the record found (suppose humza name is present , it will return true afetr checking only first record)

find vs find_by with array values

length vs count

Length:

If all entries are already loaded in the memory (Job.all), then use length to avoid another database query.

Count:

If you do not have anything loaded in the memory, use count to make a query on database.

Everytime it use sql count query to get result

Size:

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.

Note for Length Count Size:

If new record added, size(when its act as length, data loaded it act like length) and length will not show updated result.

Why do we use custom views helpers?

A helper is a method that is (mostly) used in your Rails views to share reusable code. Rails comes with a set of built-in helper methods. One of these built-in helpers is time_ago_in_words . This method is helpful whenever you want to display time in this specific format.

need to revise asynchronous behaviour

Redis is a database that we will use to queue our jobs (key value store)

Sidekiq is a ruby library that manage worker processes in rails applications and it will read off the queue to perform the job we put on.

Redis-cli

KEYS *

exit

How secret keys are managed?

Credentials are stored and accessed using two files:

1. credentials.yml.enc – This is an encrypted file that stores all the credentials. It is a YAML file that is encrypted by using the master key. Since this is encrypted, it is safe to push this file to the remote repository.
2. master.key – To decrypt the encrypted file, this file is used. It is impossible to read the encrypted file if this key is lost/modified. And, this should not be pushed to the remote repository.

Revise syntax for custom transactions

https://blog.appsignal.com/2022/03/30/5-tips-to-design-ruby-on-rails-transactions-the-right-way.html

```
ActiveRecord::Base.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

```
******************************
```

```
def some_method
  User.transaction do
    user.perform_action!
    user.perform_another_action!
  end
rescue SomeError
  # rescue
end
```

Webpacker

Webpacker is a tool that integrates Webpack with a Rails application. It makes it easy to configure and develop JavaScript-like applications and optimize them for the production environment.

Webpack

Webpack helps us organize our JavaScript file code to avoid errors and improve a website’s performance. It organizes the code into bundles.

A bundle is a file where multiple modules are intelligently placed, respecting the dependency graph that is first created. Thanks to this process, we can be sure that our code will work as expected and before we invoke a given library’s code, it will have been loaded.

Bundler

Pacakager manager for rails

STI association

Revise Up down vs change

Functionalities of assets pipeline?

hashwithindifferentaccess?

Revise Model Callback Sequence

Remote form submission - request will render which file by default if form is submitted as remote true?

Resources vs resource in routing?

Scope vs namespace?

Shallow nesting of routes?

model scope?

different options for dependent key in has_many association?

Different options to solve n+1 and what is the difference between them?

Why use custom views helpers?

best practices in views?

Service vs concerns?

Practice Queries

Custom Transaction?

find vs find_by with array values

Revise STI association