Web Dev - Intro to Ruby on Rails and MVC

Hoosier Hacks 2023-10-22

What is Ruby?

- High Level
- Interpreted Language
- Aimed for Simplicity and Productivity
- Everything is an Object (including primitive types)
  - can invoke methods directly
- Most famous for its use in web dev

A bit of syntax

```ruby
puts 'Hello world'
       # Hello world

puts 'Name?'
name = gets.chomp
puts "Hello #{name}."

a = [8, 'yes', 1.1, [3, 5]]
puts a[2]
       # 1.1
```

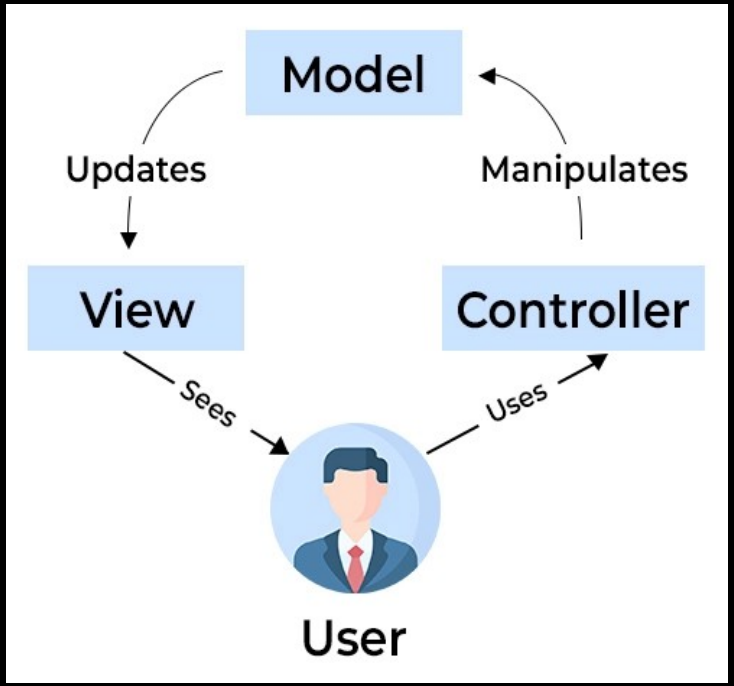
```ruby
def some_function(number)
  if number.even?
    puts "It is not odd"
  else
    puts "Is it odd?"
  end
end

some_function(185)
       # Is it odd?
```

What is Rails?

- Made in 2004
- Ruby Framework for web development
  - Examples include: GitHub, GitLab, Twitch, Hulu, Shopify
- Model-View-Controller Framework (MVC)
- Batteries Included (has all the parts required for it to work built in)

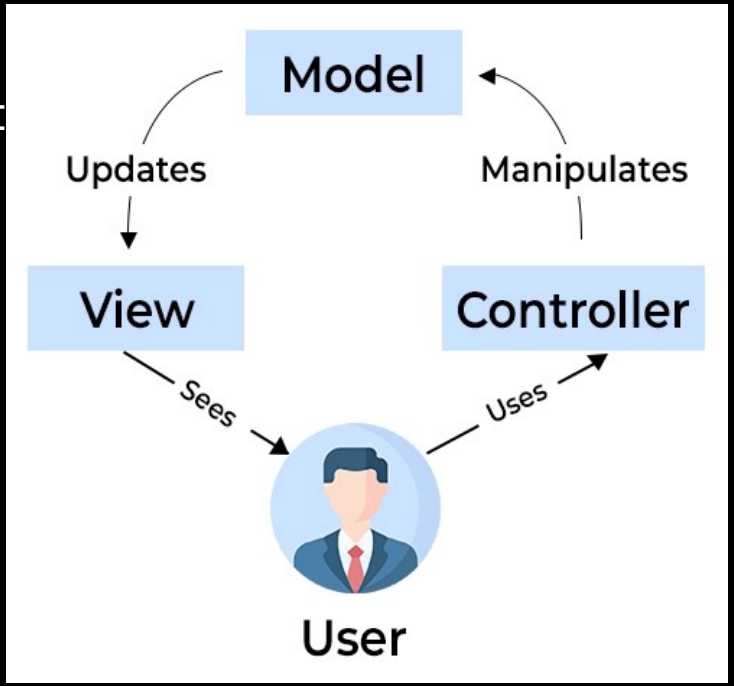
MVC - Model-View-Controller

- General pattern concept on which Ruby on Rails is structured
- Usually used with GUIs for interactive apps
  - → became popular on web apps



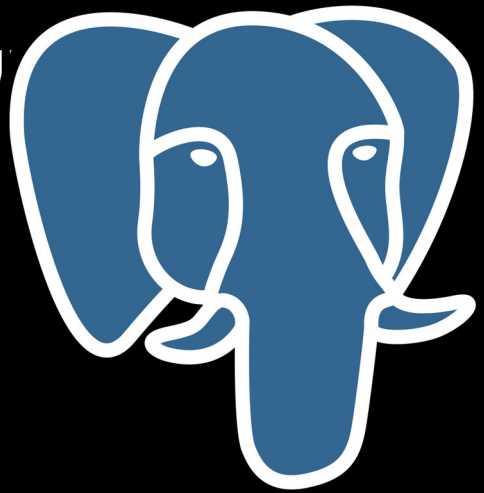
MVC - Model

- Short for “Data Model” - a type of Active Record pattern which has basic capabilities such as Insert, Update, Delet and to store information in a database
- Defines how data is stored
- Manages the “rules” and “logic” of app
- In Rails, the model represents a table in the app’s database (db)



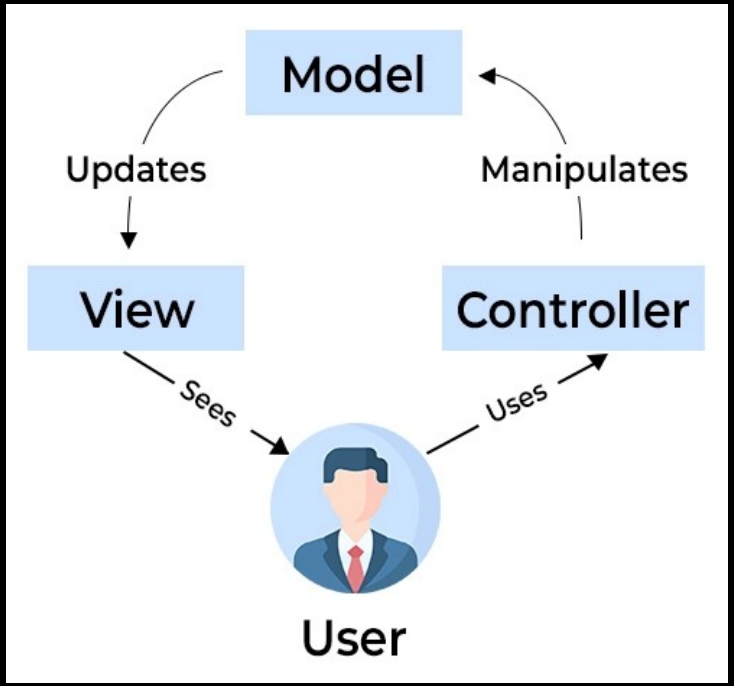
MVC - Model - Database

- Definition - an organized collection of data stored and accessed locally
  - Interactive software systems enable users to define, create, maintain, and control access of databases → ex. SQL, Oracle Database, etc.
  - Ruby is used to manage databases in a form called migrations (shown later)
- Databases like PostgreSQL used in Ruby on Rails appl
  many cells & tables



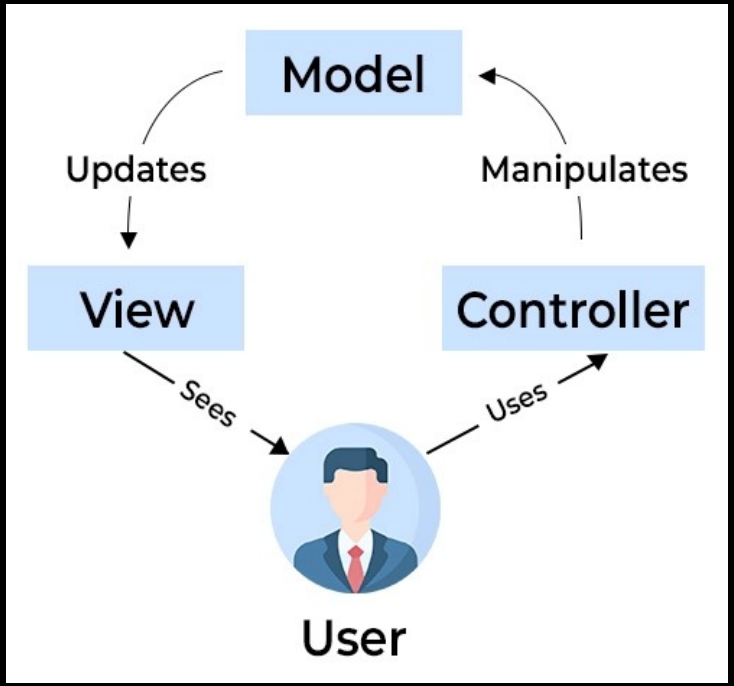
MVC - View

- The actual webpage that you see
  - Frontend
- Allows users to interact with:
  - HTML
  - CSS
  - Javascript

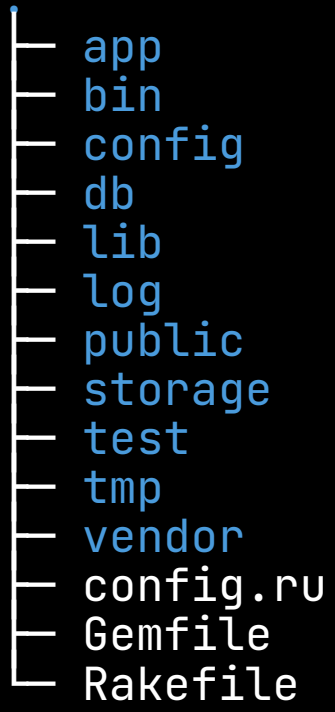


MVC - Contoller

- Handles data input relating to the Model
- Interacts and manages data
  - Most of your server side logic goes here!
  - Migrations are called to manipulate the models in the database



How does a Rails Project Actually Work?



Typical Project Structure

The app Directory

```
└ app
    ├ assets # assets and stylesheets (CSS/Sass)
    ├ channels
    ├ controllers
    │  └ application_controller.rb # e.g. posts_controller.rb
    ├ helpers
    ├ javascript
    ├ jobs
    ├ mailers
    ├ models
    │  └ application_record.rb # e.g. post.rb
    └ views
       └ layouts
          └ application.html.erb
```

Controllers

Seven basic functions of controllers:

- index
- show
- new
- edit
- create
  - params
- update
  - param
- destory

Other functions relating to other model types can also exist

Controllers - Example

```ruby
class PostsController < ApplicationController
  before_action :set_post, only: %i[ show edit update destroy ]

  # GET /posts or /posts.json
  def index
    @posts = Post.all
  end

  # GET /posts/1 or /posts/1.json
  def show
  end

  # GET /posts/new
  def new
    @post = Post.new
  end

  # GET /posts/1/edit
  def edit
  end

  # POST /posts or /posts.json
  def create
    @post = Post.new(post_params)

    respond_to do |format|
      if @post.save
        format.html { redirect_to post_url(@post), notice: "Post was successfully created." }
        format.json { render :show, status: :created, location: @post }
      else
        format.html { render :new, status: :unprocessable_entity }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /posts/1 or /posts/1.json
  def update
    respond_to do |format|
      if @post.update(post_params)
        format.html { redirect_to post_url(@post), notice: "Post was successfully updated." }
        format.json { render :show, status: :ok, location: @post }
      else
        format.html { render :edit, status: :unprocessable_entity }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /posts/1 or /posts/1.json
  def destroy
    @post.destroy

    respond_to do |format|
      format.html { redirect_to posts_url, notice: "Post was successfully destroyed." }
      format.json { head :no_content }
    end
  end

  private
    # Use callbacks to share common setup or constraints between actions.
    def set_post
      @post = Post.find(params[:id])
    end

    # Only allow a list of trusted parameters through.
    def post_params
      params.require(:post).permit(:title, :content)
    end
end
```

Models

Defines attributes and relationship between datatypes

Some associations:

- has_many
- belongs_to
- has_and_belongs_to_many
- has_one

Models - Example

post.rb

```ruby
class Post < ApplicationRecord
  has_rich_text :content
end
```

and some more complicated examples from another project:

```ruby
class CardSet < ApplicationRecord
  include FriendlyId
  friendly_id :name, use: :slugged
  has_rich_text :description
  has_many :cards, dependent: :destroy, inverse_of: :card_set
  accepts_nested_attributes_for :cards, reject_if: :all_blank, allow_destroy: true
end
```

The views Directory

```
└─ views
   ├─ layouts
   │  └─ application.html.erb
   └─ posts # example
      ├─ edit.html.erb
      ├─ index.html.erb
      ├─ new.html.erb
      └─ show.html.erb
```

Views

Frontend!

- HTML (erb) file (html.erb)
  - erb - Embedded Ruby
  - HTML with Ruby embedded
    - </> → HTML : <% %> → Ruby
- Ruby-integrated lines often contain methods of relevant object controllers
- Allows interactions between user and controller

application.html.erb

- Default layout for rendering any page
  - All the other files extend onto this one

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Rails App</title>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>

    <%= stylesheet_link_tag "application", "data-turbo-track": "reload" %>
    <%= javascript_importmap_tags %>
  </head>

  <body>
    <%= yield %>
  </body>
</html>
```

Views - Example

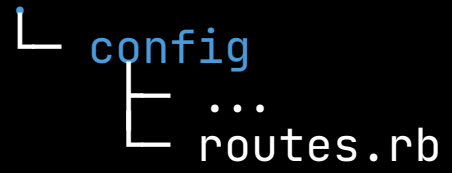
show.html.erb

```
<p style="color: green"><%= notice %></p>

<h1><%= @post.title %></h1>

<br>

<%= @post.content %>

<div>
  <%= link_to "Edit this post", edit_post_path(@post) %> |
  <%= link_to "Back to posts", posts_path %>

  <%= button_to "Destroy this post", @post, method: :delete %>
</div>
```

index.html.erb

```
<p style="color: green"><%= notice %></p>

<h1>Posts</h1>

<div id="posts">
  <% @posts.each do |post| %>
    <%= link_to post.title, post %>
  <% end %>
</div>

<br>

<%= link_to "New post", new_post_path %>
```

The config Directory



Routes

This Ruby file rewrites URL based on certain requests to controllers and actions.

- Can be thought of as a map where requests are directed to

Some keywords:

- resource
- get
- post

Routes - Example

```ruby
Rails.application.routes.draw do
  resources :posts
  # Define your application routes per the DSL in https://guides.rubyonrails.org/routing.html

  # Defines the root path route ("/")
  # root "articles#index"
end
```

Once again... a more complicated example

```ruby
Rails.application.routes.draw do
  # Define your application routes per the DSL in https://guides

  # Reveal health status on /up that returns 200 if the app boot
  # Can be used by load balancers and uptime monitors to verify
  get "up" => "rails/health#show", as: :rails_health_check

  # Defines the root path route ("/")
  # root "articles#index"
  get :upload, to: "uploads#new", as: :new_upload
  post :upload, to: "uploads#create"

  resources :card_sets, path: 's' do
    get "learn", to: "activities#learn"
    get "review", to: "activities#review"
    get "study", to: "activities#study"

    member do
      get :download
    end
  end

  root "main#index"

end
```

The db Directory

```
└── db
    ├── migrate
    │   └── 20231022130643_create_posts.rb
    ├── schema.rb
    └── seeds.rb
```

Migration

Ruby method to manipulate a database

- Very convenient because essentially no data needs to be done other than setting up controllers and actions
  - Databases will not need to be directly changed in SQL language or other nightmare!
  - In fact, it is complicated to directly modify a database → Will need to be manually configured/managed/ran
    - Not recommended for just learning Ruby on Rails
- Different databases can be used in development & production (server use)
  - Very convenient for dynamic purposes

Migration - Example

```ruby
class CreatePosts < ActiveRecord::Migration[7.0]
  def change
    create_table :posts do |t|
      t.string :title

      t.timestamps
    end
  end
end
```

Additional Resources

- The Odin Project
- Ruby on Rails Guides
- Rails API (Official)

Source Code of an Example Project:

https://github.com/WLHackClub/simple-ruby-and-rails-application/

Questions?