Day 1 (02/04/24):

Rails Philosophy

- Don’t Repeat Yourself (Reusable code defined once)
- Convention Over Configuration (Particular conventions followed as good practices)

Installing Rails

I installed and configured the rails and sqlite3 setup using the guide below:
https://gorails.com/setup/ubuntu/22.04

New Application

Can be created using an application generator
rails new blog
Gem dependencies are mentioned in the Gemfile

Important directory structure

app/ controllers, models, viewers, helpers, mailers, assets, channels, jobs
bin/ rails script to start app or set up, update, deploy, run
config/ configuration for routes, databases
db/ current database schema + migrations
Dockerfile (configuration for Docker)
Gemfile (dependencies)
lib/ extra or custom modules etc
public/ static files + compiled assets and exposed
Rakefile (tasks that can be run from command line)
vendor/ third party code + vendored gems

Hello, Rails

Staty the server using bin/rails server
Necessary to have JavaScript runtime available or else execjs error

Say Hello Rails

A route, controller with an action and a view are required.
Route maps requests to an action (controller). The action performs the work to fulfill the request, prepares data for view
Routes file = config/routes.rb

```
Rails.application.routes.draw do
        get “/articles”, to: “articles#index”
End
```

GET /articles -> mapped to index action of ArticlesControllers

Generate controller without route using:
bin/rails generate controller Articles index --skip-routes

```
Class ArticlesController < ApplicationController
        def index
        end
end
```

Rails automatically renders a view with same name of controller and action
app/views/articles.index.html.erb

```
<h1>Hello Rails!</h1>
```

<u>Application Main Page</u>

```
Rails.application.routes.draw do
        root “articles#index”
        get “/articles”, to: “articles#index”
end
```

<u>Autoloading</u>
No need for “require” to load application code. Only use it to 1) load files from lib 2) load gem dependencies

MVC (Model, View, Controller)
Design pattern to divide responsibilities

<u>Generating a Model</u>

Model -> represents data
Interact with database using Active Record

bin/rails generate model Article title:string body:text
Use singular names e.g. Article.new()

Migration File: db/migrate/<timestamp>_create_articles.rb
Model file: app/models/article.rb

<u>Database Migrations</u>

Migrations -> alter the structure of an application’s database
Written in Ruby to be database-agnostic

```
class CreateArticles < ActiveRecord::Migration[7.1]
```

```
def change
      create_table :article do |t|
            t.string :title
            t.text :body

            t.timestamps
     end
  end
end
```

create_table -> adds auto incrementing primary keys e.g. id -> 1, 2 etc
title and body are the columns here: bin/rails generate model Article title:string body:text

```
bin/rails db:migrate
```

<u>**Console for command line**</u>
bin/rails console

<u>**New Object:**</u>
article = Article.new(title: “xyz”, body: “something”)
Object is only initialized for now.

<u>**To save in database use:**</u>
article.save

created_at and updated_at are automatically created

<u>**Fetch Object:**</u>

Article.find(1) -> add id in brackets
To fetch all: Article.all

Showing all Articles using the controller action

```
class ArticlesController < ApplicationController
  def index
            @articles = Article.all
 end
end
```

<u>**CAN ACCESS CONTROLLER INSTANCES IN THE VIEW**</u>

```
<h1>Articles</h1>
```

```
<ul>
  <% @articles.each do |article| %>
        <li>
        <%= article.title %>
        </li>
  <% end %>
</ul>
```

This is ERB + HTML. ERB is a templating system.

- <% %> evaluate the ruby code inside only
- <%= %> evaluate the ruby code + output it

So only the article.title for each article gets outputted.

Procedure:

1. Make a request GET on localhost
2. Rails receives request
3. Router maps root route to index (action) in ArticlesController
4. Index then uses Article model to fetch all articles (database)
5. Rails automatically renders app/views/articles/index.html.erb view
6. ERB code -> Output HTML
7. Server sends a response to browser

CRUD

New View to show a single article:
Map route to new controller action
config.routes.rb

```
Rails.application.routes.draw do
  root "articles#index"

  get "/articles", to: "articles#index"
  get "/articles/:id", to: "articles#show"
end
```

Since we're opening a single article -> path includes :id now
Root parameters in params
GET http://localhost:3000.articles/1

To retrieve use -> params[:id]

```
  def show
        @article = Article.find(params[:id])
  end
end
```

Returned article is stored in @article instance variable

```
<% @article.title %>
<% @article.body %>
```

<u>Resourceful Routing</u>

Entity -> combination of routes, controller actions, views -> resource

```
Rails.application.routes.draw do
  root "articles#index"
  resources :articles
end
```

<u>Routes -> bin/rails routes</u>

URL and path helper methods e.g. article_path returns “/articles/#{article.id}”

```
<h1>Articles</h1>

<ul>
  <% @articles.each do |article| %>
        <li>
        <a href="<%= article_path(article) %>">
        <%= article.title %>
        </a>
        </li>
  <% end %>
</ul>
```

<u>Link_to:</u>

link_to renders a link with first argument as link’s text and second as link’s destination. If article is passed: link_to calls article_path

```
<h1>Articles</h1>

<ul>
  <% @articles.each do |article| %>
        <li>
        <%= link_to article.title, article %>
        </li>
  <% end %>
```

</ul>

Create (CRUD)

Use a form to show to client
User -> submits form
If no error -> resource is created + confirmation shown
Controller’s new and create actions (new only creates an instance, create also saves in database)

```
  def new
        @article = Article.new
  end
  def create
        @article = Article.new(title: "xyjv", body: "fisifs")
        if @article.save
        redirect_to @article
        else
        render :new, status: :unprocessable_entity
        end
  end
end
```

Form Builder
Create a form using form builder
Form_with keyword
label and text_field -> linked to attribute
Results in an HTML output

Strong Parameters
Instead of passing individual parameters use params
params[:article][:title]
params[:article][:body]

Validate values for Hash:

```
private
        def article_params
        params.require(:article).permit(:title, :body)
        End
```

Validations and Error Messages
Validations for invalid user input

```
class Article < ApplicationRecord
  validates :title, presence: true

  validates :body, presence: true, length: { minimum: 10 }
end
```

Error message: @article.errors.full_messages_for(:title).each do |message|

full_messages_for -> array that includes errors and is empty if none.

Visiting form leads to new action
Submits the form using POST / articles

New article can be created by visiting: articles/new
<%= link_to "New Article", new_article_path %>

Update
Use edit and update actions
Edit only fetches article and stores it in @article and update refetches it and attempts to update using submitted form's data (article_params).

Partials

Local data incase views are shared e.g. new and updating forms
Instead of @article, local variable is referenced as article (don't depend of specific instance)

Deletion
Only requires a route + controller action (destroy action) is mapped to DELETE / articles/:id

```
 <li><%= link_to "Destroy", article_path(@article), data: {
                    turbo_method: :delete,
                    turbo_confirm: "Are you sure?"
                    } %></li>
```

data-turbo-method and data-turbo-confirm are included by default.
data-turbo-method="delete" -> DELETE req instead of GET is made

Second Model

```
bin/rails generate model Comment commenter:string body:text article:references
```

belongs_to:article creates an Association (Active Record) Primary Key (ID) of Articles is referenced here as a foreign key.

```
class CreateComments < ActiveRecord::Migration[7.1]
  def change
          create_table :comments do |t|
          t.string :commenter
          t.text :body
          t.references :article, null: false, foreign_key: true
          t.timestamps
          end
  end
end
```

Migration is run using

```
bin/rails db:migrate
```