Práctica 4

1. El framework está compuesto de diferentes librerías:

- ActionMailer
- ActionPack
- ActionView
- ActiveJob
- ActiveModel
- ActiveRecord
- ActiveSupport
- ActionCable
- ActiveStorage
- ActionText
- ActionMailbox

  Para cada una de estas librerías, analizá y respondé las siguientes preguntas:

  1. ¿Qué función principal cumple?
  2. Citá algún ejemplo donde se te ocurra que podrías utilizarla.

- ActionMailer:
  - Permite enviar correos electrónicos desde tu aplicación utilizando clases de mailer y vistas.
  - Ejemplo: enviar un correo de confirmación cuando un usuario se registra:
- ActionPack
  - Es el conjunto de componentes que manejan las peticiones HTTP y las rutas, e incluye ActionController (controladores) y ActionDispatch (enrutamiento y middleware). Gestiona el flujo entre el navegador y la aplicación.
  - Ejemplo: definir una ruta y un controlador para mostrar productos:
- ActionView
  - Se encarga de renderizar las vistas, es decir, de generar el HTML que se envía al navegador.
  - Ejemplo: mostrar los productos en una vista
- ActiveJob

    - Abstrae y unifica el manejo de tareas en segundo plano (background jobs), como el envío de correos o la generación de reportes, para que funcionen con distintos adaptadores (Sidekiq, DelayedJob, etc.).
    - Ejemplo: Enviar un email de manera asíncrona.
- ActiveModel
    - Proporciona funcionalidades comunes a los modelos (validaciones, callbacks, serialización), incluso para clases que no se guardan en la base de datos.
    - Ejemplo: usar validaciones sin una base de datos.
- ActiveRecord
    - Es el ORM (Object Relational Mapper) de Rails. Permite interactuar con la base de datos usando objetos Ruby en lugar de SQL.
    - Ejemplo: crear y guardar un usuario.
- ActiveSupport
    - Proporcionar extensiones de lenguaje Ruby, utilidades y otras cosas transversales (tiempo, fechas, logging, inflectores, etc.).
    - Ejemplo: uso de helpers de tiempo.
- ActionCable
    - Agrega soporte para WebSockets, permitiendo comunicación en tiempo real entre el servidor y el cliente (por ejemplo, chats o notificaciones instantáneas).
    - Ejemplo: un chat en vivo donde los mensajes se actualizan sin recargar la página.
- ActiveStorage
    - Permite subir y asociar archivos (imágenes, documentos, etc.) a los modelos, y los guarda en servicios locales o en la nube (Amazon S3, Google Cloud, etc.).
    - Ejemplo: adjuntar una foto de perfil a un usuario.
- ActionText
    - Proporciona un editor de texto enriquecido (Trix) y permite guardar contenido con formato (negritas, imágenes embebidas, enlaces, etc.) directamente en el modelo.
    - Ejemplo: un modelo de artículo con cuerpo de texto enriquecido.
- ActionMailbox

- Permite que la aplicación **reciba correos electrónicos** directamente, procesándolos como si fueran solicitudes entrantes (por ejemplo, crear un comentario al recibir un mail).
- Ejemplo: recibir correos a replies@miapp.com y convertirlos en comentarios.

2. Investigá cómo se crea una aplicación Rails nueva y enumerá los pasos básico para tener la aplicación funcionando con una base de datos SQLite.

   El comando principal es rails new nombre_app, este por defecto crea la estructura de carpetas del proyecto, configura SQLite3 como base de datos e instala las dependencias necesarias usando Bundler.
   Se puede personalizar cómo se genera la app. Algunas opciones comunes son:

   - --database=sqlite3 : define la base de datos (por defecto es SQLite3, pero puede ser mysql, postgresql, etc.)
   - --skip-test: omite la carpeta de tests
   - --api: crea una aplicación tipo API (sin vistas)
   - --javascript=esbuild: define el bundler de JavaScript
   - --css=tailwind: agrega Tailwind CSS
   - --skip-git: no inicializa un repositorio Git

   Rails crea por defecto el archivo config/database.yml, donde ya está configurado SQLite3 para los entornos de development, test y production. La base se puede crear con rails db:create.

   Para ejecutar la aplicación se usa rails server o rails s.

3. Siguiendo los pasos que enumeraste en el punto anterior, creá una nueva aplicación Rails llamada practica_rails en la cual vas a realizar las pruebas para los ejercicios de esta práctica.
   1. ¿Qué estructura de directorios y archivos se generó?
      Tuve que modificar la version de ruby en el Gemfile y en .ruby-version

```
vboxuser@Ubuntu:~/practica_rails$ ls -la
total 92
drwxrwxr-x 13 vboxuser vboxuser 4096 oct 30 22:46 .
drwxr-x--- 25 vboxuser vboxuser 4096 oct 30 22:40 ..
drwxrwxr-x 11 vboxuser vboxuser 4096 oct 30 22:40 app
drwxr-xr-x  2 vboxuser vboxuser 4096 oct 30 22:40 bin
drwxrwxr-x  5 vboxuser vboxuser 4096 oct 30 22:40 config
-rw-rw-r--  1 vboxuser vboxuser  160 oct 30 22:40 config.ru
drwxrwxr-x  2 vboxuser vboxuser 4096 oct 30 22:40 db
-rw-rw-r--  1 vboxuser vboxuser 2004 oct 30 22:44 Gemfile
-rw-rw-r--  1 vboxuser vboxuser 6348 oct 30 22:46 Gemfile.lock
drwxrwxr-x  7 vboxuser vboxuser 4096 oct 30 22:40 .git
-rw-rw-r--  1 vboxuser vboxuser  327 oct 30 22:40 .gitattributes
-rw-rw-r--  1 vboxuser vboxuser  744 oct 30 22:40 .gitignore
drwxrwxr-x  4 vboxuser vboxuser 4096 oct 30 22:40 lib
drwxrwxr-x  2 vboxuser vboxuser 4096 oct 30 22:40 log
-rw-rw-r--  1 vboxuser vboxuser  226 oct 30 22:40 package.json
drwxrwxr-x  2 vboxuser vboxuser 4096 oct 30 22:40 public
-rw-rw-r--  1 vboxuser vboxuser  227 oct 30 22:40 Rakefile
-rw-rw-r--  1 vboxuser vboxuser  374 oct 30 22:40 README.md
-rw-rw-r--  1 vboxuser vboxuser    6 oct 30 22:43 .ruby-version
drwxrwxr-x  2 vboxuser vboxuser 4096 oct 30 22:40 storage
drwxrwxr-x  5 vboxuser vboxuser 4096 oct 30 22:40 tmp
drwxrwxr-x  2 vboxuser vboxuser 4096 oct 30 22:40 vendor
```

practica_rails/

```
├── app
│   ├── assets
│   │   ├── images
│   │   └── stylesheets
│   │       └── application.css
│   ├── controllers
│   │   ├── application_controller.rb
│   │   └── concerns
│   ├── helpers
│   │   └── application_helper.rb
│   ├── javascript
│   │   ├── application.js
│   │   └── controllers
│   │       ├── application.js
│   │       ├── hello_controller.js
│   │       └── index.js
```

```
│   ├── jobs
│   │   └── application_job.rb
│   ├── mailers
│   │   └── application_mailer.rb
│   ├── models
│   │   ├── application_record.rb
│   │   └── concerns
│   └── views
│       ├── layouts
│       │   ├── application.html.erb
│       │   ├── mailer.html.erb
│       │   └── mailer.text.erb
│       └── pwa
│           ├── manifest.json.erb
│           └── service-worker.js
├── bin
│   ├── brakeman
│   ├── bundler-audit
│   ├── ci
│   ├── dev
│   ├── docker-entrypoint
│   ├── importmap
│   ├── jobs
│   ├── kamal
│   ├── rails
│   ├── rake
│   ├── rubocop
│   ├── setup
│   └── thrust
├── config
│   ├── application.rb
│   ├── boot.rb
│   ├── bundler-audit.yml
```

```
│   ├── cable.yml
│   ├── cache.yml
│   ├── ci.rb
│   ├── credentials.yml.enc
│   ├── database.yml
│   ├── deploy.yml
│   ├── environment.rb
│   ├── environments
│   │   ├── development.rb
│   │   ├── production.rb
│   │   └── test.rb
│   ├── importmap.rb
│   ├── initializers
│   │   ├── assets.rb
│   │   ├── content_security_policy.rb
│   │   ├── filter_parameter_logging.rb
│   │   └── inflections.rb
│   ├── locales
│   │   └── en.yml
│   ├── master.key
│   ├── puma.rb
│   ├── queue.yml
│   ├── recurring.yml
│   ├── routes.rb
│   └── storage.yml
├── config.ru
├── db
│   ├── cable_schema.rb
│   ├── cache_schema.rb
│   ├── queue_schema.rb
│   └── seeds.rb
├── Dockerfile
├── Gemfile
```

```
├── Gemfile.lock
├── lib
│   └── tasks
├── log
│   └── development.log
├── public
│   ├── 400.html
│   ├── 404.html
│   ├── 406-unsupported-browser.html
│   ├── 422.html
│   ├── 500.html
│   ├── icon.png
│   ├── icon.svg
│   └── robots.txt
├── Rakefile
├── README.md
├── script
├── storage
├── tmp
│   ├── cache
│   │   └── bootsnap
│   │       ├── compile-cache-iseq
│   │       │   ├─...
│   │       │   ├─...
│   │       └── load-path-cache
│   ├── pids
│   └── storage
└── vendor
    └── javascript
```

2. Investigá (abrí y leé) los archivos de configuración que se generaron. ¿Qué información importante contienen?

   Estos son algunos:

- config/: contiene toda la configuración del proyecto, acá se define cómo se inicializa, se conecta a la base de datos, los entornos, rutas y políticas del sistema.
- config/application.rb:
    - Es el núcleo de configuración del proyecto.
    - Carga Rails y todas las dependencias (require "rails/all").
    - Define la clase principal del proyecto (por ejemplo, module PracticaRails; class Application < Rails::Application).
    - Configura ajustes globales como:
        - Zona horaria (config.time_zone).
        - Idioma (config.i18n.default_locale).
        - Generadores automáticos (por ejemplo, si se crean tests o no).
        - Versiones de Rails para compatibilidad (config.load_defaults 6.1).
    - Importancia: Es el punto donde se centraliza toda la configuración general de la app.
- config/environment.rb
    - Se encarga de inicializar Rails con la configuración definida en application.rb.
    - Básicamente pone en marcha toda la aplicación Rails.
- config/environments/
    - Contiene configuraciones específicas por entorno:
        - development.rb: ajustes para el entorno de desarrollo (muestra errores, recarga código sin reiniciar el servidor, etc.).
        - production.rb: optimiza el rendimiento y la seguridad (cacheo, compresión, logs mínimos).
        - test.rb: configuración para ejecutar tests (usa base de datos de test, no muestra errores de usuario, etc.).
- config/database.yml
    - Define las credenciales de conexión a la base de datos.

    - Rails usa por defecto SQLite3, con secciones separadas para cada entorno.
- config/routes.rb
    - Define todas las rutas de la aplicación (qué URL apunta a qué controlador y acción).
    - Importancia: Es el mapa que conecta las URLs con el código Ruby que las maneja.
- config/credentials.yml.enc y config/master.key
    - credentials.yml.enc: contiene datos sensibles cifrados, como claves API o contraseñas.
    - master.key: es la clave para descifrarlo.
    - Nunca se deben subir al repositorio público.
    - Importancia: Permite guardar secretos del proyecto de forma segura.
- config/puma.rb
    - Configuración del servidor web Puma, que es el servidor por defecto de Rails.
    - Define número de hilos, workers, puerto, etc.
    - Importancia: Controla cómo se lanza la aplicación en modo servidor.
- config/initializers/
    - Scripts Ruby que se cargan al iniciar la app.
    - Algunos ejemplos:
        - assets.rb: configuración de precompilación de assets (CSS, JS, imágenes).
        - filter_parameter_logging.rb: oculta parámetros sensibles (como contraseñas) de los logs.
        - wrap_parameters.rb: cómo se reciben los parámetros en controladores.
        - cookies_serializer.rb: formato de las cookies serializadas.
    - Importancia: Permiten inicializar y ajustar comportamiento de librerías o del framework.
- config/locales/en.yml
    - Contiene los textos para traducciones (i18n).

    - Importancia: Hace que la app sea fácilmente traducible a otros idiomas.
- config/cable.yml
    - Configura Action Cable, el sistema de WebSockets de Rails (para chats, notificaciones en tiempo real).
    - Define el adaptador (por ejemplo, Redis o Async).
    - Importancia: Permite comunicación en tiempo real.
- config/storage.yml
    - Configura Active Storage, el sistema de manejo de archivos (fotos, documentos).
    - Define dónde se guardan: local, Amazon S3, Google Cloud, etc.
    - Importancia: Permite manejar archivos subidos en la app.
- config/spring.rb
    - Configuración de Spring, una herramienta que acelera la carga de Rails en desarrollo.
- Importancia: Mejora la productividad del desarrollador.

3. ¿Qué scripts encontrás en el directorio bin? ¿Para qué sirven?

- bin/rails
    - Es el script principal para ejecutar comandos de Rails.
    - Se usa para correr el servidor, generar controladores, modelos, ejecutar migraciones, etc.

```
bin/rails server     # Inicia el servidor
bin/rails console    # Abre la consola interactiva de Rails
bin/rails generate   # Genera modelos, controladores, vistas, etc.
```

    - Es el punto de entrada al framework ya que todas las tareas de Rails se inician desde aquí.
- bin/rake
    - Ejecuta tareas Rake (el sistema de automatización de tareas de Ruby).
    - Rails define muchas tareas rake, como:

```
bin/rake db:migrate    # Aplica migraciones de base de datos
bin/rake db:seed       # Carga datos iniciales
```

- Permite automatizar tareas repetitivas de desarrollo y administración.
- bin/setup
    - Script para configurar el entorno de desarrollo por primera vez.
    - Generalmente ejecuta:
        - Instalación de gemas (bundle install).
        - Creación y configuración de la base de datos (rails db:setup).
        - Instalación de dependencias JavaScript (yarn install).
    - Facilita preparar el proyecto rápidamente en una nueva máquina o entorno.

4. ¿Qué gemas se incluyeron por defecto en el archivo Gemfile? Investigá para qué sirve cada una de ellas.

cat Gemfile
source "https://rubygems.org"

Bundle edge Rails instead: gem "rails", github: "rails/rails", branch: "main"
gem "rails", "~> 8.1.1"
The modern asset pipeline for Rails [https://github.com/rails/propshaft]
gem "propshaft"
Use sqlite3 as the database for Active Record
gem "sqlite3", ">= 2.1"
Use the Puma web server [https://github.com/puma/puma]
gem "puma", ">= 5.0"
Use JavaScript with ESM import maps [https://github.com/rails/importmap-rails]
gem "importmap-rails"
Hotwire's SPA-like page accelerator [https://turbo.hotwired.dev]
gem "turbo-rails"
Hotwire's modest JavaScript framework [https://stimulus.hotwired.dev]
gem "stimulus-rails"

```
# Build JSON APIs with ease [https://github.com/rails/jbuilder]
gem "jbuilder"

# Use Active Model has_secure_password
[https://guides.rubyonrails.org/active_model_basics.html#securepassword]
# gem "bcrypt", "~> 3.1.7"

# Windows does not include zoneinfo files, so bundle the tzinfo-data gem
gem "tzinfo-data", platforms: %i[ windows jruby ]

# Use the database-backed adapters for Rails.cache, Active Job, and Action
Cable
gem "solid_cache"
gem "solid_queue"
gem "solid_cable"

# Reduces boot times through caching; required in config/boot.rb
gem "bootsnap", require: false

# Deploy this application anywhere as a Docker container [https://kamal-
deploy.org]
gem "kamal", require: false

# Add HTTP asset caching/compression and X-Sendfile acceleration to Puma
[https://github.com/basecamp/thruster/]
gem "thruster", require: false

# Use Active Storage variants
[https://guides.rubyonrails.org/active_storage_overview.html#transforming-
images]
gem "image_processing", "~> 1.2"

group :development, :test do
```

```
  # See
https://guides.rubyonrails.org/debugging_rails_applications.html#debugging-
with-the-debug-gem
  gem "debug", platforms: %i[ mri windows ], require: "debug/prelude"

  # Audits gems for known security defects (use config/bundler-audit.yml to
ignore issues)
  gem "bundler-audit", require: false

  # Static analysis for security vulnerabilities [https://brakemanscanner.org/]
  gem "brakeman", require: false

  # Omakase Ruby styling [https://github.com/rails/rubocop-rails-omakase/]
  gem "rubocop-rails-omakase", require: false
end

group :development do
  # Use console on exceptions pages [https://github.com/rails/web-console]
  gem "web-console"
end
```

Gema	**Función principal**
rails	Es el framework completo Ruby on Rails. Incluye módulos como ActiveRecord, ActionController, ActionView, etc.
propshaft	Es el nuevo asset pipeline moderno de Rails (reemplaza a Sprockets). Se encarga de gestionar y servir archivos estáticos (CSS, JS, imágenes).
sqlite3	Driver para conectar Rails con la base de datos SQLite (por defecto en desarrollo y pruebas).
puma	Servidor web rápido y concurrente para ejecutar aplicaciones Rails. Es el servidor por defecto desde Rails 5.
importmap-rails	Permite usar JavaScript en Rails sin necesidad de usar Webpack o Node. Usa import maps para cargar módulos JS directamente desde el navegador.

turbo-rails	Parte de Hotwire. Acelera la navegación en Rails sin usar frameworks SPA (actualiza solo partes del HTML en lugar de recargar toda la página).
stimulus-rails	También parte de Hotwire. Proporciona un microframework JS para agregar interactividad controlada por controladores Stimulus.
jbuilder	Facilita la creación de respuestas JSON desde Rails (útil para APIs o vistas híbridas).
tzinfo-data	Proporciona información de zonas horarias en Windows (ya que no incluye estos datos por defecto).
solid_cache , solid_queue, solid_cable	Nuevos componentes de Rails 8: • solid_cache: sistema de caché persistente en base de datos. • solid_queue: colas de trabajo en base de datos (reemplaza Sidekiq o Delayed Job en apps pequeñas). • solid_cable: backend para Action Cable usando la base de datos como transporte de mensajes.
bootsnap	Acelera el arranque de la aplicación mediante caché de bytecode y carga más eficiente de gemas.
kamal	Herramienta oficial de Rails para desplegar apps fácilmente en servidores o nubes mediante contenedores Docker.
thruster	Extiende Puma con soporte para compresión HTTP, caching y envío eficiente de archivos (X-Sendfile).
image_processing	Permite transformar imágenes (redimensionar, recortar, etc.) dentro de Active Storage usando librerías como mini_magick o vips .
debug	Herramienta de depuración (breakpoints, inspección de variables) integrada con Rails.
bundler-audit	Escanea las gemas instaladas y alerta sobre vulnerabilidades conocidas.
brakeman	Analiza el código de la aplicación Rails en busca de vulnerabilidades de seguridad estáticas.

rubocop-rails-omakase	Define un conjunto de reglas de estilo y buenas prácticas sugeridas por Rails (Omakase = “la elección del chef”).
web-console	Permite abrir una consola interactiva en la página de error del navegador para probar código Ruby en tiempo real.

4. ¿Qué es un ambiente (environment) en una aplicación Rails? ¿Qué sentido considerás que tiene usar diferentes ambientes en tu aplicación? ¿Cuál es el ambiente por defecto?

   Es un conjunto de configuraciones específicas que Rails utiliza para ejecutar la aplicación en un determinado contexto.

   Rails distingue principalmente entre tres ambientes predefinidos:

   - development – Desarrollo local, cuando el programador está escribiendo código.
   - test – Para ejecutar pruebas automatizadas sin afectar los datos reales.
   - production – Para la aplicación en un servidor real, accesible por los usuarios.

   Cada ambiente tiene su propia configuración, cuyos archivos se mencionaron en el punto anterior.
   Estas configuraciones incluyen cosas como:

   - Nivel de logging (qué información se muestra en la consola).
   - Uso de caching.
   - Manejo de errores y excepciones.
   - Conexión a la base de datos específica del entorno.

   Usar distintos ambientes permite que la misma aplicación se comporte de manera diferente según el contexto, sin tener que cambiar el código.

   Rails utiliza development como ambiente por defecto, para usar otro ambiente se puede especificar con la variable RAILS_ENV:

   ```
   rails server -e production
   ```

```
rails console -e test
```

5. Sobre la configuración de Rails:
    1. ¿De qué forma podés especificar parámetros de configuración del framework en una app Rails?

       De distintas formas:
       - Archivo config/application.rb: Configuración global de la aplicación que afecta a todos los ambientes.
       - Archivos por ambiente (config/environments/development.rb, production.rb, test.rb): Configuraciones específicas para cada ambiente.
       - Initializers (config/initializers/*.rb): Configuraciones y setups que se ejecutan al iniciar la aplicación.
       - Locales y ficheros YAML (config/locales/*.yml) para traducciones.

    2. ¿Cuáles son los archivos de configuración principales? Intentá relacionar este concepto con los ambientes que antes viste.

       - config/application.rb: configuración global.
       - config/environments/*.rb: configuración específica por ambiente (development, test, production).
       - config/database.yml: configuración de bases de datos según ambiente.
       - config/initializers/*.rb: código que se ejecuta al iniciar la app.

    3. Modificá el locale por defecto de tu aplicación para que sea español.

       En config/application.rb se debe agregar config.i18n.default_locale = :es

    4. Modificá la zona horaria de tu aplicación para que sea la de la Argentina (GMT3).

       En config/application.rb se debe agregar config.time_zone = "Buenos Aires"

       ```
       module PracticaRails
         class Application < Rails::Application
       ```

```ruby
    # Initialize configuration defaults for originally
generated Rails version.
    config.load_defaults 8.1

    # Please, add to the `ignore` list any other `lib`
subdirectories that do
    # not contain `.rb` files, or that should not be reloaded
or eager loaded.
    # Common ones are `templates`, `generators`, or
`middleware`, for example.
    config.autoload_lib(ignore: %w[assets tasks])

    # Configuration for the application, engines, and
railties goes here.
    #
    # These settings can be overridden in specific
environments using the files
    # in config/environments, which are processed later.
    #
    # config.time_zone = "Central Time (US & Canada)"
    # config.eager_load_paths << Rails.root.join("extras")

    # Don't generate system test files.
    config.generators.system_tests = nil
    config.i18n.default_locale = :es
    config.time_zone = "Buenos Aires"
  end
end
```

6. Sobre los initializers:
    1. ¿Qué son y para qué se utilizan?

    Son archivos .rb en config/initializers que permiten configurar librerías externas, inicializar variables globales o constantes y ejecutar código que debe correrse al arrancar la aplicación.

2. ¿En qué momento de la vida de la aplicación se ejecutan?

   Se ejecutan cuando la aplicación arranca en el boot del servidor Rails (rails server) o cuando se carga la consola (rails console).

3. Si tu app está corriendo y modificás algún initializer, ¿los cambios se reflejan automáticamente? ¿Por qué?

   No automáticamente en producción, porque Rails carga la app solo al inicio. En desarrollo, con reload_classes habilitado, algunos cambios pueden reflejarse, pero lo más seguro es reiniciar el servidor

4. Creá un initializer en tu aplicación que imprima en pantalla el string "Booting practica_rails :)". ¿En qué momento se visualiza este mensaje?

   vboxuser@Ubuntu:~/practica_rails/config/initializers$ rails server
   => Booting Puma
   => Rails 8.1.1 application starting in development
   => Run `bin/rails server --help` for more startup options
   Booting practica_rails :)
   Puma starting in single mode...
   * Puma version: 7.1.0 ("Neon Witch")
   * Ruby version: ruby 3.3.5 (2024-09-03 revision ef084cc8f4) [x86_64-linux]
   * Min threads: 3
   * Max threads: 3
   * Environment: development
   *   PID: 11392
   * Listening on http://127.0.0.1:3000
   * Listening on http://[::1]:3000
   Use Ctrl-C to stop
   ^C- Gracefully stopping, waiting for requests to finish
   === puma shutdown: 2025-10-31 18:31:15 +0000 ===
   - Goodbye!
   Exiting

vboxuser@Ubuntu:~/practica_rails/config/initializers$ cat boot_message.rb
puts "Booting practica_rails :)"

Se visualiza apenas se levanta la aplicacion

7. Sobre los generators:
   1. ¿Qué son? ¿Qué beneficios imaginás que trae su uso?

      Son comandos de Rails que crean código boilerplate (controladores, modelos, vistas, tests, etc.). Estos permiten ahorro de tiempo y mantienen consistencia de convenciones.

   2. ¿Con qué comando podés consultar todos los generators disponibles en tu app Rails?

      Con rails generate o rails g

   3. Utilizando el generator adecuado, creá un controlador llamado PoliteController que tenga una acción salute que responda con un saludo aleatorio de entre un arreglo de 5 diferentes, como por ejemplo “Good day sir/ma’am.”.

```
rails generate controller Polite salute
```

```ruby
class PoliteController < ApplicationController
  def salute
    greetings = [
      "Good day sir/ma’am.",
      "Hello there!",
      "Greetings!",
      "Hi, how are you?",
      "Salutations!"
    ]
    render plain: greetings.sample
```

```
  end
end
```

8. Sobre routing:
    1. ¿Dónde se definen las rutas de la app Rails?

       En config/routes.rb

    2. ¿De qué formas se pueden definir las rutas? Investigá la DSL (domain specific language) para definición de rutas que Ruby on Rails provee.

       Algunas formas comunes son:

       - Ruta simple
           - get "welcome/index"
               - Mapea una solicitud GET a la acción index del controlador WelcomeController.
       - Especificando controlador y acción
           - get "about", to: "pages#about"
               - pages#about significa controlador PagesController, acción about.
           - post "users", to: "users#create"
       - Rutas con parámetros
           - get "products/:id", to: "products#show"
               - :id es un parámetro dinámico (por ejemplo /products/5).
       - Rutas RESTful (recurso completo)
           - resources :articles
               - Crea automáticamente todas las rutas CRUD:
                   - GET /articles → articles#index
                   - GET /articles/:id → articles#show
                   - POST /articles → articles#create
                   - PATCH /articles/:id → articles#update
                   - DELETE /articles/:id → articles#destroy
       - Varias acciones específicas

- resources :posts, only: [:index, :show]
- resources :comments, except: [:destroy]

- Rutas raíz y nombres
  - root "home#index"
    - root define la página principal.
  - get "contact", to: "pages#contact", as: "contact_us"
    - as: permite generar helpers como contact_us_path.
- Rutas anidadas
  - resources :articles do
    resources :comments
    end

3. ¿Qué ruta(s) agregó el generator que usaste en el último inciso del punto anterior?

   route get "polite/salute"

4. ¿Con qué comando podés consultar todas las rutas definidas en tu aplicación Rails?

   Con rails routes

5. Modificá el mapeo de rutas de tu aplicación para que al acceder a / (root) se acceda al controlador definido antes (polite#salute).

   En el archivo config/routes.rb se agrega root "polite#salute"

9. ¿De qué forma extiende AS las clases básicas de Ruby para incorporar nuevos métodos?

   Lo hace mediante “monkey patching”, reabriendo las clases básicas de Ruby y definiendo nuevos métodos.
   Por ejemplo, dentro de activesupport/core_ext/string/inflections.rb, hay algo como:

```ruby
class String
  def pluralize
    ActiveSupport::Inflector.pluralize(self)
  end
end
```

10. Investigá qué métodos agrega AS a las siguientes clases:
    1. String

       https://api.rubyonrails.org/classes/String.html

Método	Ejemplo	Resultado
camelize	"user_profile".camelize	"UserProfile"
underscore	"UserProfile".underscore	"user_profile"
titleize	"hola mundo".titleize	"Hola Mundo"
pluralize	"person".pluralize	"people"
singularize	"people".singularize	"person"
truncate	"Hola mundo".truncate(5)	"Ho..."
parameterize	"Hola Mundo!".parameterize	"hola-mundo"
starts_with? / ends_with?	"ruby".starts_with?("ru")	true

    2. Array

       https://api.rubyonrails.org/classes/Array.html

Método	Ejemplo	Resultado
to_sentence	["Rojo", "Verde", "Azul"].to_sentence	"Rojo, Verde y Azul"
to_param	[1,2,3].to_param	"1/2/3"
extract_options!	[1, {a: 2}].extract_options!	{a: 2}
second, third, fourth, fifth	[10, 20, 30].second	20
from / to	(1..10).to_a.from(3)	[4,5,6,7,8,9,10]

    3. Hash

       https://api.rubyonrails.org/classes/Hash.html

Método	Ejemplo	Resultado
except	{a:1,b:2}.except(:a)	{b:2}
slice	{a:1,b:2,c:3}.slice(:a,:c)	{a:1,c:3}
stringify_keys	{a:1}.stringify_keys	{"a"=>1}
symbolize_keys	{"a"=>1}.symbolize_keys	{a:1}
deep_merge	{a:{b:1}}.deep_merge({a:{c:2}})	{a:{b:1,c:2}}
deep_transform_keys	{a:{b:1}}.deep_transform_keys(&:to_s)	{"a"=>{"b"=>1}}

4. Date

   https://api.rubyonrails.org/classes/Date.html

Método	Ejemplo	Resultado
beginning_of_day	Date.today.beginning_of_day	2025-10-31 00:00:00
end_of_day	Date.today.end_of_day	2025-10-31 23:59:59
yesterday, tomorrow	Date.tomorrow	2025-11-01
next_week, next_month	Date.today.next_month	2025-11-30
ago / since	3.days.ago	Fecha/hora de hace 3 días
all_day	Date.today.all_day	2025-10-31 00:00:00..23:59:59

5. Numeric

   https://api.rubyonrails.org/classes/Numeric.html

Método	Ejemplo	Resultado
minutes, hours, days, weeks, months, years	2.days	172800 segundos
ago / from_now	5.hours.ago	Tiempo actual - 5 horas
in_milliseconds	1.5.seconds.in_milliseconds	1500.0
kilobytes, megabytes, gigabytes	5.megabytes	5242880
ordinalize	3.ordinalize	"3rd"

11. ¿Qué hacen y en qué clase define AS los siguientes métodos?

    https://guides.rubyonrails.org/active_support_core_extensions.html

    1. blank?

       active_support/core_ext/object/blank.rb.

       Devuelve true si el objeto está “vacío” o “sin valor útil”.

    2. present?

       active_support/core_ext/object/blank.rb.

       Es el opuesto de blank?. Devuelve true si el objeto no está en blanco.

    3. presence

       active_support/core_ext/object/blank.rb.

       Devuelve el objeto si está presente, o nil si está en blanco.

    4. try

       active_support/core_ext/object/try.rb.

       Llama a un método solo si el objeto no es nil, evitando errores tipo NoMethodError.

    5. in?

       active_support/core_ext/object/inclusion.rb.

       Devuelve true si el objeto está incluido dentro de otro enumerable.

6. alias_method_chain

   active_support/core_ext/module/aliasing.rb

   Antes se usaba para envolver un método existente con otro (una especie de "decorador" interno). Ya está deprecado en Rails 5. Se reemplaza por Module#prepend

7. delegate

   active_support/core_ext/module/delegation.rb

   Permite que una clase delegue métodos a un atributo interno.

8. pluralize

   active_support/core_ext/string/inflections.rb

   Devuelve el plural de una palabra.

9. singularize

   active_support/core_ext/string/inflections.rb

   Devuelve el singular de una palabra.

10. camelize

    active_support/core_ext/string/inflections.rb

    Convierte una cadena en CamelCase

11. underscore

    active_support/core_ext/string/inflections.rb

Convierte una cadena a snake_case

12. classify

    active_support/core_ext/string/inflections.rb

    Convierte un nombre de tabla (plural, tipo users) al nombre de clase (User)

13. constantize

    active_support/core_ext/string/inflections.rb

    Convierte una cadena en la constante (clase o módulo) que representa.

14. safe_constantize

    active_support/core_ext/string/inflections.rb

    Hace lo mismo que constantize, pero devuelve nil si la constante no existe (no lanza error).

15. humanize

    active_support/core_ext/string/inflections.rb.

    Convierte nombres técnicos en texto legible por humanos.

16. sum

    active_support/core_ext/enumerable.rb.

    Suma los elementos de una colección, con bloque opcional.

17. with_indifferent_access

active_support/core_ext/hash/indifferent_access.rb.

Devuelve una copia del hash que permite acceder a las claves como símbolos o strings indistintamente.

12. ¿De qué manera se le puede enseñar a AS cómo pasar de singular a plural (o viceversa) los sustantivos que usamos en nuestro código?

    En config/initializers/inflections.rb se pueden:

    1. Agregar reglas de pluralización

    ```
    inflect.plural /^(ox)$/i, '\1en'   # ox → oxen
    inflect.singular /^(ox)en/i, '\1'  # oxen → ox
    ```

    2. Agregar una palabra irregular

    ```
    inflect.irregular 'person', 'people'
    inflect.irregular 'mouse', 'mice'
    ```

    3. Agregar palabras que no cambian (incontables)

    ```
    inflect.uncountable %w( fish sheep equipment )
    ```

    4. Agregar propias reglas o nombres técnicos

    ```
    inflect.irregular 'datum', 'data'
    ```

13. ¿Cómo se define un modelo con ActiveRecord? ¿Qué requisitos tienen que cumplir las clases para utilizar la lógica de abstracción del acceso a la base de datos que esta librería ofrece?

    Se define como una clase Ruby que hereda de ApplicationRecord, que a su vez hereda de ActiveRecord::Base. El nombre de la clase va en singular y CamelCase, asi Rails buscará automáticamente una tabla plural y en snake_case en la base de datos. Adicionalmente, cada tabla debe tener una columna id (clave primaria).

Tambien se debe ejecutar la migración (rails db:migrate) para que las columnas definidas en la tabla se convierten en atributos del modelo automáticamente.

14. ¿De qué forma sabe ActiveRecord qué campos tiene un modelo?

    ActiveRecord lee la estructura actual de la tabla (después de correr las migraciones) para saber qué columnas existen y generar automáticamente los métodos del modelo. Genera automáticamente métodos de acceso (getters y setters) para cada columna, sin necesidad de declararlos en la clase.

15. ¿Qué metodos (getters y setters) genera AR para los campos escalares básicos (integer, string, text, boolean, float)?

    Se generan automáticamente getters y setters con el mismo nombre que la columna.

16. ¿Qué convención define AR para inferir los nombres de las tablas a partir de las clases Ruby? Citá ejemplos.

    Sigue una convención de “nombres en plural y snake_case” para inferir la tabla asociada a un modelo Ruby:

    - User → users
    - Person → peopl
    - BlogPost → blog_posts

17. Sobre las migraciones de AR:
    1. ¿Qué son y para qué sirven?

       Son clases Ruby que permiten definir cambios en la base de datos (crear/editar tablas, columnas, índices, etc.) de forma versionada y reproducible. Sirven para mantener sincronizada la estructura de la DB con el código, y para que otros desarrolladores puedan aplicar los mismos cambios fácilmente.

2. ¿Cómo se generan?

   Con generadores
   rails generate migration NombreDeLaMigracion

3. ¿Dónde se guardan en el proyecto?

   Se guardan en db/migrate. Cada archivo representa un cambio incremental en la base de datos.

4. ¿Qué formato/organización tienen sus nombres de archivo? ¿Por qué considerás que es necesario respetar ese formato?

   Rails prefija cada migración con un timestamp (formato YYYYMMDDHHMMSS) para mantener el orden cronológico.

   Es necesario respetar este formato ya que Rails usa el timestamp para saber qué migraciones se aplicaron (schema_migrations) permitiendo aplicar migraciones en el orden correcto y evitar conflictos.

5. Generá una migración que cree la tabla offices con los siguientes atributos:
   - name: string de 255 caracteres, no puede ser nulo.
   - phone_number: string de 30 caracteres, no puede ser nulo y debe ser único.
   - address: text.
   - available: boolean, por defecto true, no puede ser nulo.

```
vboxuser@Ubuntu:~/practica_rails$ rails generate migration
CreateOffices
Booting practica_rails :)
      invoke  active_record
      create    db/migrate/20251031200218_create_offices.rb
vboxuser@Ubuntu:~/practica_rails$ code
db/migrate/20251031200218_create_offices.rb
```

```ruby
class CreateOffices < ActiveRecord::Migration[8.1]
  def change
    create_table :offices do |t|
      t.string :name, null: false, limit: 255
      t.string :phone_number, null: false, limit: 30
      t.text :address
      t.boolean :available, null: false, default: true

      t.timestamps
    end

    add_index :offices, :phone_number, unique: true
  end
end
```

18. Creá el modelo Office para la tabla offices que creaste antes, e implementá en éste el método de instancia to_s.

```
rails generate model Office --skip-migration
```

```ruby
class Office < ApplicationRecord
  # Método de instancia que devuelve una representación como string
  def to_s
    "Office: #{name}, Phone: #{phone_number}, Available: #{available}, Address: #{address}"
  end
end
```

19. Utilizando migraciones, creá la tabla y el modelo para la clase Employee, con la siguiente estructura:

- name: string de 150 caracteres, no puede ser nulo.
- e_number: integer, no puede ser nulo, debe ser único.
- office_id: integer, foreign key hacia offices.

```
vboxuser@Ubuntu:~/practica_rails$ rails generate model
Employee
Booting practica_rails :)
      invoke  active_record
      create    db/migrate/20251031201417_create_employees.rb
      create    app/models/employee.rb
vboxuser@Ubuntu:~/practica_rails$ code
db/migrate/20251031201417_create_employees.rb
```

```ruby
class CreateEmployees < ActiveRecord::Migration[8.1]
  def change
    create_table :employees do |t|
      t.string :name, null: false, limit: 150
      t.integer :e_number, null: false
      t.references :office, foreign_key: true

      t.timestamps
    end

    add_index :employees, :e_number, unique: true
  end
end
```

20. Agregá una asociación entre Employee y Office acorde a la columna office_id que posee la tabla employees.

    1. ¿Qué tipo de asociación declaraste en la clase Employee?

       belongs_to :office

2. ¿Y en la clase Office?

   has_many :employees, dependent: :destroy

3. ¿Qué métodos generó AR en el modelo a partir de esto?

   employee.office # devuelve la oficina asociada
   employee.office = o # asigna la oficina
   employee.build_office # crea un nuevo objeto Office asociado
   employee.create_office # crea y guarda un Office asociado
   office.employees # devuelve todos los empleados
   office.employees << e # agrega un empleado a la oficina
   office.employees.create(...) # crea y asocia un empleado
   office.employees.build(...) # construye un empleado asociado sin guardar
   office.employees.destroy(e) # elimina la asociación (y como hay dependent: :destroy, elimina el empleado)

21. Sobre scopes:
    1. ¿Qué son los scopes de AR? ¿Para qué los utilizarías?

       Es un filtro reutilizable que se define en un modelo y devuelve un ActiveRecord::Relation. Sirve para encapsular consultas frecuentes de manera legible y reutilizable.

    2. Investigá qué diferencia principal existe entre un método de clase y un scope, cuando se los utiliza para realizar la misma tarea.

Aspecto	**Método de clase**	**Scope**
Retorna	Puede retornar cualquier cosa	Siempre devuelve un ActiveRecord::Relation
Composición	No se puede encadenar directamente con otros scopes	Se puede encadenar (Employee.vacant.order(:name))

Sintaxis	def self.vacant; ...; end	scope :vacant, -> { ... }

3. Agregá los siguientes scopes al modelo Employee:
   - vacant: Filtra los empleados para quedarse únicamente con aquellos que no tengan una oficina asignada (o asociada).
     - `scope :vacant, -> { where(office_id: nil) }`
   - occupied: Complemento del anterior, devuelve los empleados que sí tengan una oficina asignada.
     - `scope :occupied, -> { where.not(office_id: nil) }`
4. Agregá este scope al modelo Office:
   - empty: Devuelve las oficinas que están disponibles (available == true) que no tienen empleados asignados.
     - `scope :empty, -> { where(available: true).left_outer_joins(:employees).where(employees: { id: nil }) }`

22. Sobre scaffold controllers:
    1. ¿Qué son? Al generarlos, ¿qué operaciones implementan sobre un modelo? ¿Pueden extenderse o modificarse?

       Un scaffold genera de forma automática un CRUD completo para un modelo: index, show, new, create, edit, update, destroy.
       Si se pueden extender y modificarse.

    2. ¿Con qué comando se generan?

       rails generate scaffold

    3. Utilizando el generator que indicaste en el inciso anterior, generá un controlador de scaffold para el modelo Office y otro para el modelo Employee.

rails generate scaffold_controller (evita generar migraciones y modelo, solo genera controlador y vistas)

4. ¿Qué rutas agregó este generator?

5. Analizá el código que se generó para los controladores y para las vistas, y luego modifícalo para que no permita el borrado de ninguno de los elementos. Enumerá los cambios que debiste hacer para que:
   - Las vistas no muestren la opción de borrar.
     - Borre todos los botones que decían Destroy en las vistas.
   - Los controladores no tenga la acción destroy.
     - Borre la acción destroy y el before action
   - Las rutas de borrado dejen de existir en la aplicación.
     - En config/routes.rb excluyo destroy
       - resources :offices, except: [:destroy]
       - resources :employees, except: [:destroy]
6. Modificá la vista de detalle (show) de una oficina para que, además de la información de la misma que ya presenta, muestre un listado con los empleados que tenga asociados en el cual cada nombre de empleado sea un link a su vista de detalle (employees#show).

```
<div id="<%= dom_id office %>">
  <p>
    <strong>Nombre:</strong>
    <%= office.name %>
  </p>

  <p>
    <strong>Teléfono:</strong>
    <%= office.phone_number %>
```

```
  </p>

  <p>
    <strong>Dirección:</strong>
    <%= office.address %>
  </p>

  <p>
    <strong>Disponible:</strong>
    <%= office.available ? "Sí" : "No" %>
  </p>

  <h3>Empleados</h3>
  <% if office.employees.any? %>
    <ul>
      <% office.employees.each do |employee| %>
        <li><%= link_to employee.name, employee_path(employee)
%> - E-Number: <%= employee.e_number %></li>
      <% end %>
    </ul>
  <% else %>
    <p><em>No hay empleados asignados a esta oficina</em></p>
  <% end %>
</div>
```

Eso esta en _office.html.erb que se renderiza dentro de show.html.erb

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

<%= render @office %>

<div>
```

```
  <%= link_to "Edit this office", edit_office_path(@office) %>
|
  <%= link_to "Back to offices", offices_path %>
</div>
```

23. ¿Qué son los validadores de AM? ¿Cuáles son los validadores básicos que provee esta librería?

    Son mecanismos que proporciona la librería ActiveModel (y que también usa ActiveRecord) para asegurar que los datos de un objeto cumplan ciertas reglas o restricciones antes de guardarlos en la base de datos.

Validador	Uso	Ejemplo
presence	Verifica que el valor no esté vacío o nulo	validates :nombre, presence: true
uniqueness	Verifica que el valor sea único en la base de datos	validates :email, uniqueness: true
length	Controla la longitud mínima o máxima	validates :nombre, length: { maximum: 50 }
numericality	Verifica que sea un número y permite opciones (solo enteros, mayor que, etc.)	validates :edad, numericality: { only_integer: true, greater_than: 0 }
format	Valida que el valor coincida con una expresión regular	validates :email, format: { with: URI::MailTo::EMAIL_REGEXP }
inclusion	Verifica que el valor esté dentro de una lista	validates :rol, inclusion: { in: %w(admin empleado) }
exclusion	Verifica que el valor **no** esté dentro de una lista	validates :nombre, exclusion: { in: %w(root admin) }
confirmation	Requiere que haya un campo con _confirmation y que coincidan	validates :password, confirmation: true

24. Agregá a los modelos Office y Employee las validaciones necesarias para hacer que sus atributos cumplan las restricciones definidas para las columnas de la tabla que cada uno representa.

```ruby
class Office < ApplicationRecord
  has_many :employees, dependent: :destroy
  scope :empty, -> { where(available:
true).left_outer_joins(:employees).where(employees: { id: nil })
}

  validates :name, presence: true, length: { maximum: 255 }
  validates :phone_number, presence: true, uniqueness: true,
length: { maximum: 30 }
  validates :address, length: { maximum: 1000 }, allow_blank:
true
  validates :available, inclusion: { in: [true, false] } # Si no
puede ser null no tiene sentido un valor por defecto

  def to_s
    "Office: #{name}, Phone: #{phone_number}, Available:
#{available}, Address: #{address}"
  end
end

class Employee < ApplicationRecord
  belongs_to :office
  scope :vacant, -> { where(office_id: nil) }
  scope :occupied, -> { where.not(office_id: nil) }

  validates :name, presence: true, length: { maximum: 150 }
  validates :e_number, presence: true, uniqueness: true,
numericality: { only_integer: true }
  validates :office_id, presence: true

  def to_s
```

```
    "Employee: #{name}, E-Number: #{e_number}, Office:
#{office.name}"
  end
end
```

25. Validadores personalizados:
    1. ¿Cómo podés definir uno para tus modelos AR?
       - Un archivo en app/validators/ y usarlo con validates_with.

```
class NombreValidator < ActiveModel::Validator
  def validate(record)
    if record.name =~ /^\d/
      record.errors.add(:name, "no puede comenzar con un
número")
    end
  end
end
```

Y en modelo

```
validates_with NombreValidator
```

https://api.rubyonrails.org/classes/ActiveModel/Validator.html

    2. Implementá un validador que chequee que un string sea un número telefónico con un formato válido para la Argentina.

The easiest way to add custom validators for validating individual attributes is with the convenient ActiveModel::EachValidator class.

```
class ArgentinePhoneValidator < ActiveModel::EachValidator
  ARG_PHONE_REGEX = /\A(\+54\s?9?\s?\d{2,4}\s?\d{3,4}-
?\d{4}|\d{2,4}\s?\d{3,4}-?\d{4})\z/
```

```ruby
  def validate_each(record, attribute, value)
    return if value.blank? || value.match?(ARG_PHONE_REGEX)

    record.errors.add(attribute, "no tiene un formato válido para un número argentino")
  end
end
```

3. Agregá el validador que definiste en el punto anterior a tu modelo Office para validar el campo phone_number.

```ruby
validates :phone_number, presence: true, uniqueness: true, length: { maximum: 30 }, argentine_phone: true
```

26. ¿A qué hacen referencia estos conceptos? (internacionalización (i18n) y localización (l10n))

Concepto	Significado	En qué consiste
Internacionalización (i18n)	Preparar la aplicación para soportar múltiples idiomas.	Estructurar el código para que sea adaptable (no contiene textos fijos).
Localización (l10n)	Adaptar la app a un idioma/región concreta.	Traducir textos, ajustar formatos, monedas, etc.

27. Investigá qué metodos provee la clase I18n para realizar la traducción (i18n) de texto y la localización (l10n) de valores.

Rails usa la clase I18n (de la gema i18n) para manejar traducciones mediante archivos YAML o JSON.
Los métodos principales son:

- I18n.t : traduce una clave según el idioma actual.

```ruby
I18n.t('greetings.hello')
# => "Hola"  (si el locale es :es)
```

```
I18n.t('greetings.hello', locale: :en)      # Traduce en inglés
I18n.t('greetings.hello', default: "Hi")    # Valor por defecto
I18n.t('greetings.user', name: "Agustina") # Interpolación: "Hola, Agustina"
```

- Ejemplo en config/locales/es.yml

```
es:
  greetings:
    hello: "Hola"
    user: "Hola, %{name}"
```

- I18n.available_locales: devuelve los idiomas disponibles en la app.

```
I18n.available_locales
# => [:en, :es, :fr]
```

- I18n.locale: devuelve o establece el idioma actual.

```
I18n.locale             # => :es
I18n.locale = :en       # Cambia el idioma a inglés
```

- Para formatear fechas, horas, números o monedas según la región/idioma, se usa el método I18n.l.

```
I18n.l(Date.today)
# => "5 de noviembre de 2025"  (si locale es :es)

I18n.l(Time.now, format: :short)
# => "05/11/25 19:30"

I18n.l(Time.now, format: :long, locale: :en)
# => "November 5, 2025 7:30 PM"
```

- Ejemplo en config/locales/es.yml

```
es:
  time:
    formats:
      short: "%d/%m/%y %H:%M"
```

```
      long: "%d de %B de %Y, %H:%M"
```

28. Modificá el controlador PoliteController que creaste antes para que utilice traducciones al imprimir el mensaje de saludo.
    1. Agregá las traducciones en el locale por defecto (inglés). ¿Dónde está ubicado ese archivo? ¿Qué convención debe seguir el nombre de los archivos de traducciones para que Rails sepa a qué locale corresponden?

       Está ubicado en config/locales/en.yml. La convención que debe seguir el nombre es <locale>.yml

```
en:
  hello: "Hello world"
  polite:
    greetings:
      - "Good day sir/ma'am."
      - "Hello there!"
      - "Greetings!"
      - "Hi, how are you?"
      - "Salutations!"
```

    2. Agregá un nuevo archivo de traducciones para el idioma español, y en él definí los mismos mensajes de traducción que en el inciso anterior, pero esta vez en español.

```
es:
  hello: "Hola"
  activerecord:
    models:
      office: "Oficina"
    attributes:
      office:
        name: "Nombre"
        phone_number: "Teléfono"
```

```
      address: "Dirección"
      available: "Disponible"
  polite:
    greetings:
      - "Buen día, señor/a."
      - "¡Hola!"
      - "Saludos!"
      - "¡Hola! ¿Cómo estás?"
      - "¡Muy buenas!"
```

3. Modificá la lógica de este controlador para que cambie el locale con que mostrará los mensajes internacionalizados en función del parámetro lang que reciba:
   - Si no recibe el parámetro, o el mismo no es un locale de los reconocidos por la aplicación (en o es), tomará por defecto el default locale de la aplicación (I18n.default_locale). Ejemplos de este caso: se accede a localhost:3000/, localhost:3000/?lang=fr, o localhost:3000/?lang=english.
   - Si recibe un locale válido, debe utilizarse ese locale para realizar la internacionalización de mensajes en la respuesta a ese request. Esto no debe modificar el locale por defecto de la aplicación. Ejemplos de este caso: se accede a localhost:3000/?lang=es, y localhost:3000/?lang=en.

```
class PoliteController < ApplicationController
  before_action :set_locale

  def salute
    greetings = I18n.t("polite.greetings")
    render plain: greetings.sample
  end

  private

  def set_locale
    lang = params[:lang]
```

```
      if I18n.available_locales.map(&:to_s).include?(lang)
        I18n.locale = lang
      else
        I18n.locale = I18n.default_locale
      end
    end
  end
```

29. Modificá los scaffold controllers que generaste para que utilicen i18n, tanto en las vistas como en los mensajes flash del controlador.

    Rails provee el helper t (abreviatura de I18n.t) para realizar traducciones directamente dentro de las vistas.

30. ¿Qué son los callbacks de controladores de AP? ¿Para qué los utilizarías?

    Son métodos que se ejecutan automáticamente antes, después o alrededor de una acción del controlador. Se usan para ejecutar lógica común que se necesita en varias acciones sin repetir código.

31. Tomando como base la lógica que implementaste en PoliteController para permitir que se especifique el locale a utilizar en la petición, refactorizá eso para que sea un callback de ese controlador.

    Ya lo tenía.

32. Refactorizá nuevamente el callback para que el mismo se ejecute también en el resto de los controladores de tu aplicación (los scaffold controllers en este caso), sin repetir el código en cada controlador. ¿Cómo hiciste eso?

    Para no repetir el callback en cada se mueve a la superclase ApplicationController, de la cual heredan todos los controladores (y se borra de PoliteController)

```ruby
class ApplicationController < ActionController::Base
  before_action :set_locale
  allow_browser versions: :modern
  stale_when_importmap_changes

  private

  def set_locale
    requested_locale = params[:lang]&.to_sym
    I18n.locale =
I18n.available_locales.include?(requested_locale) ?
requested_locale : I18n.default_locale
  end
end
```