

Modelo de Implementación (Uso de Patrones)



Equipo de Trabajo

Yulián Andrés Naranjo Cardona. 1.020.464.753

Luis Alejandro Zambrano Sánchez. 1.214.714.749

Pablo Andrés Díaz Gómez. 1.214.717.460

Estefanía Morales Araque. 1.152.438.479

Julián David Quiroz 1.152.443.697

Asesor

Diego Botia.

Universidad de Antioquia

Medellín (ANT)

2015- 1

Contenido

| | |
|--|-----------|
| 1. CALIDADES SISTÉMICAS VS REQUISITOS | 3 |
| Requisitos | 3 |
| Requisitos Operacionales..... | 3 |
| Requisitos Funcionales | 3 |
| Requisitos No Funcionales | 4 |
| Matriz | 4 |
| 2. PRUEBAS DE CONCEPTO EN EL FRAMEWORK | 4 |
| 3. ARQUITECTURA DEL FRAMEWORK..... | 5 |
| 3.2. Estrategia Para El Montaje De La Aplicación En El Framework..... | 6 |
| 3.3 Árbol De Directorios Del Framework Con El Contenido De Cada Carpeta..... | 11 |
| 3.4. Código De Ejemplo Del Montaje En Cada Capa..... | 14 |
| 3.5. Despliegue De La Aplicación Montada En El Framework | 15 |
| 4. PATRÓN POSA O J2EE | 18 |
| Patrón POSA..... | 18 |
| BIBLIOGRAFIA..... | 23 |

1. CALIDADES SISTÉMICAS VS REQUISITOS

La realización de este análisis se hará mediante la Metodología SUN-Tone AM, para esto se considerarán todos los requisitos, tanto funcionales como no funcionales que han sido identificados para nuestro caso de estudio y proyecto, cuya arquitectura se trabajará a lo largo de este escrito. Además, todos los requisitos listados se confronta con los atributos de calidad propuestos por Oracle en su metodología llamada: SUN-Tone AM - Sun Microsystems.

Requisitos

Requisitos Operacionales

1. El socio podrá tener una de las tres categorías manejadas por la aerolínea BinLaden.
2. Los socios podrán redimir las millas acumuladas por otros trayectos si tiene un mínimo de 50000
3. El canje de las millas para Vuelos Nacionales en temporada baja será de 50000 por trayecto y en temporada alta 80000 por trayecto.
4. El canje de las millas para Vuelos Internacionales en temporada baja será de 70000 y en temporada alta 90000 por trayecto (Destinos América).
5. Las temporadas bajas se manejan de Marzo a Mayo y de Septiembre a Noviembre.
6. Las temporadas altas se manejan de Diciembre a Febrero y de Junio a Agosto.

Requisitos Funcionales

7. Permitir el "CRUD" (create, read, update, delete) de los clientes que quieran ser socios de la "Aerolínea Binladen".
8. Realizar el "CRUD" básico en el manejo de datos personales de socios pertenecientes a la "Aerolínea Binladen"
9. Controlar la realización de la venta de tickets de los vuelos "Aerolínea Binladen".
10. Permitir la realización y consulta de reserva de tickets a socios de la "Aerolínea Binladen".
11. Permitir la realización de reserva de silletería para vuelos en la "Aerolínea Binladen".
12. Permitir a los socios adquirir los beneficios de pertenecer a la "Aerolínea Binladen".
13. Permitir a los socios consultar y canjear el número de millas que tiene acumulados hasta el momento, además de permitir comprar millas individuales.

Requisitos No Funcionales

14. Los agentes de viaje se deben de identificar en el sistema mediante un usuario y contraseña.
15. En caso de fallas de conexión con el sistema central, el sistema debe permitir operar normalmente 24/7, sincronizando posteriormente la información con el sistema central.
16. El sistema se deberá comunicar con el sistema financiero con el cual se tenga convenio de diferentes pasarelas de pago para comprar a crédito
17. La información de las reservas de los vuelos deberá estar disponible a las otras sucursales y servicios on-line en un corto plazo (2 horas).

Matriz

Matriz QoS vs Requisitos del sistema.

https://drive.google.com/open?id=1AE8Y-s3yHaXJNm6rtm_JZuuAdpObjj3WRetRy-dRnXk

Matriz Layers vs Tiers

<https://drive.google.com/open?id=15l7LqtNomi9Uuss0EkPSpyuyqOxw62UDEgGvkqAQPlc>

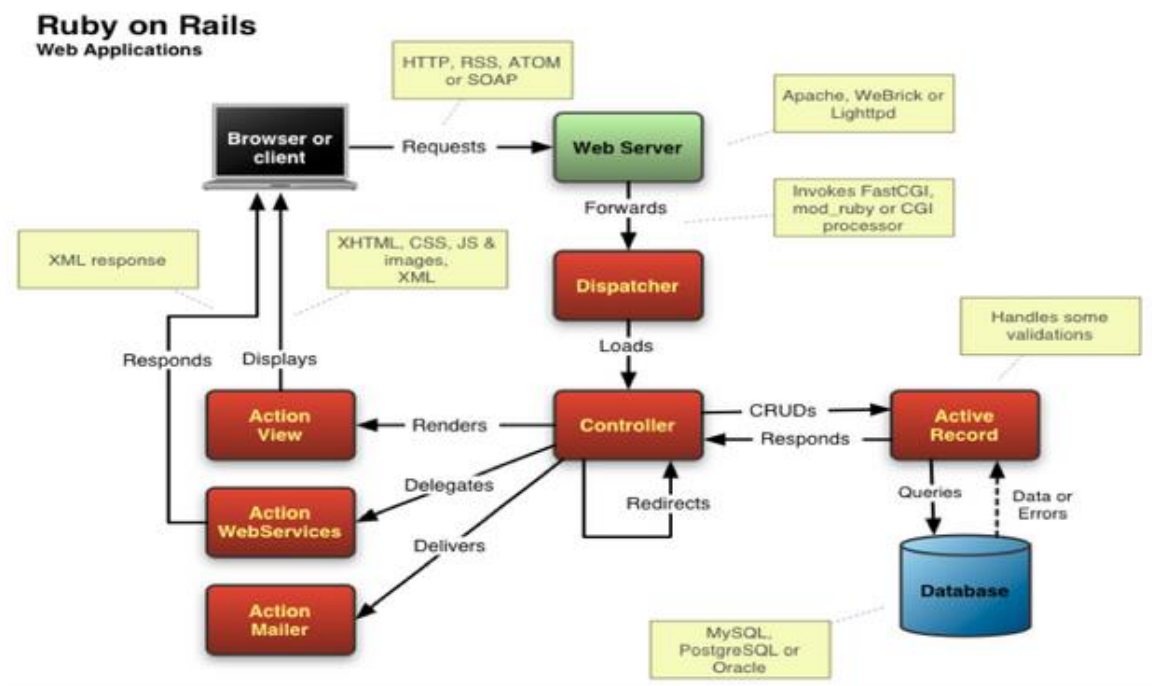
Matriz Layers vs QoS

https://drive.google.com/open?id=1iKhC_W0E9hWBph3r9qKSXGxmnPYwQOMZ1a9lZ-mxbKA

2. PRUEBAS DE CONCEPTO EN EL FRAMEWORK

Ver implementación del proyecto.

3. ARQUITECTURA DEL FRAMEWORK



MVC en Rails 4

Generalmente para interactuar con una aplicación de Rails, un navegador envía una solicitud, la cual es recibida por un servidor web y se transmite a un controlador de Rails, que es el encargado de definir qué hacer a continuación. El controlador interactúa con un modelo, que es un objeto de Ruby que representa un elemento de la página (por ejemplo, un usuario) y se encarga de comunicarse con la base de datos. Después de invocar el modelo, el controlador, genera la vista y devuelve la página web completa en el navegador como HTML.

Modelo:

- Acceso a los datos de la aplicación y reglas para manipularlos
- Módulo genérico ActiveRecord
- Modelos almacenados en BBDD relacionales ->ORM -> módulo ActiveRecord
- Una clase representa una tabla.
- Se descubren automáticamente los campos y la tabla (Alumno ! alumnos)
- Se pueden declarar relaciones con otros modelos/tablas
- Se puede personalizar y añadir métodos

Vista:

- Para decidir el aspecto
- Módulo: ActionView
- Plantillas en formato Embedded Ruby (por defecto)
- HTML con Ruby embebido
- Una por cada acción de cada controlador
- Corolario: dependen de los controladores
- HTML repetitivo, a funciones externas (helpers).

Controlador:

- «Bisagra» del MVC y lógica de cálculos
- Módulo: ActionController
- Cada controlador, una clase de Ruby
- Cada método, una acción
- «Andamios» (scaffold) para avanzar más rápido

Componentes de Rails 4

- **Action Pack** Gema que contiene tres componentes
- **Action Controller** parámetros HTTP, sesiones, redirecciones.
- **Action View** genera HTML o XML a partir de plantillas
- **Action Dispatch** enruta las peticiones dentro de la aplicación o a otra aplicación Rack
- **Action Mailer** para enviar y recibir correos electrónicos
- **Active Model** interfaz entre Action Pack y la capa de persistencia (ORM-Active Record)
- **Active Record** ORM, base de los modelos
- **Active Resource** permite utilizar recursos REST como objetos locales
- **Active Support** clases útiles y extensiones a las de Ruby
- **Railties** para crear aplicaciones y unir los demás componentes

3.2. Estrategia Para El Montaje De La Aplicación En El Framework

Primero que todo procederemos a crear nuestro proyecto y para esto crearemos un nuevo espacio de trabajo, colocando un nombre deseado y si se quiere una pequeña descripción del proyecto. Nosotros haremos el proyecto visible y trabajaremos bajo el template de Ruby On Rails, como lo podremos apreciar en la siguiente imagen.

Create a new workspace

Workspace name

Description

Already have your own development machine in the cloud? Create a new cloud9 workspace that connects to it [right here](#).

Hosted workspace



Private

This is a workspace for your eyes only



Public

This will create a workspace for everybody to see

Clone from Git or Mercurial URL (optional)

Choose a template



Custom



HTML5



Node.js



Meteor



PHP, Apache & MySQL

django

Python



Ruby



C++



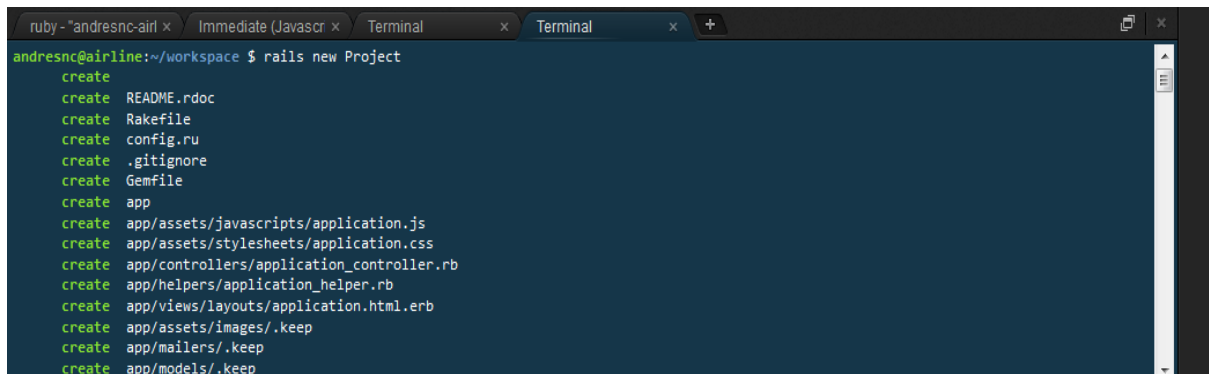
Wordpress



Ruby on Rails Tutorial

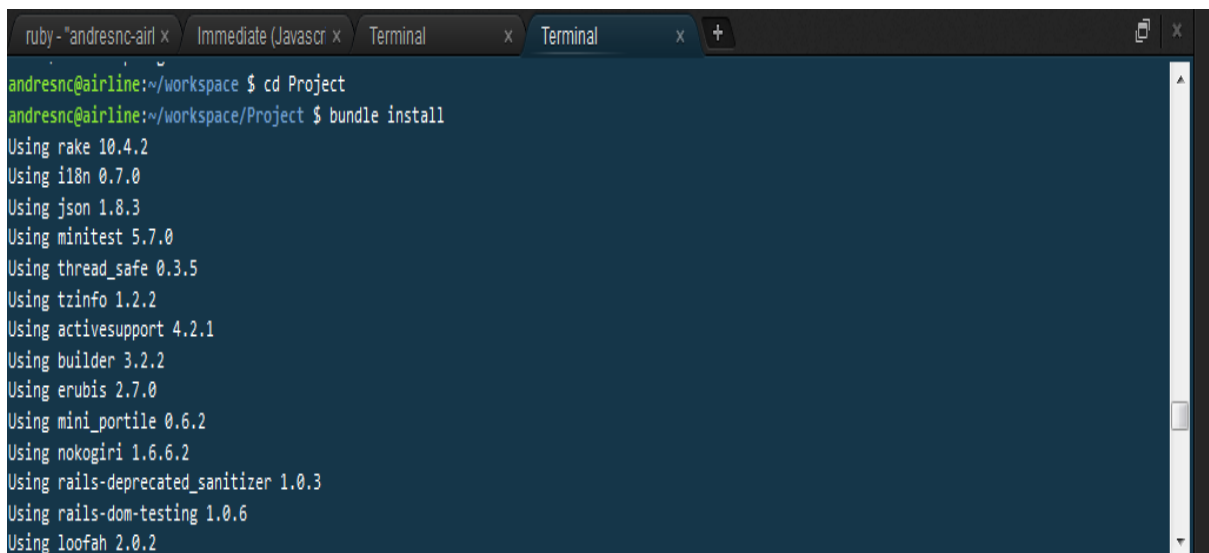
Create workspace

Después de crear el proyecto, nos dirigimos a la terminal y ejecutamos rails new project, el cual creara por defecto todos los archivos necesarios para desplegar nuestra aplicación en RoR tal como se muestra a continuación.

A terminal window with a dark blue background. The prompt is 'andresnc@airline:~/workspace'. The command 'rails new Project' has been executed. The output shows a list of files and directories being created, each preceded by the word 'create' in green. The files include README.rdoc, Rakefile, config.ru, .gitignore, Gemfile, app, app/assets/javascripts/application.js, app/assets/stylesheets/application.css, app/controllers/application_controller.rb, app/helpers/application_helper.rb, app/views/layouts/application.html.erb, app/assets/images/.keep, app/mailers/.keep, and app/models/.keep.

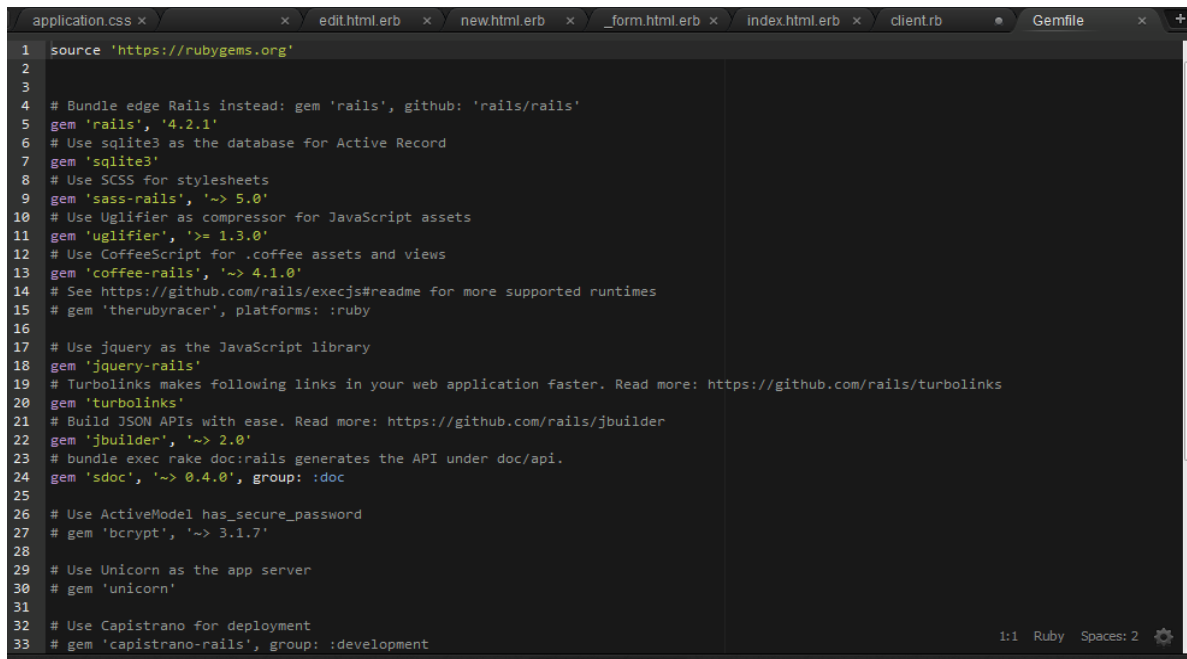
```
andresnc@airline:~/workspace $ rails new Project
create
create  README.rdoc
create  Rakefile
create  config.ru
create  .gitignore
create  Gemfile
create  app
create  app/assets/javascripts/application.js
create  app/assets/stylesheets/application.css
create  app/controllers/application_controller.rb
create  app/helpers/application_helper.rb
create  app/views/layouts/application.html.erb
create  app/assets/images/.keep
create  app/mailers/.keep
create  app/models/.keep
```

Después de la correcta realización de la creación del proyecto, de nuevo nos dirigimos a la terminal a realizar un paso sumamente importante y es el de instalar todas las gemas para el correcto funcionamiento del proyecto. Para esto únicamente nos ubicamos en el proyecto y corremos la instrucción bundle install, el cual por defecto instalara todas las gemas necesarias, tal como se muestra en la siguiente imagen.

A terminal window with a dark blue background. The prompt is 'andresnc@airline:~/workspace'. The command 'cd Project' has been executed, changing the directory to '~/workspace/Project'. The prompt is now 'andresnc@airline:~/workspace/Project'. The command 'bundle install' has been executed. The output shows a list of gems being installed, each preceded by the word 'Using' in green. The gems include rake 10.4.2, i18n 0.7.0, json 1.8.3, minitest 5.7.0, thread_safe 0.3.5, tzinfo 1.2.2, activesupport 4.2.1, builder 3.2.2, erubis 2.7.0, mini_portile 0.6.2, nokogiri 1.6.6.2, rails-deprecated_sanitizer 1.0.3, rails-dom-testing 1.0.6, and loofah 2.0.2.

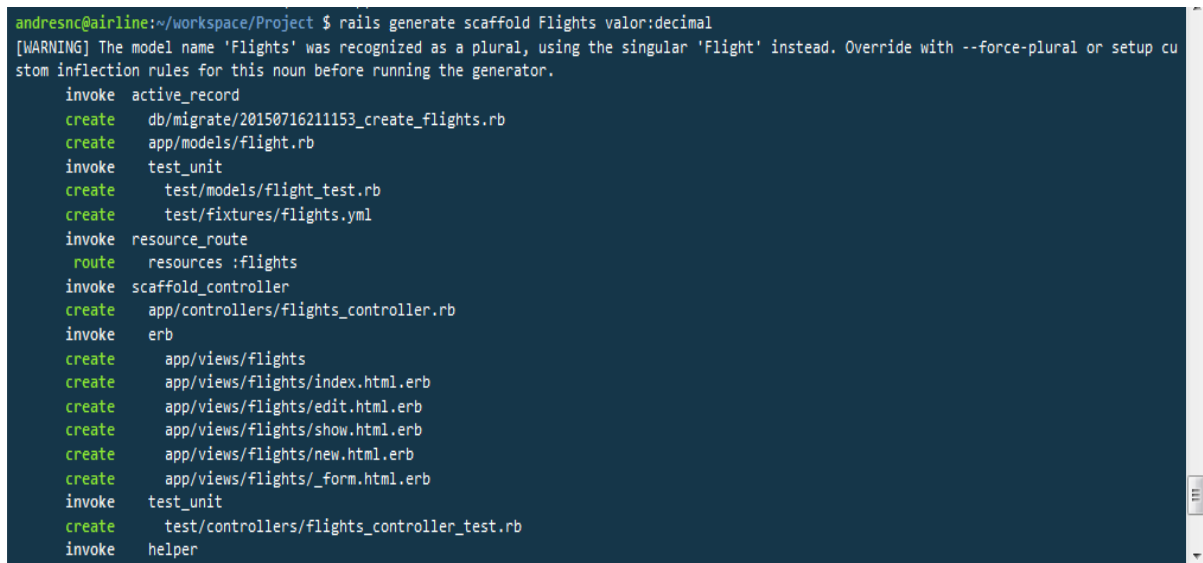
```
andresnc@airline:~/workspace $ cd Project
andresnc@airline:~/workspace/Project $ bundle install
Using rake 10.4.2
Using i18n 0.7.0
Using json 1.8.3
Using minitest 5.7.0
Using thread_safe 0.3.5
Using tzinfo 1.2.2
Using activesupport 4.2.1
Using builder 3.2.2
Using erubis 2.7.0
Using mini_portile 0.6.2
Using nokogiri 1.6.6.2
Using rails-deprecated_sanitizer 1.0.3
Using rails-dom-testing 1.0.6
Using loofah 2.0.2
```


Para comprobar que ha instalado todo correctamente, usted debería ser capaz de ver en el árbol de directorio en el apartado llamado GemFile toda la información relacionada con RoR, como la versión la base de datos etc.



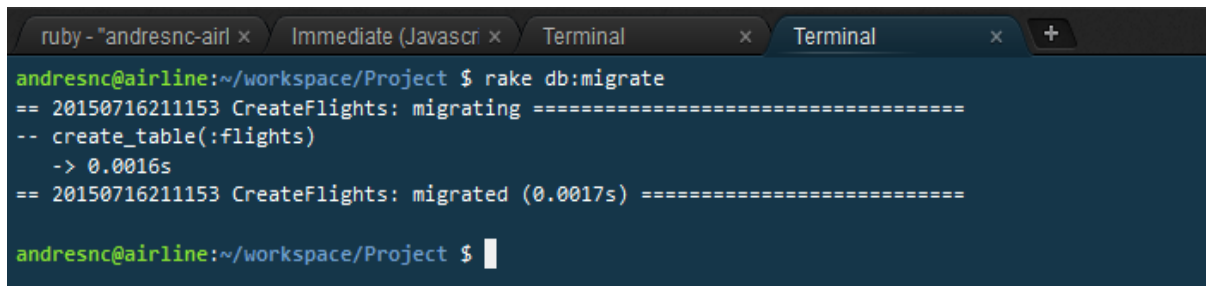
```
1 source 'https://rubygems.org'
2
3
4 # Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
5 gem 'rails', '4.2.1'
6 # Use sqlite3 as the database for Active Record
7 gem 'sqlite3'
8 # Use SCSS for stylesheets
9 gem 'sass-rails', '~> 5.0'
10 # Use Uglifier as compressor for JavaScript assets
11 gem 'uglifier', '>= 1.3.0'
12 # Use CoffeeScript for .coffee assets and views
13 gem 'coffee-rails', '~> 4.1.0'
14 # See https://github.com/rails/execjs#readme for more supported runtimes
15 # gem 'therubyracer', platforms: :ruby
16
17 # Use jquery as the JavaScript library
18 gem 'jquery-rails'
19 # Turbolinks makes following links in your web application faster. Read more: https://github.com/rails/turbolinks
20 gem 'turbolinks'
21 # Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
22 gem 'jbuilder', '~> 2.0'
23 # bundle exec rake doc:rails generates the API under doc/api.
24 gem 'sdoc', '~> 0.4.0', group: :doc
25
26 # Use ActiveModel has_secure_password
27 # gem 'bcrypt', '~> 3.1.7'
28
29 # Use Unicorn as the app server
30 # gem 'unicorn'
31
32 # Use Capistrano for deployment
33 # gem 'capistrano-rails', group: :development
```

Procedemos a ejecutar una gema de ruby llamada scaffold la cual nos genera un crud simplemente le damos el nombre al modelo en nuestro caso fue Flights el cual por defecto nos genera automáticamente un id (clave primaria), con sus datos y atributos correspondientes. Esto lo podemos apreciar en la siguiente imagen.



```
andresnc@airline:~/workspace/Project $ rails generate scaffold Flight valor:decimal
[WARNING] The model name 'Flight' was recognized as a plural, using the singular 'flight' instead. Override with --force-plural or setup custom inflection rules for this noun before running the generator.
  invoke  active_record
  create  db/migrate/20150716211153_create_flights.rb
  create  app/models/flight.rb
  invoke  test_unit
  create  test/models/flight_test.rb
  create  test/fixtures/flights.yml
  invoke  resource_route
  route   resources :flights
  invoke  scaffold_controller
  create  app/controllers/flights_controller.rb
  invoke  erb
  create  app/views/flights
  create  app/views/flights/index.html.erb
  create  app/views/flights/edit.html.erb
  create  app/views/flights/show.html.erb
  create  app/views/flights/new.html.erb
  create  app/views/flights/_form.html.erb
  invoke  test_unit
  create  test/controllers/flights_controller_test.rb
  invoke  helper
```


Procedemos a ejecutar el comando `rake db:migrate` en el cual se ejecutan las migraciones que no se han ejecutado todavía. Y se crean todas las tablas correspondientes como se muestra a continuación.



```
ruby - "andresnc-airline" x Immediate (Javascript) x Terminal x Terminal x +
andresnc@airline:~/workspace/Project $ rake db:migrate
== 20150716211153 CreateFlights: migrating =====
-- create_table(:flights)
   -> 0.0016s
== 20150716211153 CreateFlights: migrated (0.0017s) =====

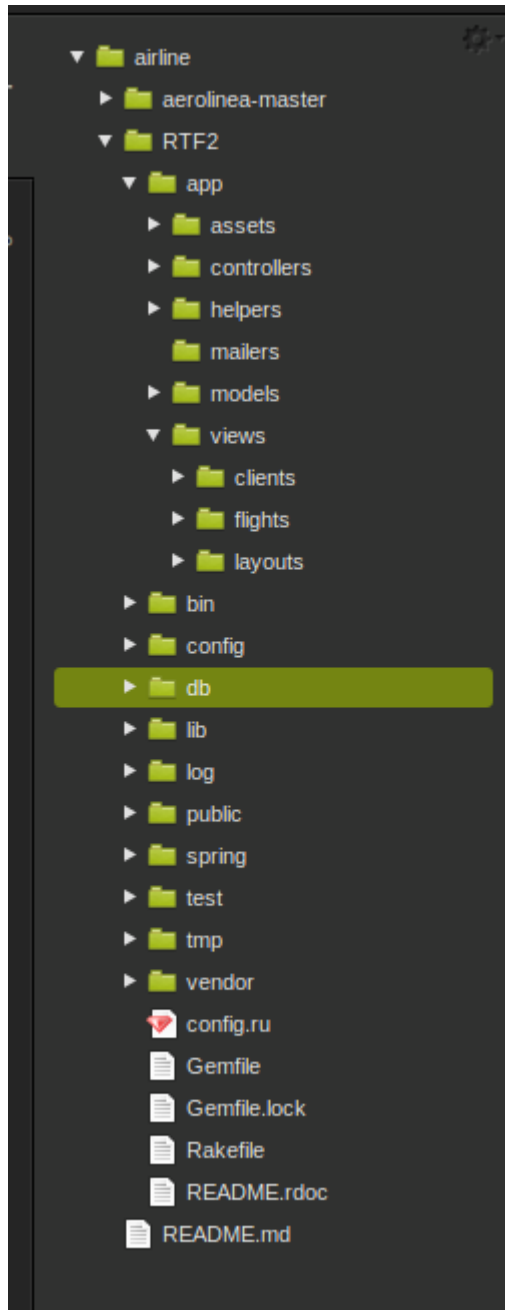
andresnc@airline:~/workspace/Project $
```

Esto lo podemos apreciar en el árbol de nuestro proyecto, en la parte de `db/schema.rb`, el cual nos muestra la tabla creada con sus respectivos atributos.



```
application.c x x edit.html.erb x new.html.erb x _form.html.e x index.html.e x client.rb x Gemfile x schema.rb x + Collaborate Outline Debugger Live Coding Help
1 # encoding: UTF-8
2 # This file is auto-generated from the current state of the database. Instead
3 # of editing this file, please use the migrations feature of Active Record to
4 # incrementally modify your database, and then regenerate this schema definition.
5 #
6 # Note that this schema.rb definition is the authoritative source for your
7 # database schema. If you need to create the application database on another
8 # system, you should be using db:schema:load, not running all the migrations
9 # from scratch. The latter is a flawed and unsustainable approach (the more migrations
10 # you'll amass, the slower it'll run and the greater likelihood for issues).
11 #
12 # It's strongly recommended that you check this file into your version control system.
13
14 ActiveRecord::Schema.define(version: 20150716211153) do
15
16   create_table "flights", force: :cascade do |t|
17     t.decimal "valor"
18     t.datetime "created_at", null: false
19     t.datetime "updated_at", null: false
20   end
21
22 end
23
```

3.3 Árbol De Directorios Del Framework Con El Contenido De Cada Carpeta.



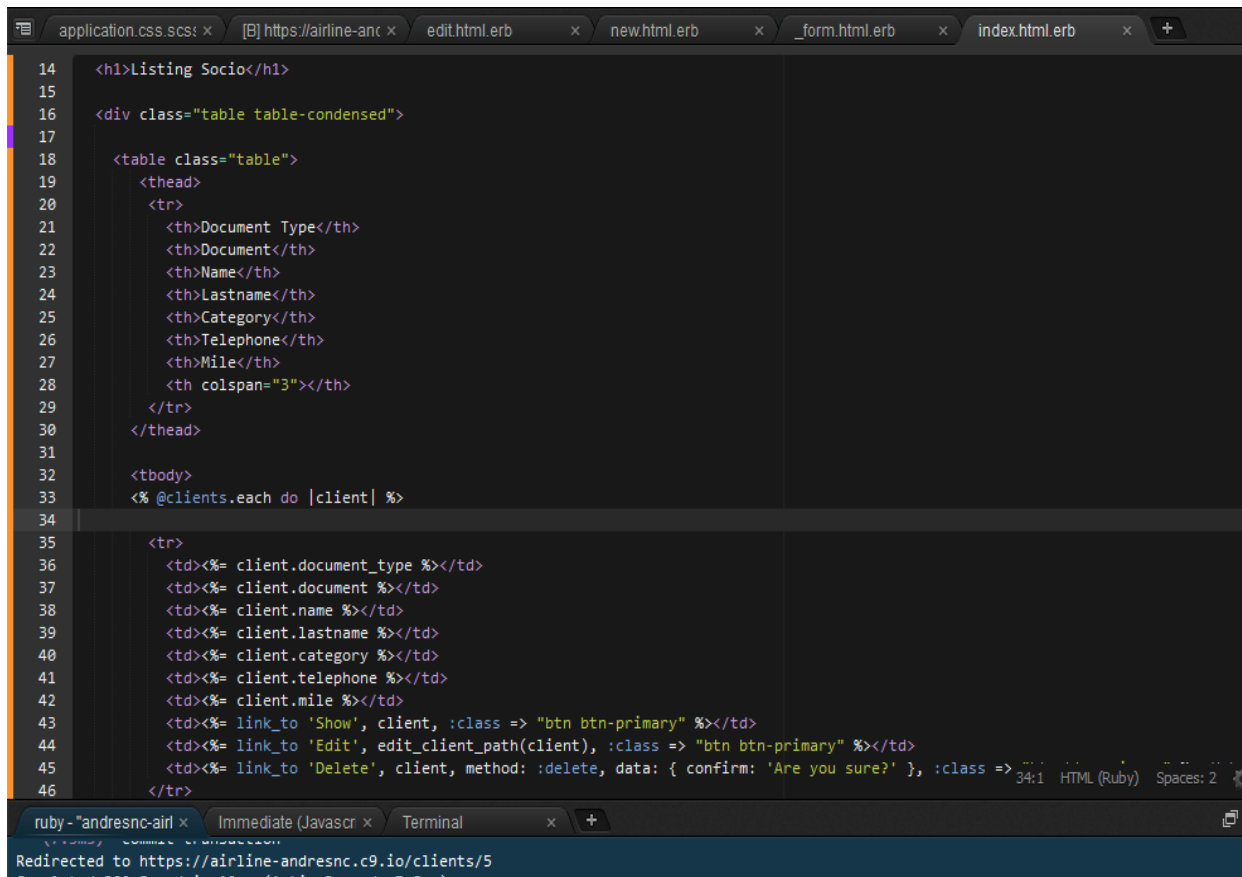
- app: Éste organiza los componentes de la aplicación. Los subdirectorios que contienen la vista (view - helper), controlador (controller), y la lógica de negocio de back-end (model).
- app / controllers: El subdirectorio controllers es donde Rails intenta encontrar las clases controller. Un controlador controla una solicitud web por parte del usuario.

- `app / helpers`: El subdirectorío `helpers` tiene cualquier clase de ayuda que se utilizan para ayudar a las clases modelo, vista y controlador. Esto ayuda a mantener el código pequeño, centrado, y despejado.
- `app / models`: El subdirectorío `models` contiene las clases que envuelven el modelo y los datos almacenados en la base de datos de nuestra aplicación. En la mayoría de los frameworks, esta parte de la aplicación puede ser bastante complejo, aburrido y propenso a errores. Rails hace que sea mucho más simple !
- `app / view`: El subdirectorío `view` mantiene las plantillas de visualización para rellenar con los datos de nuestra aplicación, convierte a HTML, y retorna los datos del usuario.
- `app / view / layouts`: Contiene los archivos de plantilla para los diseños que se utilizarán en las views. En este modelo es común encontrar los métodos de header/footer.
- `components`: Este directorio almacena componentes de aplicaciones autónomas diminutas que se agrupan en el modelo, vista/controlador.
- `config`: Este directorio contiene una pequeña cantidad de código de configuración que tendrá su aplicación, incluyendo la configuración de base de datos (en `database.yml`), la estructura de su entorno de Rails (`environment.rb`), y el enrutamiento de las peticiones web entrantes (`routes.rb`). También se puede adaptar el comportamiento de los árboles de Rails para la prueba, el desarrollo y el despliegue con los archivos que se encuentran en este directorio.
- `db`: Por lo general, una aplicación Rails tendrá objetos del modelo que tienen acceso a las tablas de bases de datos relacionales. Se puede gestionar la base de datos relacional con los scripts que se crean y estos se colocan en este directorio.
- `doc`: Ruby tiene un Framework, llamado RubyDoc, que puede generar automáticamente la documentación para el código que se creó. Usted puede ayudar a RubyDoc con comentarios en su código. Este directorio contiene toda la documentación Rails y aplicación RubyDoc generados.
- `lib`: Aquí van todas las librerías, a menos que pertenezcan expresamente en otros lugares (como librerías de vendor).

- log: Rails crea scripts que ayudan a administrar varios registros de errores. Se encuentran separados los registros para el servidor (server.log) y para cada entorno Rails (development.log, test.log y production.log).
- public: Al igual que el directorio public de un servidor web, este directorio tiene archivos web que no cambian, como archivos JavaScript (publics / javascripts), gráficos (publics / images), hojas de estilo (public / stylesheets), y archivos HTML (publics).
- script: Este directorio contiene secuencias de comandos para lanzar y gestionar las distintas herramientas que utiliza Rails.
- Test: Los tests se escriben y Rails los guarda en este directorio. Hay subdirectorio para mocks (mocks), pruebas unitarias (unit), accesorios (fixtures), y pruebas funcionales (functionals).
- tmp: Rails utiliza este directorio para almacenar archivos temporales para el procesamiento intermedio.
- vendor: Librerías proporcionadas por terceros (tales como librerías de seguridad o utilidades de bases de datos más allá de la distribución Rails básicos).
- README: Este archivo contiene un detalle básico sobre Aplicación Rail y una descripción de la estructura de directorios se ha explicado anteriormente.
- Rakefile: Este archivo es similar a Unix Makefile que ayuda con la construcción, empaquetamiento y prueba el código de Rails.

3.4. Código De Ejemplo Del Montaje En Cada Capa

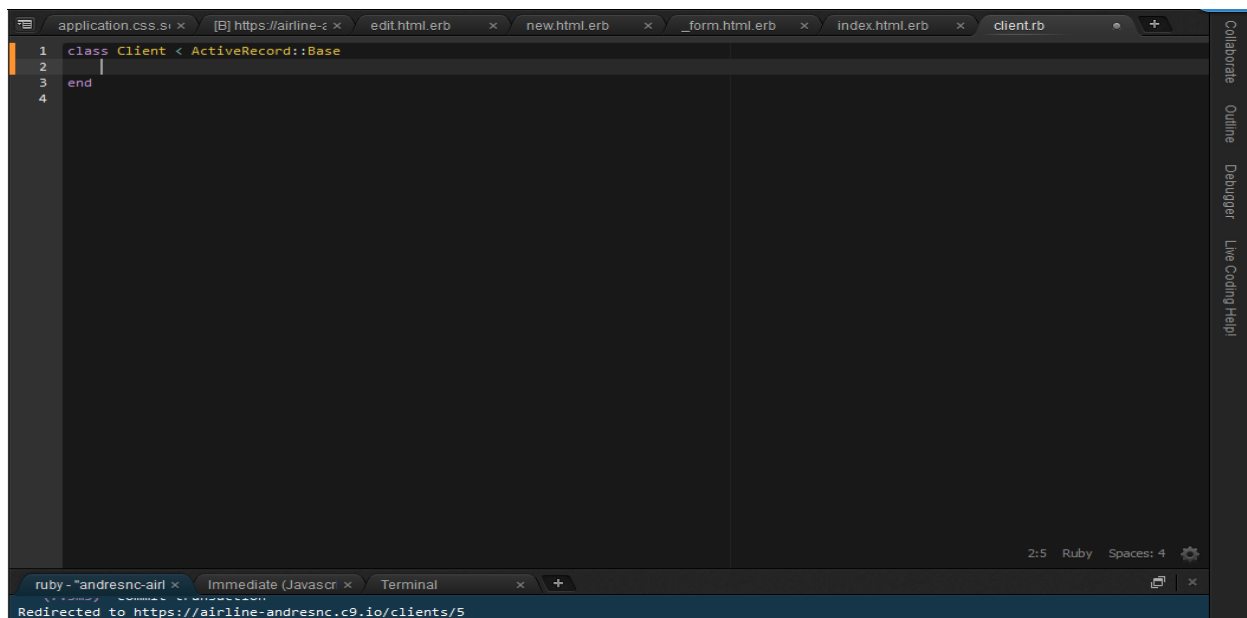
Como podemos apreciar esta capa contiene todo el html, lo cual indica que es la capa de la vista



```
14 <h1>Listing Socio</h1>
15
16 <div class="table table-condensed">
17
18   <table class="table">
19     <thead>
20       <tr>
21         <th>Document Type</th>
22         <th>Document</th>
23         <th>Name</th>
24         <th>Lastname</th>
25         <th>Category</th>
26         <th>Telephone</th>
27         <th>Mile</th>
28         <th colspan="3"></th>
29       </tr>
30     </thead>
31
32     <tbody>
33       <% @clients.each do |client| %>
34
35         <tr>
36           <td><%= client.document_type %></td>
37           <td><%= client.document %></td>
38           <td><%= client.name %></td>
39           <td><%= client.lastname %></td>
40           <td><%= client.category %></td>
41           <td><%= client.telephone %></td>
42           <td><%= client.mile %></td>
43           <td><%= link_to 'Show', client, :class => "btn btn-primary" %></td>
44           <td><%= link_to 'Edit', edit_client_path(client), :class => "btn btn-primary" %></td>
45           <td><%= link_to 'Delete', client, method: :delete, data: { confirm: 'Are you sure?' }, :class =>
46             </td>
```

Terminal output: Redirected to https://airline-andresnc.c9.io/clients/5

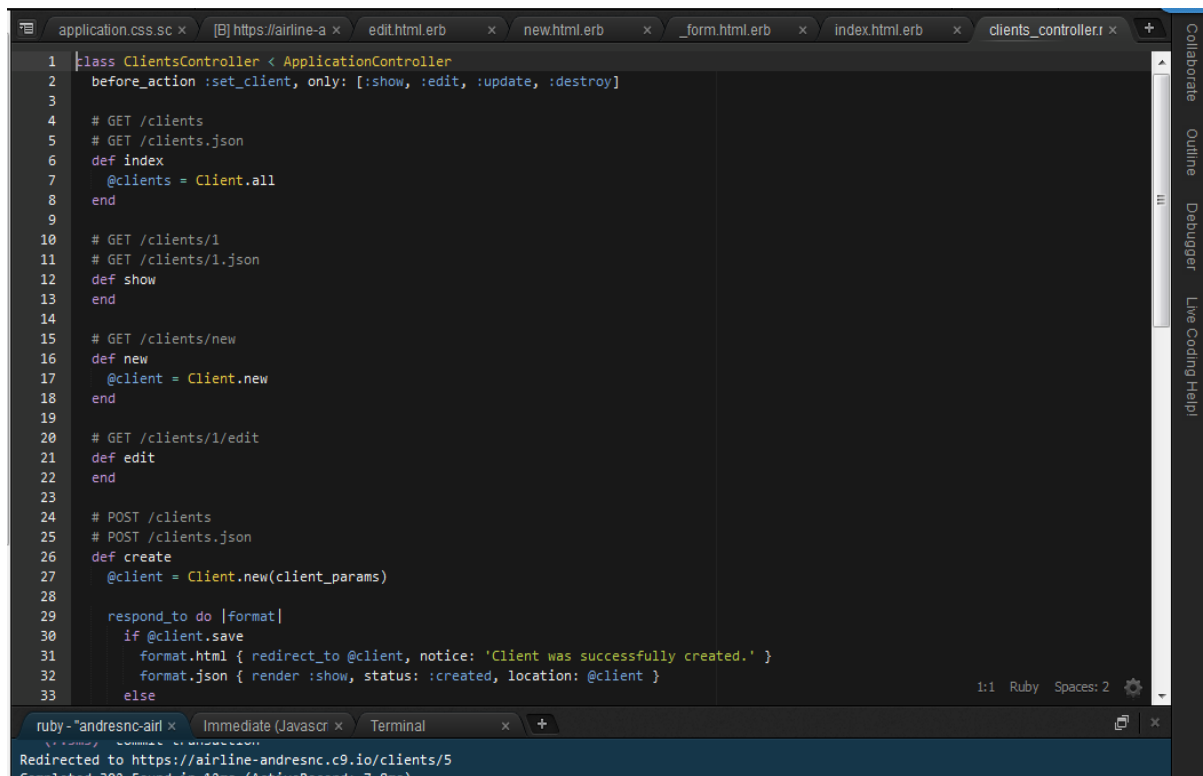
Capa del Modelo.



```
1 class Client < ActiveRecord::Base
2
3 end
4
```

Terminal output: Redirected to https://airline-andresnc.c9.io/clients/5

Capa del Controlador



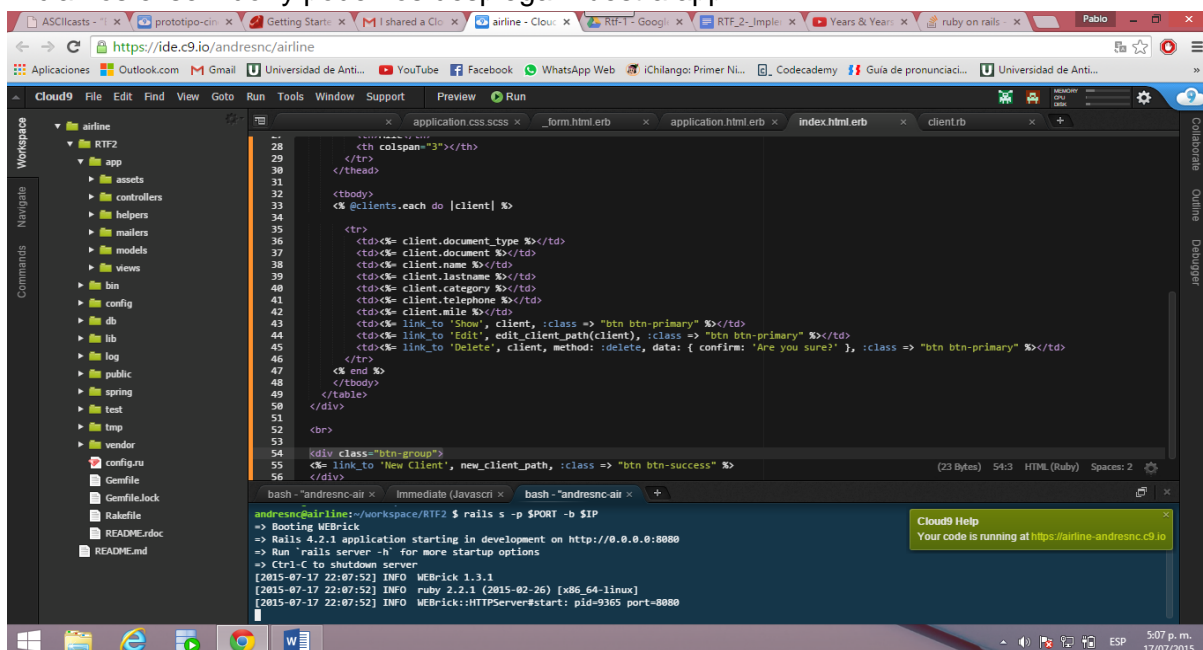
```
1 class ClientsController < ApplicationController
2   before_action :set_client, only: [:show, :edit, :update, :destroy]
3
4   # GET /clients
5   # GET /clients.json
6   def index
7     @clients = Client.all
8   end
9
10  # GET /clients/1
11  # GET /clients/1.json
12  def show
13  end
14
15  # GET /clients/new
16  def new
17    @client = Client.new
18  end
19
20  # GET /clients/1/edit
21  def edit
22  end
23
24  # POST /clients
25  # POST /clients.json
26  def create
27    @client = Client.new(client_params)
28
29    respond_to do |format|
30      if @client.save
31        format.html { redirect_to @client, notice: 'Client was successfully created.' }
32        format.json { render :show, status: :created, location: @client }
33      else
34      end
35    end
36  end
37 end
```

1:1 Ruby Spaces: 2

Redirected to https://airline-andresnc.c9.io/clients/5

3.5. Despliegue De La Aplicación Montada En El Framework

Para esto nos dirigimos a nuestra plataforma de desarrollo en la nube c9.io y en la terminal provista por este ingresamos el siguiente comando rails s -p \$PORT -b \$IP con esto iniciamos el servidor y podemos desplegar nuestra app.



Browser: <https://ide.c9.io/andresnc/airline>

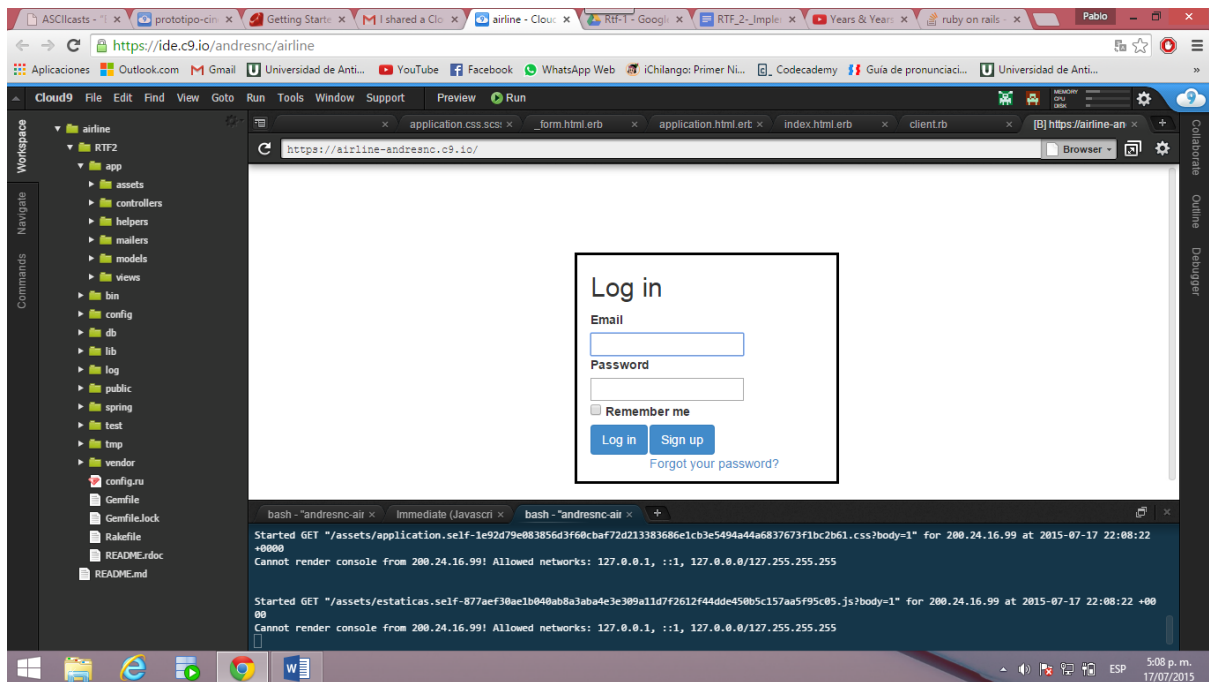
Terminal:

```
andresnc@airline:~/workspace/RTF2 $ rails s -p $PORT -b $IP
=> Booting WEBrick
=> Rails 4.2.1 application starting in development on http://0.0.0.0:8080
=> Run "rails server -h" for more startup options
=> Ctrl-C to shutdown server
[2015-07-17 22:07:52] INFO WEBrick 1.3.1
[2015-07-17 22:07:52] INFO ruby 2.2.1 (2015-02-26) [x86_64-linux]
[2015-07-17 22:07:52] INFO WEBrick:HTTPServer#start: pid=9365 port=8080
```

Cloud9 Help: Your code is running at <https://airline-andresnc.c9.io>

En el paso anterior también es posible acceder a la aplicación colocando la url en el navegador deseado el cual lo dirigirá al despliegue de la aplicación.

En nuestra aplicación la página de inicio será el log in que se muestra a continuación, el cual nos permitirá acceder como agente de ventas para realizar las transacciones deseadas.



En esta imagen podemos apreciar el listado de los socios que actualmente se encuentran registrados en nuestra aerolínea, mostrando detalladamente su información, además de permitir la realización de un CRUD sobre cada miembro.

Listing Socio

| Document Type | Document | Name | Lastname | Category | Telephone | Mile | | | |
|---------------|------------|------|----------|----------|-----------|--------|------|------|--------|
| cc | 222 | psps | sjows | 1 | 323 | 123.0 | Show | Edit | Delete |
| T.l | 123 | wert | wertt | 1 | 23 | 2344.0 | Show | Edit | Delete |
| C.c | 1214714749 | luis | zambrano | 2 | 98765432 | 89.0 | Show | Edit | Delete |

New Client

Continuamos con la visualización de como el agente de ventas podrá registrar un nuevo socio, en el cual se permite ingresar los atributos correspondientes a un socio con su debida categoría las cuales pueden ser, "Silver", "Golden", "Elite"

New Member

Document type

C.c ▼

Document

Name

Lastname

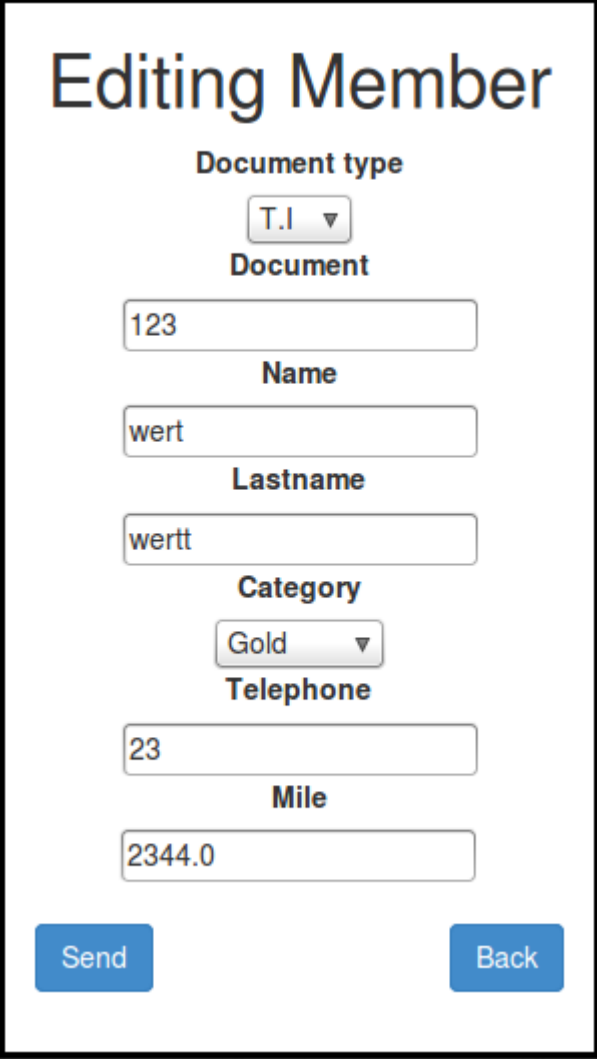
Category

Gold ▼

Telephone

Mile

Por último se aprecia la ventana de cuando se edite un socio, permitiendo hacer todos los cambios correspondientes.



The screenshot shows a web form titled "Editing Member". It contains several input fields and dropdown menus. The "Document type" dropdown is set to "T.I.". The "Document" field contains "123". The "Name" field contains "wert". The "Lastname" field contains "wertt". The "Category" dropdown is set to "Gold". The "Telephone" field contains "23". The "Mile" field contains "2344.0". At the bottom, there are two blue buttons: "Send" and "Back".

| Field | Value |
|---------------|--------|
| Document type | T.I. |
| Document | 123 |
| Name | wert |
| Lastname | wertt |
| Category | Gold |
| Telephone | 23 |
| Mile | 2344.0 |

4. PATRÓN POSA O J2EE

Patrón POSA.

1. Nombre

Model-View-Controller MVC (Modelo Vista Controlador)

2. Propósito

El patrón modelo-vista-controlador separa los datos de una aplicación de la interfaz de usuario y la lógica del negocio, cada uno en un componente distinto y con una responsabilidad particular, cada uno de ellos se puede ver como un

componente de software diferente, cada uno de ellos puede ser reemplazado sin afectar las otras 2 capas restantes.

Es parte importante de muchas aplicaciones o sistemas usar un mecanismo de almacenamiento persistente, en el cual se puedan recuperar datos que después serán consultados por el usuario. Lo anterior genera un gran acoplamiento en la manera como se ingresan y muestran los datos y la manera como se guardan, es decir se ligan demasiado la capa de presentación y la capa de persistencia, lo cual genera algunos problemas. Si en vez de manejar una capa diferente para la lógica del negocio se tiene en la misma capa de presentación, si se debe realizar un cambio por más pequeño que sea, en código puede llegar a ser una gran modificación.

Es muy difícil para el programador y costoso para la aplicación, mostrar los datos de maneras diferentes y si se realiza el cambio en los datos en una de las 2 vistas se deben de actualizar las demás, algo que se realizaría de una manera no eficiente.

El propósito fundamental de patrón es hacer una separación tanto física como lógica de los datos, peticiones y la forma en cómo se muestran los datos asociados a dichas peticiones en una interfaz gráfica visible directamente por el usuario.

El problema sobre el que enfatiza este patrón es: ¿Cómo modularizar la funcionalidad de la interfaz de usuario de una aplicación (especialmente web) para que se puedan modificar fácilmente las piezas individuales de software?

3. Sinónimos

En inglés es conocido como Model-View-Controller, en español traducido directamente se conoce como Modelo-Vista-Controlador. En cualquier idioma se conoce por su sigla en inglés MVC.

4. Motivación

En cada programa podemos encontrar tres responsabilidades principales:

1. La lógica de la aplicación (qué hacer).
2. Los datos sobre los que se opera (sobre qué datos se opera y cómo se organizan esto).
3. La presentación de la aplicación (como se expone a sus usuarios).

Cada una de las responsabilidades anteriores pueden ser implementadas por una o más clases, en la mayoría de los casos más de una, el patrón MVC nos dice cómo hacer esas interacciones para que la solución final sea extensible y reusable

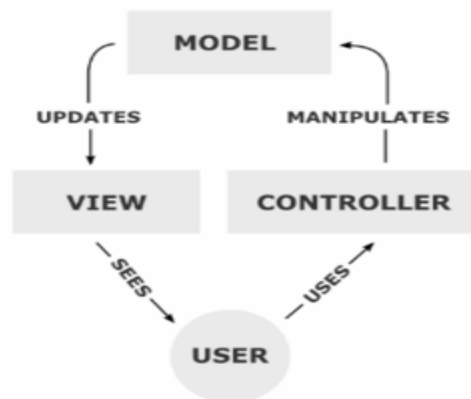
5. Aplicabilidad

Se aplica cuando es de gran importancia separar vista/lógica/datos un gran acoplamiento entre estas es un gran problema para cualquier aplicación por más pequeña que sea.

En primera instancia se le reconoce por la estructura del proyecto si quienes lo realizaron tuvieron en cuenta una buenas prácticas de programación se reconocerá con facilidad la división por capas. así se hace mucho más simple para un programador que retome un proyecto de otro, encontrar con mayor facilidad donde se encuentra el problema y sea más fácil de modificar, para mayor eficiencia

a la hora de programar y menores costos se debe tener una estructura bien definida del proyecto,

6. Estructura



Tomado de <https://es.wikipedia.org/wiki/Modelo%2%80%93vista%2%80%93controlador>.

7. Participantes

Model: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador', por lo general lleva el nombre de Client.rb, Flight.rb, etc.

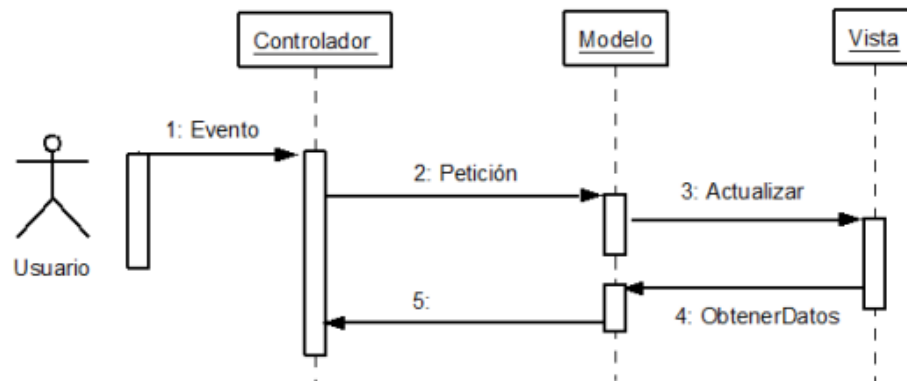
View: Presenta el 'modelo' en un formato adecuado para interactuar por tanto requiere de dicho 'modelo' la información que debe representar como salida. Cada acción dentro de un controlador puede tener un punto de vista que es un archivo RHTML lleva el nombre de la acción asociada. Un archivo RHTML es un archivo HTM con trozos de Ruby en su interior.

Controller: Responde a eventos e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos), lo ideal sería que no contuviese SQL. También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo', se encuentra en el directorio de controladores y lleva el nombre del objeto plural: clients.rb, flights.rb.

8. Colaboraciones

La interacción inicia en la capa de la vista, cuando el usuario interactúa con esta de alguna forma. El controlador la notificación de la interacción del usuario con la

vista, dado el caso el controlador accede al modelo para crear, leer, modificar o eliminar datos. Por último la vista recibe una señal del controlador que le indica qué debe hacer.



9. Consecuencias

El patrón alcanza sus objetivos con la división de responsabilidades, no hacer que sobre una misma clase recaiga toda la carga de la aplicación sino que cada capa tenga sus propias tareas con las que pueda cumplir a cabalidad y sea fácil comunicarse con las demás capas.

Si el patrón es bien aplicado se mejora la reutilización, extensibilidad, flexibilidad y el resto de elementos de las aplicaciones

10. Implementación

Es necesario realizar un buen diseño donde se divida correctamente cada capa, con el fin de evitar que se mezcle el código al momento de implementar el patrón.

11. Ejemplo de Código

El código mostrado anteriormente destaca el uso de este patrón.

12. Usos conocidos

- Ruby on rails
- Backbone.js
- Ember.js
- Spring
- Struts.
- Django
- ASP.NET mvc
- Cocoa

13. Patrones relacionados

- HMVC (MVC Jerárquico)

MVA (Modelo-Vista-Adaptador)
MVP (Modelo-Vista-Presentador)
MVVM (Modelo-Vista Vista-Modelo)

14. Caso de Aplicación

Busque un escenario apropiado dentro de su proyecto, para la aplicación de los patrones trabajados en la primera parte de este taller, y describa con claridad el problema que se presenta.

15. Justificación

MVC tiene por objeto separar la lógica del negocio de las consideraciones de la interfaz de usuario para que los desarrolladores puedan modificar cada parte más fácilmente sin afectar a la otra. En MVC el modelo representa la información (los datos= y las reglas del negocio; la vista contiene elementos de la interfaz de usuario como textos, formularios de entrada; y el controlador administra la comunicación entre la vista y el modelo.

Más allá del MVC, "*Ruby and Rails*" está diseñado para hacer la vida del programador más fácil. Las convenciones no son difíciles de comprender e implementar. Un ejemplo de una convención significativa es en la denominación de las tablas de BD y campos. Esta convención hace que sea fácil de establecer relaciones entre los objetos en una aplicación. Tablas de tablas se establecen mediante la inclusión de un campo estratégicamente nombrado en la tabla correspondiente, declaraciones pertinentes en los modelos de los objetos. Una aplicación Rails tiene una convención de la estructura de directorios y archivos estándar. Cada cosa tiene su lugar y cada lugar tiene su cosa. De aquí sus rieles de trabajo se basan en el MVC

16. Aplicación

Dado que se quieren desacoplar las partes antes mencionadas (vista, lógica del negocio, lógica de la aplicación, y persistencia)

17. Ejemplo de Aplicación

El ejemplo de código se puede ver en el apartado de este trabajo referente al framework ruby on rails

BIBLIOGRAFIA

Mejia, Adrian. (2011) Ruby on Rails Architectural Design. Online. Available:
<http://adrianmejia.com/blog/2011/08/11/ruby-on-rails-architectural-design/>

Rails.org. Rails Guides. Online. Available:
http://guides.rubyonrails.org/getting_started.html.

Sun Microsystems. (2001). SUNTONE ARCHITECTURE METHODOLOGY A 3-DIMENSIONAL APPROACH TO ARCHITECTURAL DESIGN. USA. Online. Available:
http://rieck.dyndns.org/architecture/suntoneam_wp_5.24.pdf.