Introducció a Ruby on Rails

Jordi Sánchez Cano

PID_00172687

Universitat Oberta de Catalunya
www.uoc.edu

Índex

Introducció

Ruby on Rails és un entorn de treball (*framework*) que té com a objectiu el desenvolupament ràpid i senzill d'aplicacions web. En els últims anys, Rails ha anat guanyant popularitat i un gran nombre d'adeptes per la seva agilitat i senzillesa. D'una banda, Ruby és un llenguatge molt còmode i expressiu que intenta ser tan natural com es pugui per al llenguatge humà. D'una altra banda, Rails està construït sobre una filosofia pràctica que evita la repetició de codi i les configuracions.

En aquest mòdul es fa una introducció a l'entorn de treball Rails d'una manera general. Al final, haureu après alguns conceptes bàsics de Ruby on Rails. Aquests no són suficients per a fer aplicacions web professionals, però són un bon principi per a introduir-se en aquest entorn de treball i en el llenguatge Ruby.

Objectius

Amb el treball d'aquest mòdul assolireu els objectius següents:

1. Conèixer què és *Ruby on Rails*.
2. Entendre l'arquitectura bàsica d'una aplicació Ruby on Rails.
3. Conèixer el patró de disseny model controlador vista.
4. Aprendre a fer aplicacions senzilles amb Ruby on Rails.
5. Estudiar com s'ha d'usar AJAX en Ruby on Rails.

1. Què és *Ruby on Rails*

En aquest apartat veurem els conceptes bàsics de Ruby on Rails.

1.1. El llenguatge de programació Ruby

Ruby és un llenguatge de programació interpretat i orientat a objectes inspirat en altres llenguatges com Python, Perl i Smalltalk. El resultat és un llenguatge de programació concís i llegible però alhora potent.

Origen de Ruby

Ruby va ser creat per Yukihiro "Matz" Matsumoto, programador japonès. En va iniciar el desenvolupament el 1993 i el va presentar l'any 1995.

L'entorn de Matsumoto anomenava aquest llenguatge *robí*, ateses les similituds amb Perl, que significa *perla*.

Ús lliure i portabilitat de Ruby

Ruby és un llenguatge de codi obert i gratuït. Pot ser usat, copiat, modificat i distribuït lliurement.

Una aplicació Ruby és fàcilment portable a altres sistemes operatius com GNU/Linux, UNIX, Mac OS X i tota la família de sistemes operatius Windows.

Les característiques més importants de Ruby són:

- **Està orientat a objectes**: la programació orientada a objectes facilita l'escalabilitat i reutilització del programari. Ruby està orientat a objectes i tots els seus tipus de dades són objectes, fins i tot les simples com, per exemple, numèriques, booleanes, valors nuls, etc.

- **Interpretat**: disposa d'intèrprets per a diferents arquitectures i sistemes operatius que permeten l'execució de programes escrits en Ruby sense compilació prèvia. Aquesta característica té grans avantatges, com la independència de plataforma i la reflexió.

- **Reflectiu**: Ruby pot generar i executar codi creat d'una manera dinàmica. Per exemple, es pot executar codi Ruby contingut en una cadena de caràcters de manera dinàmica.

- **Biblioteca estàndard**: disposa d'un conjunt de classes i funcions escrites en Ruby que proporcionen suport a determinades operacions: tipus abstractes de dades, *sockets*, gestió de dates, etc.

- **Recol·lector d'escombraries**[1]: gestiona la destrucció i l'alliberament de memòria de manera automàtica.

(1)En anglès, *garbage collector*.

1.2. L'entorn de treball Rails

Ruby on Rails, o simplement Rails, és un entorn de treball basat en el llenguatge Ruby que facilita el desenvolupament, desplegament i manteniment d'aplicacions web.

Hi ha altres entorns de treball, per exemple Struts i Spring per a Java/J2EE. Maverick és per al desenvolupament d'aplicacions web, però Rails se'n destaca per l'agilitat i simplicitat.

Rails està construït sobre dos principis molt sòlids:

1) Convenció sobre configuració. Aquest principi consisteix a definir només les configuracions que no segueixen les convencions de Rails. Es pot prescindir de la majoria d'arxius de configuració i qualsevol configuració no definida d'una manera explícita serà assumida per una per defecte.

2) No et repeteixis (DRY[2]). Aquesta filosofia segueix una idea molt simple: tot concepte en l'aplicació s'ha de definir una sola vegada. L'objectiu d'aquest principi és evitar la duplicació d'informació.

(2) *DRY* és la sigla de *don't repeat yourself*.

2. Arquitectura d'una aplicació Ruby on Rails

En aquest apartat veurem com s'organitza una aplicació Ruby on Rails.

2.1. Patró MVC

Una aplicació Rails presenta una arquitectura MVC[3], un dels patrons més estesos per al desenvolupament d'aplicacions web. Aquest tipus d'arquitectura separa la interfície d'usuari, les dades i la lògica de control en tres components ben diferenciats i defineix la relació entre aquests:

(3)*MVC* és la sigla de *model vista controlador*.

1) Model: representa les dades de l'aplicació i s'encarrega de gestionar-ne la càrrega, la persistència i la integritat. Aquesta capa haurà d'operar amb la informació a partir d'una base de dades, de fitxers binaris, de l'XML, etc.

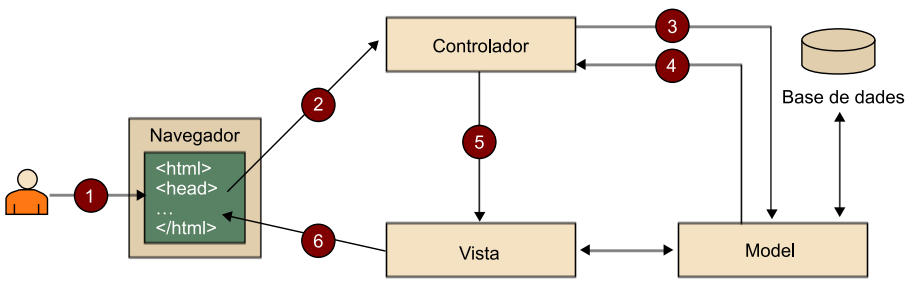
2) Vista: és la responsable de generar les interfícies d'usuari. Part de les dades mostrades en les interfícies es basen generalment en les dades obtingudes a partir del model. En una aplicació web, les vistes representen el conjunt de pàgines HTML (i les generades de manera dinàmica), codi JavaScript i estils CSS associats.

3) Controlador: rep les peticions provinents de la interfície d'usuari i en coordina les respostes. Cada petició es delega a un controlador que s'encarrega de fer les accions requerides fins a generar una resposta.

Per exemple, es disposa d'una aplicació web per a la venda de llibres. Si l'usuari fa clic sobre el títol d'un llibre per a visualitzar-ne les característiques, el servidor rep la petició i delega la tasca al controlador encarregat d'aquesta funció. Aquest accedirà al model, obtindrà el llibre amb l'ISBN indicat en l'URL com a paràmetre i retornarà al navegador una vista que en mostrarà els detalls.

La figura 1 mostra els tres tipus de components i la relació entre ells des del punt de vista d'una acció, i enumera cada pas.

Figura 1. Arquitectura MVC



1) L'usuari fa una acció sobre la interfície d'usuari. Per exemple, modificar un camp o sol·licitar informació que ha de ser visualitzada.

Acció

L'acció també pot ser generada a partir d'un procés periòdic sense la intervenció de l'usuari.

2) El navegador genera la petició, que el servidor delega al controlador apropiat.

3) El controlador accedeix al model per a fer les consultes o modificacions.

4) El model retorna les dades sol·licitades, en cas que es tracti d'una consulta.

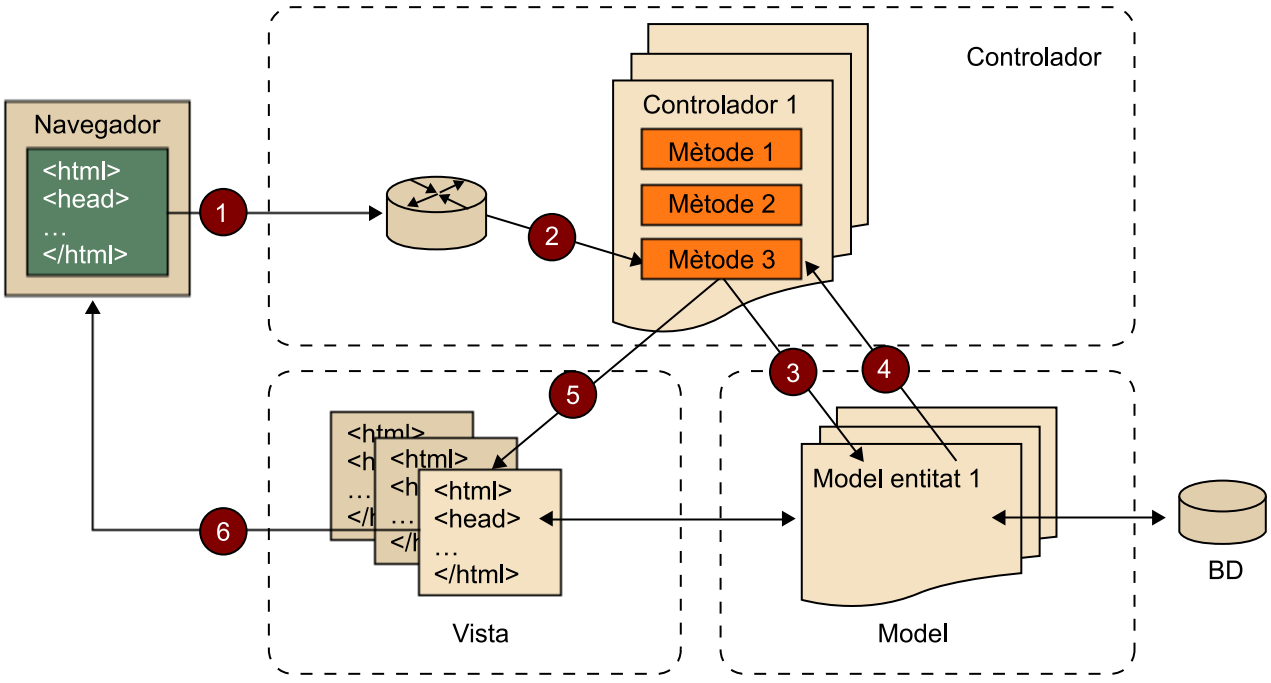
5) El controlador selecciona la vista encarregada de visualitzar la resposta i passa les dades que ha de visualitzar obtingudes pel model.

6) S'envia de retorn a l'usuari la vista com a resposta a la petició.

En una aplicació Rails, totes les peticions són processades per un component anomenat sistema d'encaminament[4]. Aquest obté l'URL i decideix quin controlador i quina acció d'aquest controlador s'ha d'encarregar de la resposta.

[4] En anglès, *routing*.

Figura 2. Encaminament de peticions HTTP en Rails



Com mostra la figura 2, el sistema d'encaminament rep les peticions i les examina a partir de les accions següents:

Acció i mètode

Una acció d'un controlador està relacionada amb l'execució del mètode que porta el mateix nom.

- Determina quin controlador s'ha d'instanciar.
- Determina quin mètode s'ha d'executar.
- Obté els paràmetres de la petició i els traspassa al controlador.

[5] *ORM* és la sigla d'*object-relation mapping*, 'mapatge objecte-relacional'.

2.2. ActiveRecord

ActiveRecord forma part de Rails i és una implementació del patró de disseny ORM[5] en Rails. Aquest patró connecta els objectes de negoci amb les taules corresponents en una base de dades relacional per a obtenir i guardar els objectes a partir d'una o diverses taules i abstreu les diferències entre la programació orientada a objectes i les bases de dades relacionals.

En Rails, una classe del model que hereta d'`ActiveRecord`, disposa d'un conjunt ampli de mètodes per a obtenir, modificar i emmagatzemar registres en la base de dades. Active Record assumeix que el nom de la taula és el plural del nom de la classe. Els atributs de la classe s'obtenen de les columnes de la base de dades associada sense necessitat de redefinir-los també en la classe. Per exemple, si es defineix la classe `Persona` i aquesta hereta d'`ActiveRecord`, Rails assumeix per convenció que hi ha una taula en la base de dades anomenada `persones`, i els atributs de classe es prendran a partir de les columnes.

Classe *Persona*

La classe següent hereta d'`ActiveRecord` i defineix un mètode que crea un registre en la base de dades:

```
class Persona < ActiveRecord::Base
  def CrearRegistre
    person = Persona.new
    person.nom = "Juan"
    person.edat = 15
    person.save
  end
end
```

El mètode `CrearRegistre` crea una instància de la classe `Persona` i estableix els atributs *nom* i *edat*. Aquests atributs corresponen a les columnes `nom` i `edat` de la taula persones en la base de dades. Abans d'executar el mètode `CrearRegistre`, haurà d'existir la taula `persones` amb les columnes `nom` i `edat`. Finalment, es crida el mètode `save`, heretat de `ActiveRecord`, que emmagatzema el registre.

L'herència de `ActiveRecord::Base` proporciona l'enllaç amb la taula corresponent i un gran nombre de mètodes de consulta, actualització i eliminació de registres sense que sigui necessari l'ús d'SQL. D'altra banda, ofereix una abstracció que permet als desenvolupadors despreocupar-se de les particularitats de l'SGBD que estiguin utilitzant.

2.3. ActionPack

ActionPack és una biblioteca que consta de dos mòduls Ruby: ActionController i ActionView. L'ús conjunt d'aquests mòduls permet gestionar les peticions i generar les respostes corresponents.

ActionController és el mòdul encarregat de controlar les peticions i dur a terme les accions pertinents per a cadascuna. Qualsevol controlador a Rails hereta de la classe `ActionController` i declara una sèrie de mètodes públics que defineixen les accions disponibles. En rebre una petició des del navegador,

el sistema d'encaminament selecciona el controlador i el mètode a partir de l'URL. ActionController també proporciona mecanismes per a gestionar les peticions en curs, com ara seleccionar la vista de resposta, l'encaminament a una altra acció, gestionar els paràmetres de cada petició, etc.

D'altra banda, **ActionView** encapsula tota la funcionalitat necessària per a renderitzar plantilles. Les plantilles són una barreja entre text estàtic i codi Ruby per a la generació dinàmica pàgines que seran retornades al client com a resposta d'una acció. Hi ha dos tipus de plantilles:

1) rxml: s'utilitzen per a generar respostes XML.

2) rhtml: les més utilitzades. Contenen una barreja de codi HTML i codi Ruby incrustat.

3. Creació d'una aplicació: restaurant UOC

En aquest exemple es desenvoluparà una aplicació web per a gestionar les reserves d'un restaurant. L'aplicació serà accessible per a clients i empleats del restaurant i hi podran donar d'alta, de baixa i modificar reserves. Els empleats del restaurant disposaran de vistes per a visualitzar les reserves previstes per als propers dies i així planificar el programador controlador amb anterioritat.

La configuració de l'entorn requerida per a seguir l'exemple és la següent:

- Tenir instal·lat Ruby i l'entorn de treball Rails amb suport per a MySQL.
- MySQL instal·lat i el servei funcionant al port estàndard. Per defecte, el port d'accés a MySQL és el 3306.
- Aptana RadRails, extensió de l'IDE Eclipse per al desenvolupament d'aplicacions ROR. Requereix Java Runtime.

3.1. Creació del projecte

Per al desenvolupament de l'aplicació s'utilitzarà Aptana Studio, un entorn de treball integrat amb el connector[6] RadRails instal·lat. Igual que altres IDE, Aptana agrupa el codi dins de projectes.

(6)En anglès, *plug-in*.

Aptana Studio i RadRails

Aptana Studio inclou suport per a JavaScript, HTML, DOM i CSS. Està basat en Eclipse, un dels entorns de treball integrat de codi obert més populars. RadRails és un connector de codi obert per al suport de Ruby on Rails a Aptana.

Per a aquest exemple es crearà un projecte anomenat *RestauranteUOC*. Per a això, s'haurà de localitzar la finestra **Ruby Explorer**, clicar el botó dret i seleccionar New > Rails Project.

Obrir finestra Ruby Explorer

Pot ser que la finestra Ruby Project estigui tancada. Per obrir-la, feu clic en Window > Show View > Ruby Explorer.

A continuació, apareixerà un diàleg en què s'especificarà el nom del projecte i altres opcions, que quedaran de la manera següent:

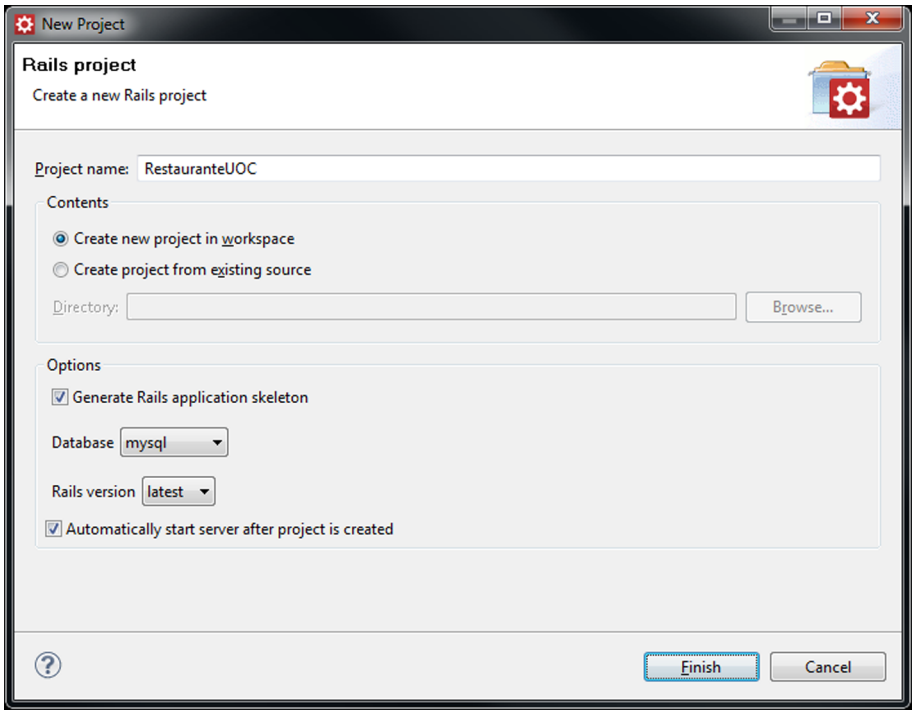
- En el quadre de text Project name: "RestauranteUOC".
- L'opció "Create new project in workspace" haurà d'estar seleccionada.
- La casella de selecció "Generate Rails application skeleton" haurà d'estar activada.
- En el combo Database, se seleccionarà "mysql".
- La casella de selecció "Automatically start server after project is created" haurà d'estar activada.

Una altra manera de crear el projecte

També és possible crear el projecte des de consola utilitzant l'ordre "Rails RestauranteUOC".

La figura 3 mostra com hauria de quedar el formulari abans de continuar:

Figura 3. Creació d'un nou projecte Rails



En fer clic en "Finish" apareixerà dins de la vista Ruby Explorer el projecte nou, representat pel seu nom en forma de directori. Dins del projecte apareixen diversos elements, la majoria dels quals són directoris i fitxers. Si es desplega des de l'arrel apareixeran, entre d'altres, els directoris següents:

- *app*: dins de la jerarquia d'aquest directori hi ha el codi de l'aplicació.
- *controllers*: dins d'aquest directori es troben els fitxers que contenen els controladors de l'aplicació.
- *helpers*: són mètodes Ruby que poden ser cridats des de les vistes.
- *models*: aquest directori contindrà els fitxers que representen les entitats de dades del model.
- *views*: aquí hi ha totes les vistes encarregades de generar HTML que seran enviades al navegador.
- *config*: dins d'aquest directori hi ha els fitxers de configuració de l'entorn de l'aplicació.
- *doc*: Rails pot generar automàticament documentació a partir del codi de l'aplicació. Dins d'aquest directori és on es guardarà aquesta documentació.
- *lib*: les biblioteques addicionals necessàries per al projecte seran dins d'aquest directori.

Directoris

El directori més important per a aquest exemple és `app`, en el qual hi haurà tot el codi de l'aplicació.

- *log*: aquí hi ha diversos fitxers log que es van generant durant l'execució de l'aplicació.

- *public*: dins d'aquest directori s'emmagatzemaran diversos arxius d'accés general com, per exemple, imatges, pàgines HTML estàtiques, fitxers CSS, codi Javascript, etc.

3.1.1. La vista Servers

Per a provar l'aplicació que s'acaba de crear, caldrà iniciar un servidor que allotgi l'aplicació i permeti executar-la. En crear el projecte, es crea de manera automàtica un servidor[7] que allotjarà l'aplicació. Aptana disposa d'una vista anomenada *Servers* que permet iniciar, aturar i configurar els servidors per als diferents projectes. Si aquesta no és visible, es pot mostrar accedint al menú Window > Show View > Servers.

[7] En anglès, *server*.

Dins d'aquesta vista apareixerà una fila per servidor configurat, un dels quals amb el nom del projecte. Les columnes més importants són les següents:

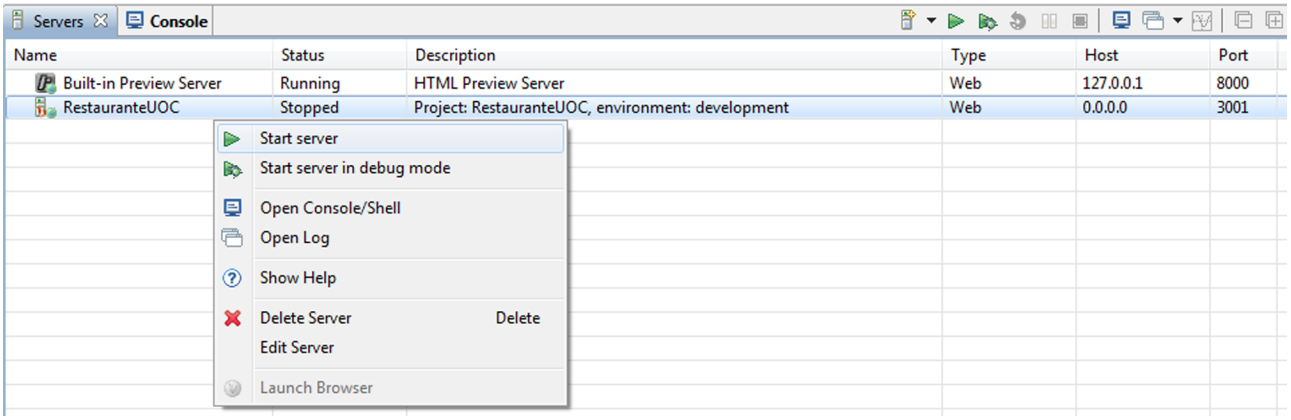
- ***Name***: indica el nom del projecte.

- ***Status***: mostra l'estat del servidor per a cada projecte, que pot ser *stopped* i *started*, si aquest està aturat o iniciat, respectivament.

- ***Port***: indica el port per a accedir a l'aplicació. Cada projecte tindrà un port assignat de manera automàtica a partir del port 3000: la primera aplicació creada tindrà el port 3000, la segona 3001, etc.

Si el servidor de l'aplicació està aturat, es pot fer clic sobre el botó dret i seleccionar l'opció Start server per a iniciar-lo.

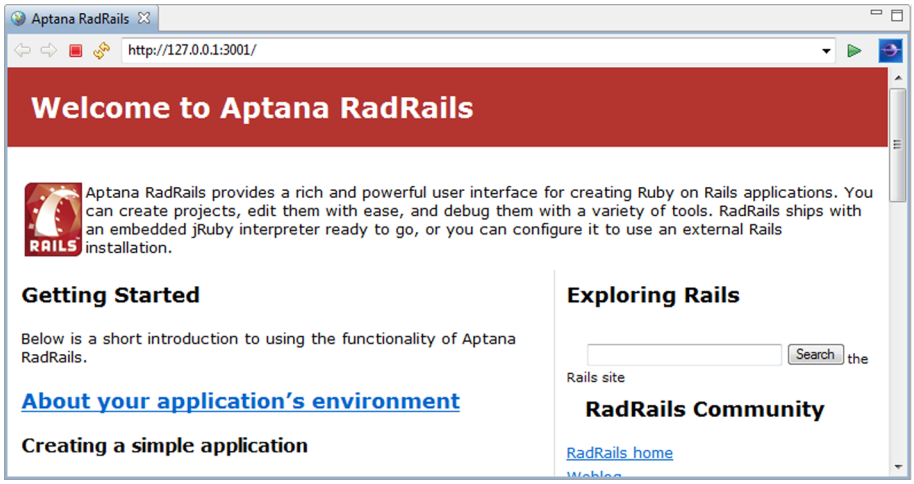
Figura 4. Menú contextual de la vista Servers



En executar l'aplicació en el sistema local, aquesta serà accessible i indicarà l'URL `http://localhost:port` a qualsevol navegador. També es pot utilitzar un navegador incorporat a Aptana que es pot llançar fent clic contextual a la fila del servidor en la vista Servers i seleccionant "Launch Browser". Això farà que aparegui una nova pestanya amb un navegador incrustat.

La figura 5 mostra la pantalla de benvinguda del nou projecte:

Figura 5. Pantalla inicial del projecte que s'acaba de crear



3.1.2. Configuració de l'accés a la base de dades

En aquest punt, ja es disposa d'un projecte i un servidor iniciat. Per defecte, Ruby on Rails intentarà connectar la base de dades amb el nom d'usuari *root* i sense contrasenya. Pot ser que sigui necessari canviar aquesta configuració d'acord amb la configuració de l'SGBD que es vol utilitzar. Per a això, dins de la vista Ruby Explorer s'obrirà l'arxiu RestauranteUOC > config > database.yml. La configuració per defecte dins de l'arxiu és la següent:

Configuració de la base de dades

Si el servidor MySQL està instal·lat a la màquina local i el compte d'administrador no té contrasenya, no caldrà modificar la configuració de la base de dades.

```
development:
  adapter: mysql
  encoding: utf8
  database: RestauranteUOC_development
  username: root
  password:
  host: localhost
```

La primera línia indica que la configuració següent pertany a l'entorn de treball. Hi ha dos blocs més similars al codi anterior per als entorns de test i producció. La taula següent descriu cadascun dels paràmetres.

Paràmetre	Descripció
adapter	Indica l'adaptador per a la connexió a l'SGBD.
encoding	Tipus de codificació utilitzada. Per defecte és UTF8.
database	Nom de la base de dades.
username	Nom d'usuari per a l'accés a la base de dades.
password	Contrasenya.
host	URL en què es localitza la base de dades. Si la base de dades és en el mateix equip, n'hi haurà prou d'indicar *localhost*. Si, per contra, es troba en un altre equip, se n'ha d'indicar la localització.

Nom de base de dades

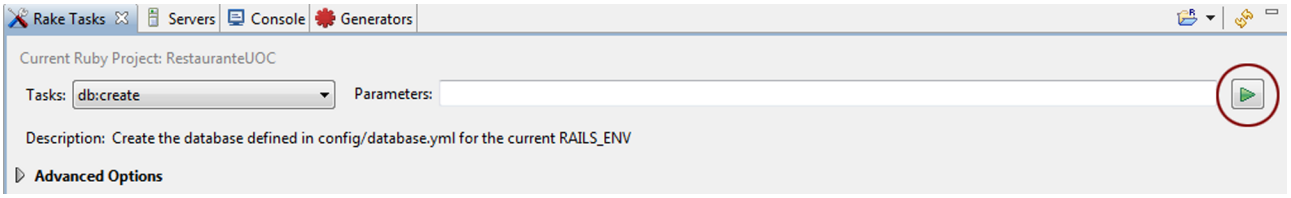
Per convenció, el nom de la base de dades contindrà el nom del projecte seguit del nom de la configuració (`development`, `test`, `production`).

Cada aplicació disposarà de la seva base de dades en l'SGBD establert. Per a crear la base de dades es pot utilitzar l'eina `rake`, que automatitza diferents tasques sobre el nostre projecte. Aptana disposa d'una vista per a executar instruccions amb Rake, a la qual es pot accedir des del menú Window > Show View > Rake Tasks. Una vegada oberta la vista, se seleccionarà del combo l'opció `db:create` i el quadre de text *Parameters* ha de quedar buit.

Rake

Rake permetrà recrear la base de dades i les taules relacionades amb el model quan sigui necessari. Abans de fer una operació amb la instrucció Rake, s'ha de comprovar que s'ha seleccionat el projecte correcte en la vista Ruby Explorer.

Figura 6. Vista Rake Tasks



A continuació, cal fer clic sobre el botó destacat en la figura 6 amb un cercle. L'execució de la instrucció obrirà una vista anomenada Console, que permetrà veure el resultat de l'operació.

Console

Des de consola (Console) també es pot crear la base de dades. Situats en el directori arrel del projecte, s'executarà:

```
rake db:create
```

Si s'ha fet amb èxit, la instrucció anterior haurà creat una taula amb el nom `restauranteuoc_development`. Si s'ha produït error, s'hauran de comprovar els aspectes següents:

1) Que el servei de la base de dades es trobi en execució.

2) Que cap tallafocs impedeixi la comunicació amb la base de dades.

3) Que la configuració en el fitxer `database.yml` coincideix amb la configuració de la base de dades.

4) Que el projecte seleccionat en la vista Ruby Explorer és RestauranteUOC.

3.2. Gestió de reserves

El requisit principal de l'aplicació és la gestió de reserves, que ha de tenir altes, baixes i modificacions. Una reserva disposarà dels camps següents:

Camp	Tipus	Descripció
Nom	Cadena (*string*)	Nom del client que fa la reserva.
Cognoms	Cadena	Cognoms.
Telèfon	Cadena	Telèfon de contacte.
Data	Data-hora	Data i hora en què vol tenir la taula disponible.
Comensals	Numèric	Nombre de persones que hi assistiran.
Comentaris	Cadena llarga	Comentaris addicionals.

Per a disposar de gestió de reserves en l'aplicació, s'hauran de crear els elements següents:

- **Model de la reserva**: classe que representi una reserva i que també permeti fer diferents operacions com ara la cerca d'una reserva a partir d'uns criteris, actualització i eliminació de reserves, etc.

- **Vistes**: l'aplicació ha de disposar de les vistes necessàries per a visualitzar la llista de reserves, el formulari d'alta d'una reserva, etc.

- **Controladors**: gestionaran les peticions.

3.2.1. Creació d'una matriu de suport

Rails és un entorn de treball àgil i disposa de **generadors**, *scripts* que permeten crear peces de la nostra aplicació com ara vistes, models, controladors i altres elements en funció dels paràmetres proporcionats. Aquests generadors creen de manera automàtica els fitxers, classes, mètodes, codi i altres configuracions necessàries. Un generador molt útil a Rails és la matriu de suport[8], que permet crear controladors, vistes i altres configuracions sobre un model de dades determinat que es pot definir per paràmetres.

[8] En anglès, *scaffold*.

Matriu de suport

En anglès, *scaffold* significa 'bastida, estructura utilitzada per a suportar gent i materials en una construcció' o el significat molt més ampli de 'matriu'. A Rails, la matriu de suport és una manera ràpida de generar la majoria de peces per a una aplicació. Es podria considerar una estructura sobre la qual s'adapten les particularitats de l'aplicació amb pocs retocs.

La instrucció de consola per a generar una matriu de suport seria la següent:

```
script/generate scaffold [nom_del_model] [[nom_atribut:tipus]...]
```

Aptana disposa d'una vista per a executar generadors que es pot visualitzar accedint al menú Window > Show View > Generators. Una vegada oberta la vista, se seleccionarà del combo *Generator* l'element *scaffold* i en el quadre de text de paràmetres s'introduirà el següent:

```
Reserva nombre:string apellidos:string telefono:string fecha:timestamp
comensales:integer comentarios:text
```

Aquests paràmetres indiquen que s'ha de crear un model anomenat *Reserva* amb els camps *nom*, *cognoms*, *telèfon*, *data*, *comensals* i *comentaris* segons els tipus expressats *a posteriori* dels dos punts (:). Una vegada introduïts, es farà clic sobre el botó per a executar el generador. En la vista Console es mostra el progrés de la creació de la matriu.

Una vegada executada la instrucció anterior, es crearà tota la infraestructura necessària per a donar d'alta o de baixa, modificar i eliminar reserves.

3.2.2. El model Reserva

El generador ha creat el model `Reserva` declarat en el fitxer `app/models/reserva.rb`, accessible fent doble clic des de la vista Ruby Explorer.

Extensió .rb

L'extensió .rb representa un fitxer de codi Ruby.

El contingut del fitxer és el següent:

```
class Reserva < ActiveRecord::Base
end
```

Aquestes dues línies defineixen la classe `Reserva`, que hereta d'`ActiveRecord`. Aquesta herència proveeix la funcionalitat necessària per a l'accés i manipulació de reserves en la base de dades. Per convenció, es prendran els atributs de la classe a partir de les columnes de la taula `reservas`.

Mapatge d'ActiveRecord

ActiveRecord proporcionarà el mapatge entre els objectes de la classe `Reserva` i la taula `reservas` de la base de dades.

Per a crear i destruir aquesta taula, també s'ha creat la classe `CreateReservas` en el fitxer `db/migrate/create_reservas.rb`:

```
class CreateReservas < ActiveRecord::Migration
  def self.up
    create_table :reservas do |t|
      t.string :nombre
      t.string :apellidos
      t.string :telefono
      t.timestamp :fecha
      t.integer :comensales
      t.text :comentarios
      t.timestamps
    end
  end
```

```
    def self.down
      drop_table :reservas
    end
  end
```

`CreateReservas` hereta d'`ActiveRecord::Migration` i defineix dos mètodes:

La classe `CreateReservas`

`CreateReservas` conté el nom del model indicat en el generador matriu de suport i s'hi ha afegit automàticament el plural per convenció.

1) up: s'executa per a crear la taula que representa una reserva en la base de dades configurada en el projecte. Les columnes són les mateixes que es van especificar en els paràmetres del generador matriu de suport.

2) down: elimina la taula de reserves de la base de dades.

Les migracions són programes Ruby que permeten crear i alterar l'estructura de la base de dades. Per a fer la migració d'aquesta aplicació i crear la taula `reservas` en la base de dades, es pot utilitzar l'eina `rake`. Per a això, en la vista Rake Tasks, s'ha de seleccionar l'opció `db::migrate` del combo i iniciar la tasca. Després de l'execució, es pot comprovar que s'ha creat la taula de reserves amb les columnes següents:

Columna	Tipus
id	INT
nombre	VARCHAR(255)
apellidos	VARCHAR(255)
teléfono	VARCHAR(255)
fecha	DATETIME
comensales	INT
comentarios	TEXT
created_at	DATETIME
updated_at	DATETIME

També s'han afegit columnes addicionals a les indicades en la generació: `id`, `created_at` i `updated_at` s'han afegit automàticament per a donar més informació a cada registre. En particular, `id` és un identificador únic, autoincremental i clau principal de la taula. `created_at` i `updated_at` contenen la data i hora de creació i l'última actualització, respectivament.

Els atributs d'una reserva podrien estar declarats en totes dues classes: en el model `Reserva` i en la classe `ReservaMigration`, però entraria en contradicció amb el principi DRY.

3.2.3. Execució de l'aplicació

A continuació s'executarà l'aplicació per a comprovar els resultats de la generació del nou model, el controlador i les vistes. S'ha de comprovar que s'ha fet la migració mitjançant `Rake` i que el servidor està iniciat, s'ha d'accedir a l'URL següent:

Migració

Abans d'executar l'aplicació i gestionar reserves, s'haurà d'executar la migració amb l'eina `rake`.

`http://localhost:puerto/reservas`.

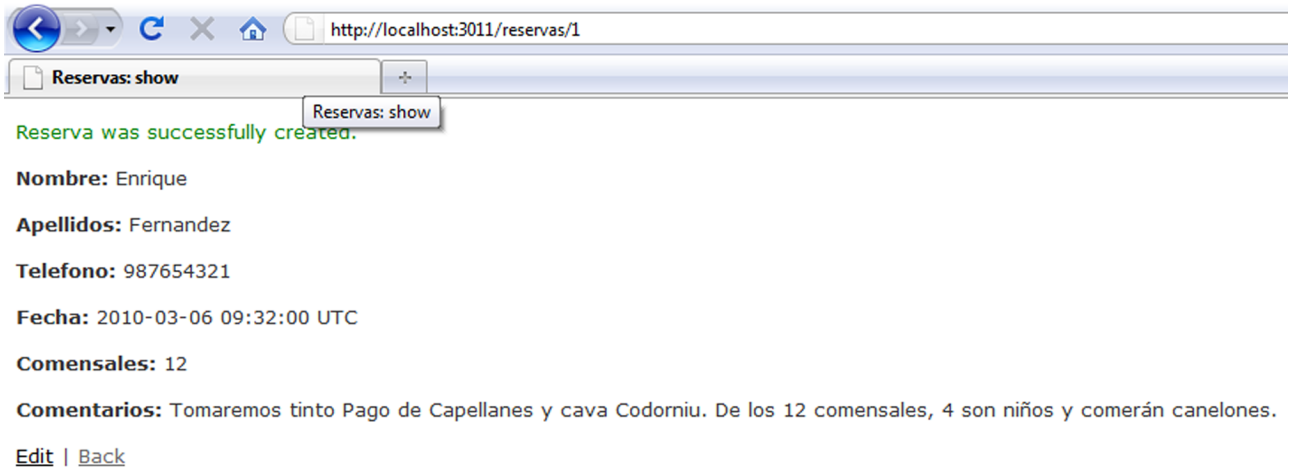
Figura 7. Pàgina de llista de reserves

La pantalla mostra una llista de reserves, que inicialment serà buida (figura 7). L'enllaç "New reserva" navega a una altra pàgina amb l'URL `http://localhost:port/reservas/new`. Aquesta altra mostra un formulari amb els camps *Nom, Cognoms*, *Telèfon*, *Data*, *Comensals* i *Comentaris* definits en executar el generador matriu de suport. En emplenar els camps i fer clic sobre "Create", es crea un registre de reserva nou en la base de dades.

Figura 8. Formulari de creació d'una reserva

En crear la reserva, es mostrarà una fitxa de la reserva en l'URL `http://localhost:port/reservas/1`. L'últim número de l'URL indica l'`id` de la reserva obtingut en crear el registre. Al peu hi ha dos enllaços: el primer, *Edit*, que torna al formulari anterior per a modificar la reserva. El segon, *Back*, torna a la llista de reserves inicial.

Figura 9. Vista d'una reserva



Si torna a la llista inicial, es pot veure el registre de reserva nou. En la mateixa fila apareixen tres enllaços: *Show*, *Edit* i *Destroy* que serveixen per a mostrar la reserva, editar-la i eliminar-la, respectivament.

Figura 10. Llista inicial amb la reserva creada recentment

Si es vol accedir a la llista de reserves amb l'URL inicial de l'aplicació, com, per exemple `http://localhost:3011`, s'haurà de modificar l'arxiu `routes.rb` dins del directori `config` afegint la línia comentada amb la indicació *nou*.

```
ActionController::Routing::Routes.draw do |map|
  ...
  map.root :controller => "reservas"      //nou
  ...
```

Aquesta línia readreça qualsevol petició dirigida a l'arrel cap al controlador `Reservas`. L'acció executada per defecte serà `index`. També s'ha d'eliminar la pàgina inicial anterior situada en el directori `public` amb el nom `index.html`. Finalment, s'haurà de reiniciar el servidor perquè els canvis tinguin efecte.

3.3. Funcionament del patró model vista controlador (MVC) en Rails

En l'apartat anterior s'ha creat una infraestructura per a la gestió de reserves composta per un model, un controlador i diferents vistes. El controlador creat per la matriu de suport disposa de les accions `index`, `show`, `new`, `edit`, `create`, `update` i `destroy`. Les accions s'executen a partir de l'URL de la petició que serà interpretada pel sistema d'encaminament, que per defecte té la configuració següent:

```
http://servidor:port/controlador/acció[?paràmetres]
```

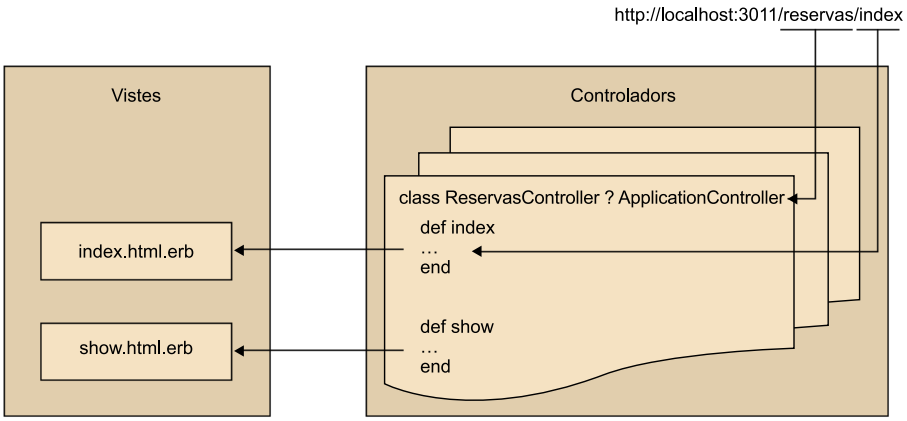
- **servidor**: indica el servidor que executa l'aplicació. Durant el desenvolupament pot referir-se la màquina local indicant `localhost` o `127.0.0.1`.
- **port**: cada aplicació Rails dins d'un servidor s'executa en un port diferent.
- **controlador**: indica quin controlador gestionarà la petició.
- **acció**: mètode del controlador que serà executat per a gestionar la resposta.
- **paràmetres**: permet enviar valors al controlador.

Formes d'encaminament

Rails permet definir altres formes d'encaminament a partir de fitxers de configuració.

La figura 11 mostra la selecció del controlador i acció a partir d'un URL.

Figura 11. Mapatge entre un URL i una acció



Codi ocult

S'ha ocultat el codi de cada mètode per a simplificar l'exemple.

Rails analitza una petició i obté el nom del controlador i l'acció de l'URL. Posteriorment, localitza el controlador a partir de tots els arxius situats en el directori de controladors que tinguin el nom corresponent. Per exemple, en el cas de les reserves, es localitzaria el fitxer `reservas_controller.rb`. Dins del fitxer es troba una classe que hereta de `ApplicationController`, que disposa d'un conjunt de mètodes:

```
class ReservasController < ApplicationController
  def index
  end

  def show
  end

     ...


  def update
  end

  def destroy
  end
```

```
end
```

El mètode executat com a resposta a la petició serà el que s'hagi especificat com a acció en l'URL. Per exemple, a partir de `http://localhost:3011/reservas/index`, s'executaria el mètode `index` de la classe `ReservasController`.

La vista associada a un mètode té, per defecte, el mateix nom que el mètode. Per exemple, si l'acció duta a terme és `index`, es mostrarà la pàgina `index.html.erb`. El controlador és responsable d'accedir al model i proporcionar les dades necessàries a les vistes.

3.3.1. ERb bàsic

ERb[9] permet afegir codi Ruby dins de pàgines HTML mitjançant un conjunt d'etiquetes. D'aquesta manera, es poden avaluar condicions, fer declaracions de variables, bucles, etc., i construir pàgines dinàmiques. L'extensió per defecte d'una pàgina HTML que disposa de codi Ruby incrustat és `.erb`. ERb forma part del mòdul ActionView.

(9) *ERb* és la sigla d'*embedded Ruby*.

Etiquetes	Descripció
`<%= %>`	Conté una expressió Ruby que serà mostrada a la part del document en què es trobi declarada l'etiqueta.
`<%= -%>`	Mostra una expressió sense un salt de línia, a diferència de l'etiqueta anterior.
`<% %>`	Conté una peça de codi Ruby que no mostra cap sortida.
`<% -%>`	Igual que l'etiqueta anterior sense el salt de línia.
`<%# %>`	És un comentari Ruby que serà ignorat i no produirà cap sortida.

En mostrar una vista, Rails genera la pàgina HTML per interpretació de les etiquetes de codi incrustat.

Generació d'una vista

L'acció `saludo` declara un conjunt de variables:

```
def saludo
    @textosaludo = 'hola mundo'
    @titulosaludo = 'Titulo saludo'
end
```

Les variables `@textosaludo` i `@titulosaludo` es declaren en el mètode del controlador i contenen una cadena de caràcters. Una vegada executat el mètode, es cridaria a la vista corresponent representada pel fitxer `saludo.html.erb`:

```
<html>
    <head>
        <title><%=@titulosaludo%></title>
    </head>
    <body>
        Saluda 5 veces: <br/>
        <% for i in 1..5 %>
            <h<%=i%>><%= @textosaludo%></h<%=i%>>
            <% end %>
        </body>
</html>
```

En generar la vista, s'interpreta el codi ERb i s'obté la sortida següent:

```
<html>
    <head>
        <title>Titulo saludo</title>
    </head>
    <body>
        Saluda 5 veces: <br/>
            <h1>hola mundo</h1>
            <h2>hola mundo</h2>
            <h3>hola mundo</h3>
            <h4>hola mundo</h4>
            <h5>hola mundo</h5>
    </body>
</html>
```

Variables d'instància

Les variables precedides del símbol @ s'anomenen *variables d'instància*. Aquestes seran visibles des de les vistes.

3.4. Creació d'un controlador i una vista

El pas següent és crear un controlador i una vista que permetin gestionar les reserves dels propers dies. S'hi accedirà a partir de l'URL següent:

```
http://localhost:puerto/proximas?dias=5
```

El paràmetre `dias` és opcional i permet especificar els dies d'antelació amb què es volen visualitzar les reserves. Si aquest paràmetre no s'especifica, es visualitzaran les reserves de les pròximes 24 hores per defecte.

Per a això, s'obrirà la vista **Generators**, se seleccionarà l'opció "Controller" del combo i s'introduirà "Proximas" dins del quadre de text. El generador haurà creat un fitxer anomenat `proximas_controller.rb` i en el seu interior la classe `ProximasController` sense cap mètode.

Dins del fitxer s'afegirà el mètode `index` i `obtenerReservas` dins de la classe `ProximasController`, amb el codi següent:

Mètode `index`

El mètode `index` s'executa per defecte si l'URL sol defineix el controlador.

```
def obtenerReservas
    @dias = params[:dias] || 1
    inicio = DateTime.now
    fin = @dias.to_i.days.from_now
    @reservas = Reserva.all(:conditions => ['fecha >= ?
    && fecha <= ?',
    inicio.to_s(:db), fin.to_s(:db)])
end

def index
    obtenerReservas
end
```

El mètode `obtenerReservas` fa el següent:

1) Obté de l'URL el paràmetre dies en la variable `@dias`. En cas que el paràmetre no existeixi, s'establirà 1 dia per defecte.

2) Crea una segona variable, anomenada `inicio`, amb la data i hora actuals com a valor.

3) Crea una tercera variable, anomenada `fin`, en què s'estableix la data resultant de sumar a la data en curs els dies obtinguts per paràmetre.

4) Finalment, es declara una quarta variable, anomenada `@reservas`, que contindrà les reserves en la qual la data estigui compresa entre la data d'inici i la data final. El conjunt de reserves s'obté a partir de la classe `Reserva`. El mètode `all` s'hereta d'`ActiveRecord` i permet obtenir tots els registres. Aquest mètode també permet establir condicions; en aquest cas, s'ha indicat que el camp `data` de la taula de reserves sigui superior a la data d'inici i inferior a la data final. El resultat és un conjunt de reserves amb data anterior als dies indicats per paràmetre en l'URL.

El mètode `index` crida el mètode `obtenerReservas`. `Index` és el mètode per defecte del controlador, per la qual cosa no cal indicar l'acció en l'URL, simplement `http://localhost:puerto/proximas`.

Ara caldrà crear la vista relacionada amb el mètode `index` per a visualitzar la llista de reserves contingudes en la variable `@reservas`, definida en el controlador que acabem de crear. Per a això, s'utilitzaran unes subplantilles anomenades **`partials`**, que permeten reusar codi entre vistes.

Partials

Els `partials` són vistes ERb. S'identifiquen fàcilment perquè els fitxers que els contenen comencen pel símbol "_".

En aquest cas, es crearà una subplantilla per a mostrar la llista de reserves. Per a això es farà el següent:

1) Accedir a la vista RubyExplorer i accedir a la carpeta `/app/views/próximas`.

2) Fer clic contextual i seleccionar New > ERB/RHTML file.

3) En el formulari mostrat a continuació, establir el nom del fitxer com `"_mostrarReservas.html.erb"` en la caixa de text *File name*.

4) Fer clic en *Finish*.

Dins del fitxer creat, s'introduirà el codi següent:

```
<style type=''text/css''>
    div.Item {
        width:600px;
        border:1px solid gray;
        background-color:#FFF8F0;
        margin:10px ;
        text-align:left;
    }
</style>

<% @reservas.each do |reserva|%>
    <div class='Item'>
        <table>
            <tr>
                <td><b>Nombre</b></td>
                <td><%= reserva.nombre %></td>
            </tr>
            <tr>
                <td><b>Apellidos</b></td>
                <td><%= reserva.apellidos %></td>
            </tr>
            <tr>
                <td><b>Telefono</b></td>
                <td><%= reserva.telefono %></td>
            </tr>
            <tr>
                <td><b>Fecha y hora</b></td>
                <td><%= reserva.fecha.to_s(:short) %></td>
            </tr>
            <tr>
                <td><b>Comensales</b></td>
                <td><%= reserva.comensales %></td>
```

```
            </tr>
            <tr>
                <td><b>Comentarios</b></td>
                <td><%= reserva.comentarios %></td>
            </tr>
        </table>
    </div>
<% end %>
```

L'etiqueta `<style>` defineix un estil per als elements `<div>` de la pàgina. Posteriorment, amb codi Ruby incrustat, es recorren totes les reserves. Cadascuna està continguda en un `<div>` que aplica l'estil anterior. Dins de la divisió, es defineix una taula que accedeix als diversos membres i mostra tota la informació d'una reserva.

Ara falta crear la vista relacionada amb el mètode `index` del controlador `proximas` que utilitzarà el `partial` creat. La vista s'ha de crear en el directori `/app/views/proximas` amb el nom `index.html.erb` i el contingut següent:

```
<%= render :partial => 'mostrarReservas' %>
```

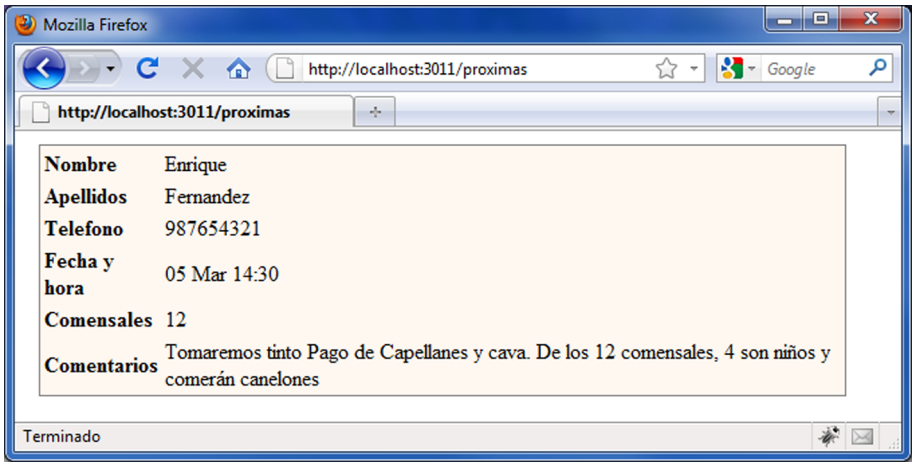
Quan s'executi el mètode `index` del controlador, es renderitzarà el fitxer `index.html.erb`. Aquest, crida a `partial _mostrarReservas` i mostra tots les reserves contingudes en la variable `@reservas` proporcionada pel controlador.

Per a provar la nova acció, és recomanable reiniciar el servidor des de la vista Servers i accedir a l'URL `/proximas`. Cal assegurar que s'han desat els canvis en tots els fitxers modificats.

No cal indicar l'acció `index`, ja que s'executarà per defecte en no indicar l'acció en l'URL.

Perquè aparegui algun resultat, s'ha de crear una reserva per a abans de les pròximes 24 hores.

Figura 12. Llista de les pròximes reserves



3.5. Formats

Els formats (*layouts*) són vistes que contenen codi HTML comú per a un conjunt de vistes relacionades amb un controlador. En general, mostren una capçalera i un peu de pàgina i defineixen una secció en el seu interior en què es mostrarà la vista en curs.

Formats

Els formats s'emmagatzemen en el directori `/app/views/layouts`.

Els formats s'utilitzen generalment per a mostrar elements comuns per a un conjunt d'accions: menú, capçalera i peu de pàgina, etc.

Per a mostrar una capçalera i un peu de pàgina en totes les vistes associades amb el controlador `Reservas`, es modificarà la plantilla `reservas.html.erb` del directori de formats amb el contingut següent:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <title>Reservas: <%= controller.action_name %></title>
</head>
<body>
    <h2 style="color: blue" align="center"> Restaurante UOC - Gestión de reservas</h2>
    <hr width="100%" align="center"/>
    <%= yield %>
    <hr width="100%" align="center"/>
    <p style="size: 8; color: blue">Universitat Oberta de Catalunya</p>
</body>
</html>
```

Aquesta plantilla mostra el nom del restaurant i un peu de pàgina. Dins del cos hi ha l'etiqueta `<%= yield %>` que serà substituïda per la vista de cada acció. El resultat d'accedir a la pàgina `/reservas/index` és el següent:

Reinici de servidor

Pot ser necessari reiniciar el servidor per a visualitzar els nous canvis en la plantilla.

Figura 13. Acció `index` del controlador reserves amb la plantilla modificada

3.6. AJAX

Rails utilitza internament l'entorn de desenvolupament Prototype per a l'ús d'Ajax. Com a conseqüència, una pàgina utilitza AJAX haurà d'incorporar l'etiqueta següent:

Prototype

Prototype és una biblioteca escrita en JavaScript per a facilitar el desenvolupament de pàgines web dinàmiques amb AJAX. Ofereix funcions que abstreuen les diferències entre les implementacions del DOM pels navegadors.

```
<%= javascript_include_tag "prototype" %>
```

Mentre es gestionen reserves, seria interessant conèixer si una reserva és pròxima perquè es pugui preparar el servei. Per a obtenir aquesta informació, es modificarà el format del controlador `Reservas` afegint un procés que comprova periòdicament amb AJAX si hi ha alguna reserva pròxima. En cas que n'hi hagi alguna, es mostrarà un enllaç a la pàgina que les visualitza.

Obtenció de notificacions

En afegir una funció AJAX en el format de Reservas, s'obtindran notificacions sobre noves reserves en qualsevol vista del controlador en la qual estiguem situats.

La plantilla `app/views/layouts/reservas.html.erb` modificada anteriorment quedaria de la manera següent:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type" content="text/html;charset=UTF-8" />

        <%= javascript_include_tag "prototype" %>
        <%= periodically_call_remote(
            :url => {:controller => 'proximas',
            :action => :existeReserva, :dias => '1'},
            :frequenc => '5',
            :update => 'notificacionReserva') %>
        <title>Reservas: <%= controller.action_name %></title>
</head>
```

```
<body>
    <h2 style="color: blue" align="center"> Restaurante UOC - Gestión de reservas</h2>
    <div id="notificacionReserva"></div>
    <hr width="100%" align="center"/>
    <%= yield %>
    <hr width="100%" align="center"/>
    <p style="size: 8; color: blue">Universitat Oberta de Catalunya</p>
</body>
</html>
```

El codi afegit al format del controlador `Reservas` consta dels elements següents:

1) `<%= javascript_include_tag "prototype" %>`: importa la biblioteca JavaScript Prototype per a l'ús d'AJAX.

2) `<%= periodically_call_remote(...)%>`: defineix una function AJAX que fa peticions asíncrones de manera periòdica. Dins d'aquesta funció, es defineixen les característiques següents:

- `:url =>`: especifica el controlador, acció i paràmetres de la petició. En aquest cas, el controlador serà `proximas`, l'acció serà `existeReserva` i dias és un paràmetre passat per l'URL.

- `:frequenc =>`: indica la freqüència de consulta en segons. En aquest cas, cada 5 segons es comprovarà si hi ha pròximes reserves.

- `:update =>`: indica la divisió en què es mostrarà el resultat de cada petició.

3) `<div id="notificacionReserva"></div>`: divisió que contindrà la resposta de les peticions periòdiques.

La modificació anterior fa peticions asíncrones cada 5 segons a una acció del controlador `proximas` i envia com a paràmetre en l'URL els dies d'avís. Ara, s'ha de definir el mètode `existeReserva` al controlador `proximas`:

```
def existeReserva
    dias = params[:dias] || 1
    inicio = DateTime.now
    fin = dias.to_i.days.from_now
    @number = Reserva.count(:conditions =>
    ['fecha >= ? && fecha <= ?', inicio.to_s(:db), fin.to_s(:db)]) || 0
    render :layout => false
end
```

Aquest mètode obté només el nombre de reserves entre la data en curs i els pròxims dies passats per paràmetre. Aquest nombre s'emmagatzema en la variable `@number` i serà utilitzat per la vista `existeReserva.html.erb`, que es crearà a continuació. L'última instrucció indica que no s'ha de renderitzar cap format per a aquesta acció. La crida feta és asíncrona i la pàgina retornada s'incorporarà dins d'una pàgina ja existent. Altrament, es mostraria el format i la resposta d'aquesta acció dins de la divisió `notificaciónReserva`.

La vista associada al mètode `existeReserva` haurà de retornar una informació o una altra segons el nombre obtingut. Si no hi ha reserves, es mostrarà un text informatiu. Si n'hi ha alguna, haurà de mostrar un enllaç cap a `/proximas` per a visualitzar-les. Ara s'ha de crear la vista `existeReserva.html.erb` dins del directori `/app/views/proximas` i escriure el contingut següent:

```
<% if @number == 1 %>
    <%= link_to ('Hay 1 reserva en las próximas 24 horas',
        {:action => 'index', :dias => '1'},:popup =>
        ['new_Window','height=600, vwidth=650,scrollbars=yes']) %>
<% elsif @number > 1 %>
    <%= link_to ("Hay #{@number} reservas en las próximas 24 hores",
        {:action => 'index', :dias => '1'},:popup =>
        ['new_Window','height=600, width=650,scrollbars=yes']) %>
<% else %>
    No hay reservas en las próximas 24 horas.
<% end %>
```

La vista avalua el nombre de reserves obtingut pel controlador. El text de sortida d'aquesta vista pot ser alguna d'aquestes tres possibilitats:

1) Si el nombre de reserves pròximes és 1, es retorna un enllaç que obre una finestra emergent cap a `proximas/index`, que mostrarà la llista de la pròxima reserva.

2) Si el nombre de reserves és més gran que 1, es retornarà un enllaç semblant a l'anterior però amb el text en plural, ja que hi ha més d'una reserva.

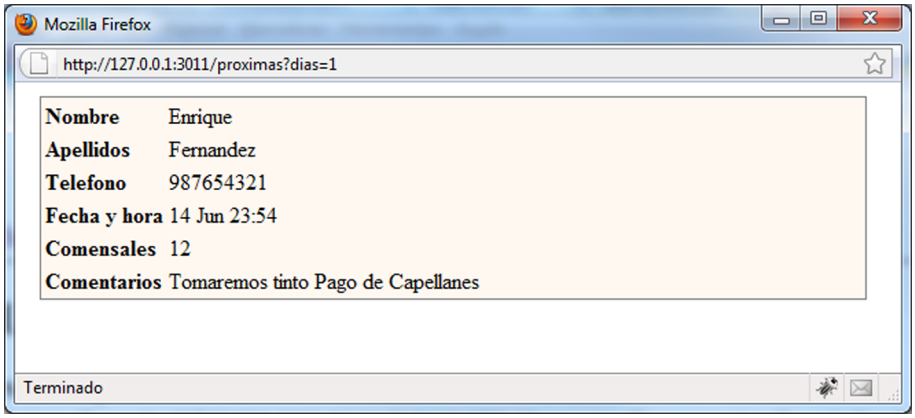
3) Si no es compleix cap de les dues anteriors, es mostra un text que indica que no hi ha reserves en les pròximes 24 hores.

Per a comprovar el format nou, el mètode nou i la vista nova, s'hauran de desar els fitxers editats i obrir l'URL `/reservas` de l'aplicació. Si hi ha una reserva en les pròximes 24 hores, la vista principal de gestió de reserves mostrarà un enllaç com el de la figura 14.

Figura 14. Actualització asíncrona de la pàgina que notifica que hi ha una reserva pròxima

En clicar sobre l'enllaç, s'obrirà una finestra emergent que mostrarà la reserva.

Figura 15. Llista de pròximes reserves en una finestra emergent



3.6.1. Format per a les pròximes reserves

El pas següent és modificar el format del controlador `Proximas` perquè la finestra emergent (com la de la figura 15) mostri el següent:

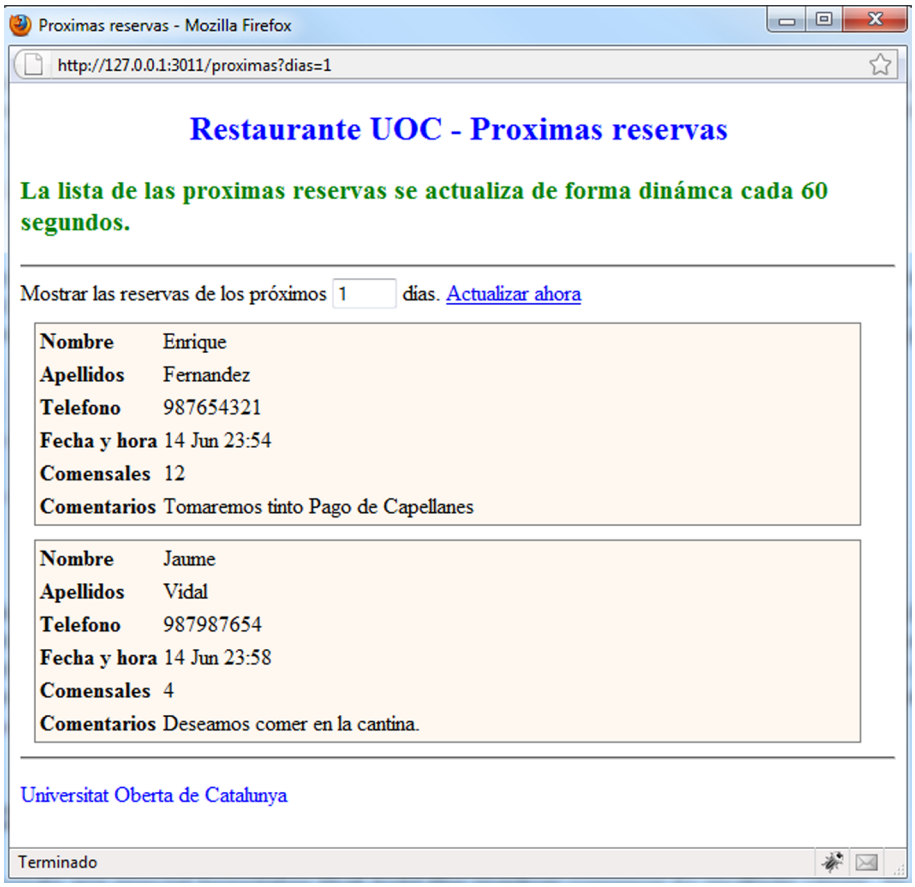
- Informació sobre el restaurant, amb una capçalera i un peu de pàgina, similar al format del controlador `Reservas`.

- Informació sobre la llista de pròximes reserves de manera automàtica, periòdica i asíncrona amb AJAX.

- Un quadre de text per a poder indicar el rang de dies de pròximes reserves.

La finestra emergent es mostrarà de la manera següent:

Figura 16. Finestra emergent amb el format modificat



Aquesta vista s'actualitzarà a mesura que passi el temps amb les reserves que s'apropin a les pròximes 24 hores o el dia indicat en el camp de text.

El primer pas és crear un nou mètode dins del controlador `Proximas` que retorni la llista de les pròximes reserves:

```
def buscarNuevas
    obtenerReservas
    render :layout> => false
end
```

Igual que el mètode `index`, `buscarNuevas` crida a `obtenerReservas` per a retornar a la vista les variables `@dias` i `@reservas`, que contenen el nombre de dies de la petició i el conjunt de reserves obtingudes de la base de dades. La instrucció `render :layout => false` evita que la vista retornada per la crida asíncrona mostri el format definit per al controlador `Proximas`, ja que es tracta d'una crida asíncrona i la vista obtinguda s'incorporarà dins d'una divisió.

També s'haurà de crear una vista dins del directori associada a l'acció `buscarNuevas` en el directori `/app/views/proximas` anomenada `buscarNuevas.html.erb`. El contingut del fitxer serà exactament igual que `index.html.erb`:

```
<%= render :partial => 'mostrarReservas' %>
```

Ara, es crearà el fitxer `proximas.html.erb` dins de la carpeta `/app/views/layouts` per a definir el format del controlador `Proximas`, que disposarà del contingut següent:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type" content="text/html;charset=UTF-8"/>
    <%= javascript_include_tag "prototype" %>
    <%= periodically_call_remote( :url => {:action => :buscarNuevas},
        :with => "'dias=' + $('numDias').value",
        :frequency => 60,
        :update => 'notificacionReserva') %>

    <title>Proximas reservas</title>
</head>
<body>
    <h2 style="color: blue" align="center">
        Restaurante UOC - Proximas reservas
    </h2>
    <h3 style="color: green" align="left">
        La lista de las próximas reservas se actualiza de manera dinámica cada 60 segundos.
    </h3>
    <hr width="100%" align="center"/>
        Mostrar las reservas de los próximos
    <%= text_field_tag :numDias,"#{@dias}", :maxlength => 3, :size => 3 %> días.
    <%= link_to_remote ('Actualizar ahora',
        :url => {:action => :buscarNuevas},
        :with => "'dias=' + $('numDias').value",
        :update => 'notificacionReserva') %>
    <div id="notificacionReserva">
        <%= yield %>
    </div>
    <hr width="100%" align="center"/>
    <p style="size: 8; color: blue">Universitat Oberta de Catalunya</p>
</body>
</html>
```

Les etiquetes més destacables del codi anterior són:

- `<%= text_field_tag :numDias,...%>`: mostra un quadre de text que inicialment contindrà el nombre de dies obtingut per la variable `@dias`. A aquest control, se li estableix l'identificador `numDias`. La llista de pròximes reserves s'actualitzarà prenent els dies indicats en aquest control.

- `<% periodically_call_remote ... %>`: funció AJAX que fa consultes periòdiques. Cada minut actualitza la llista de pròximes reserves continguda dins de la divisió `notificacionReserva`. L'acció duta a terme és `buscarNuevas` del controlador `Proximas`.

- `:with`: afegeix un paràmetre a l'URL amb el nombre de dies i el valor s'estableix a partir del quadre de text declarat anteriorment.

- `<%= link_to_remote... %>`: mostra un enllaç amb el text "Actualitzar ara" i, en fer-hi clic, actualitza la llista de reserves d'una manera asíncrona. El resultat és el mateix que la funció AJAX anterior, llevat que aquesta es fa manualment.

Amb aquestes modificacions, es mantindrà la llista actualitzada amb les pròximes reserves a partir del valor del quadre de text. Si es volguessin obtenir les reserves pròximes a una setmana, n'hi hauria prou de canviar el valor del quadre de text a 7.