Contabilità domestica: un'introduzione al web development con ruby on rails

[Railsgirls Udine - 5 e 6 giugno 2014]

Cosa e come:

Al termine di questo tutorial avremo sviluppato l'architettura di base di una semplice applicazione per gestire la contabilità domestica in base ad uno schema che prevede un registro di entrate e uscite distinte in categorie.

Sarà possibile inserire categorie per entrate o uscite e registrare le singole voci di registro associandole alla categoria di riferimento.

Sarà possibile inoltre visualizzare separatamente un riassunto delle entrate e delle uscite divise per categoria accompagnato da un grafico a torta per una rappresentazione più chiara del rapporto tra le varie categorie di spesa/ricavo.

Potremo 'assaggiare' ruby/ruby on rails e anche un pizzico di javascript, che utilizzeremo per i grafici grazie alla libreria visualize fornita come plugin per jquery.

Va da sè che si tratta di un'abbozzo di applicazione, invito chi fosse interessato a completarla e proporre eventuali direzioni di sviluppo.

A chi è diretto questo tutorial:

Questo documento è pensato principalmente in funzione dell'evento railsgirls che si terrà a Udine il 5 e 6 giugno 2014. E' accessibile ad un'utenza con nozioni elementari di sviluppo web.

Come leggerlo:

- I comandi da inserire direttamente nella shell/prompt dei comandi (il metodo consigliato è fare copia e incolla) sono indicati in ***grassetto corsivo*** come nell'esempio qui sotto

cd contabilita_domestica

- le porzioni di codice sono visualizzate come l'esempio che segue

```
  <div class="control-group">
    <%= f.label :category_id, :class => 'control-label' %>
    <div class="controls">
      <%= f.collection_select :category_id, Category.all, :id, :nome,
prompt: false %>
    </div>
  </div>
```

Prerequisiti:

Devono essere installati sul sistema

- Ruby 1.9
- rails
- un editor di testo avanzato (sublime text - notepad++ - textmate) o un'ide (aptana studio - rubymine)

Cominciamo !

Predisponiamo una directory per i nostri progetti rails

Dopo aver scelto la posizione nel filesystem in cui vogliamo conservare i nostri progetti in rails, creiamo la directory che li conterrà

mkdir railsgirls

e spostiamoci al suo interno

cd railsgirls

Lasciamo che rails costruisca lo scheletro del sistema

Ruby on rails è un framework evoluto, che offre una serie di comandi shell grazie ai quali predisporre un'applicazione è davvero semplice e rapido.

rails new contabilita_domestica

Il comando appena eseguito genera un output piuttosto interessante: rails ha creato per noi tutti i file necessari allo sviluppo della nostra prima applicazione e la struttura delle directory del progetto.

> *MVC: un pattern architetturale in grado di separare la logica di presentazione dei dati dalla logica di business.*
> *Guardate dentro la cartella app, rails ha creato una directory chiamata* ***models****, una directory chiamata* ***views*** *e una directory chiamata* ***controllers***

(unix/linux/osx) ***cd contabilita_domestica/app***

(windows) ***cd contabilita_domestica\app***

Configuriamo l'applicazione affinchè includa le librerie che ci serviranno per il progetto

Per la formattazione di una pagina web (un'applicazione web è composta di varie pagine) si utilizza una combinazione di HTML per il markup, la definizione della struttura semantica della pagina (distinguere titoli, paragrafi, tabelle ecc.) e di CSS per la formattazione effettiva dei vari elementi della pagina (definizione dei colori, font, impaginazione degli elementi ecc.)

Da qualche anno hanno preso piede sul web dei framework CSS che incorporano soluzioni consolidate per i problemi più comuni legati alla creazione di una pagina web.

In questa sede intendiamo utilizzare **bootstrap**, un framework rilasciato da alcuni sviluppatori di twitter, che ci permetterà di ottenere con poche operazioni un layout ordinato e facilmente personalizzabile.

> ***RubyGems*** *è un gestore di pacchetti per il linguaggio di programmazione Ruby che fornisce un formato standard, per distribuire i programmi e le librerie scritti in Ruby, chiamato gems (dall'inglese: gemme) è inoltre uno strumento progettato per facilitare la gestione dell'installazione delle "gemme" e per la loro distribuzione.*

L'ecosistema di ruby on rails è in costante evoluzione, e molte librerie vengono aggiornate con frequenza. Questo espone i programmi scritti con una determinata versione al rischio di incompatibilità e potenziali errori futuri.

rails utilizza un sistema di gestione delle librerie o gemme chiamato bundler, che permette di definire per ciascuna libreria la versione esatta richiesta, per mitigare il problema.

L'esecuzione di bundler si basa su un file di configurazione, chiamato **Gemfile** che si trova nella directory root del progetto

---

Apriamo Gemfile con l'ide/editor

Decommentiamo la riga

```
# gem 'therubyracer', platforms: :ruby
```

che diventa così

```
gem 'therubyracer', platforms: :ruby
```

Aggiungiamo le seguenti righe

```
# aggiungo twitter bootstrap 3 e relativi generators
gem 'bootstrap-sass'
gem 'bootstrap-generators'
```

A questo punto dobbiamo installare le librerie nella nostra applicazione, e lo facciamo utilizzando bundler con il comando

bundle install

Consiglio di leggere l'output del programma, come spesso accade con rails è molto informativo.

Una volta completata l'installazione delle gemme, **bootstrap** è disponibile e utilizzabile.

Dicevo che rails dispone di diversi strumenti richiamabili da linea di comando che rendono la scrittura di un'applicazione divertente. Per la nostra applicazione ci faremo aiutare da un'utilità che nel gergo di rails si chiama generator. Un comando che genera automaticamente lo scheletro dei file che compongono l'applicazione (per ora magari credetemi sulla parola :smile:)

Quello che manca per integrare completamente bootstrap nella nostra applicazione è la riconfigurazione automatica dei generators, che otteniamo digitando il comando

rails g bootstrap:install

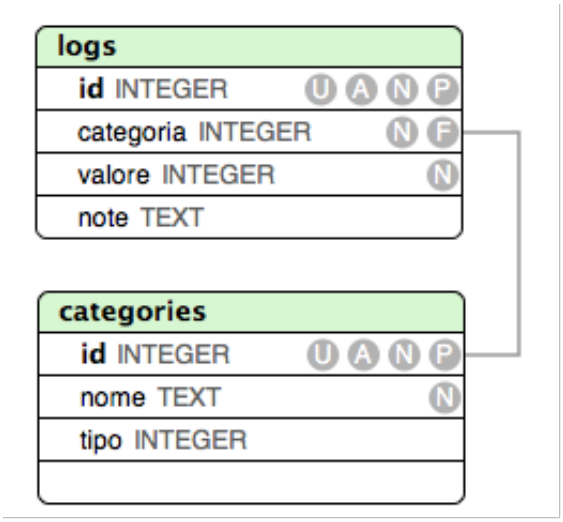
Due parole sul database

La nostra applicazione si basa su un database molto semplice, composto di due tabelle, categories e logs. La tabella categorie ha un campo denominato 'tipo' che è booleano, ovvero può assumere solo due valori, vero o falso.

La utilizzeremo per determinare se la categoria sia di entrata (true) o di uscita (false)

La tabella logs contiene una chiave esterna, denominata 'category_id', che associa ogni riga di registro ad una categoria. Utilizzeremo sqlite, che è il database di default per le applicazioni rails

(*il programma che ho utilizzato per generare il diagramma sotto non mi permetteva di impostare un campo booleano*)



rails utilizza un sistema per gestire la struttura del database in modo da incorporarla nell'applicazione, basata su file in ruby definiti migrations. Le migrazioni incorporano nel nome del file la data quindi tramite questi file è possibile tener traccia delle modifiche effettuate sul db nel corso del tempo.

Lasciamo che rails abbozzi la struttura dell'applicazione

Bene.

Siamo pronti per cominciare a lavorare sulla struttura reale dell'applicazione.

Di cosa abbiamo bisogno ? Dal punto di vista logico pare che abbiamo essenzialmente due entità, il registro e l'elenco delle categorie. In base al MVC per ciascuna di esse dovremmo generare

- un model
- una view
- un controller

Ci viene in aiuto il comando **rails generate scaffold**, che ci permette di creare questi elementi, ma non si ferma qui. Tra le altre cose genera anche le migrations necessarie a creare concreatamente il database. Specifichiamo la struttura della tabella come parametri al comando

rails g scaffold Category nome:string tipo:boolean data:date

rails g scaffold Log valore:integer note:text category:references

(*notate la sintassi utilizzata per definire una chiave esterna*)

E siamo finalmente pronti a generare il database con il comando

rake db:migrate RAILS_ENV=development

(*utilizziamo la configurazione di default definita in* **config/database.yml**)

Configurazione e aggiustamenti

rails è magico.

L'applicazione comincia ad avere una forma, le linee di codice effettivamente scritte sono due, e riguardano le librerie utilizzate.

Potremmo anche avviare il server e vedere come si presenta la nostra applicazione, ma prima vediamo di aggiungere alcuni parametri di configurazione e modificare quanto serve la presentazione dei dati.

Il file **config/routes.rb** serve per indicare a rails la corrispondenza tra una url come inserita nella barra dell'indirizzo e la coppia controller/funzione a cui questa fa riferimento

Apriamo **config/routes.rb** ed inseriamo le righe

```
# alla url 'entrate' corrisponde la funzione entrate nel controller logs
get 'entrate' => 'logs#entrate'

# alla url 'uscite' corrisponde la funzione uscite nel controller logs
get 'uscite' => 'logs#uscite'

# la pagina principale del progetto corrisponde alla funzione index del
controller logs
root 'logs#index'
```

E' necessario ora creare le funzioni menzionate nel paragrafo precedente all'interno del controller **app/controllers/logs_controller.rb** e per ciscuna di esse preparare la vista corrispondente nella cartella **app/views**

Apriamo il file **app/controllers/logs_controller.rb**

sostituiamo la funzione **index** copiando il riquadro che segue

```
  def index
    @logs = Log.all
    @totale = 0
    @logs.each{|riga| riga.category.tipo ? @totale += riga.valore : @totale
-= riga.valore}

    # oppure così
    #
    # @logs.each{ |riga|
    #   if riga.category.tipo == true then @totale = @totale + riga.valore
    #   else @totale = @totale - riga.valore
    #   end
    # }
  end
```

creiamo le funzioni **entrate** e **uscite** copiando il riquadro qui sotto

```
  def entrate
    t = 'Entrate'
    @entrate = Log.includes(:category).where('categories.tipo = ?',
true).group("categories.nome").sum(:valore)
  end

  def uscite
    t = 'Uscite'
    @uscite = Log.includes(:category).where('NOT categories.tipo = ?',
true).group("categories.nome").sum(:valore)
  end
```

Ora le viste:

Modifichiamo il file **app/assets/stylesheets/application.css.scss** inserendo la riga che indica a rails di includere gli stili di bootstrap

```
@import "bootstrap";
```

Creiamo il file **app/views/logs/entrate.html.erb** con il seguente contenuto

```
  <%- model_class = Log -%>
  <div class="page-header">
    <h1>Entrate</h1>
  </div>
  <script type="text/javascript">
    // il javascript per il grafico
    $(function() {
      $('table#entrate').visualize({
        type: 'pie',
        width: '450px',
        height: '300px',
        pieMargin: 10
      });
    });
  </script>
  <table class="table table-striped" id="entrate" style="margin-bottom:
40px">
    <thead>
      <tr>
        <th><%= model_class.human_attribute_name(:category_id) %></th>
        <th><%= model_class.human_attribute_name(:valore) %></th>
      </tr>
    </thead>
    <tbody>
      <% @entrate.each do |k,v| %>
        <tr>
          <th><%= k  %></th>
          <td><%= v %></td>
        </tr>
      <% end %>
    </tbody>
  </table>
```

e creiamo il file **app/views/logs/uscite.html.erb** con il seguente contenuto

```
  <%- model_class = Log -%>
  <div class="page-header">
    <h1>Uscite</h1>
  </div>
  <script type="text/javascript">
    $(function() {
      $('table#uscite').visualize({type: 'pie', width: '450px', height:
'300px', pieMargin: 10});
    });
  </script>
  <table class="table table-striped" id="uscite" style="margin-bottom:
40px">
    <thead>
      <tr>
        <th><%= model_class.human_attribute_name(:category_id) %></th>
        <th><%= model_class.human_attribute_name(:valore) %></th>
      </tr>
    </thead>
    <tbody>
      <% @uscite.each do |k,v| %>
        <tr>
          <th><%= k  %></th>
          <td><%= v %></td>
        </tr>
      <% end %>
    </tbody>
  </table>
```

Modifichiamo il layout generale del'applicazione per utilizzare un'impaginazione a due colonne, gazie a bootstrap è un'operazione rapida e semplice. Nella colonna di destra potremo inserire del testo esplicativo

Apriamo il file **app/views/layouts/application.heml.erb** emodifichiamolo in modo che corrisponda al riquadro sotto

```
  <!DOCTYPE html>
  <html>
  <head>
    <meta charset="utf-8">
    <title>ContabilitaDomestica</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, user-scalable=no">
    <meta name="description" content="">
    <meta name="author" content="">

    <!-- HTML5 shim and Respond.js IE8 support of HTML5 elements and media
queries -->
    <!--[if lt IE 9]>
```

```
      <%= javascript_include_tag
"https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js",
"https://oss.maxcdn.com/libs/respond.js/1.3.0/respond.min.js" %>
    <![endif]-->

    <%= stylesheet_link_tag    "application", media: "all", "data-
turbolinks-track" => true %>
    <%= javascript_include_tag "application", "data-turbolinks-track" =>
true %>
    <%= csrf_meta_tags %>
  </head>
  <body>
    <div class="navbar navbar-inverse navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <%= link_to "Contabilità Domestica", "#", :class => "navbar-brand"
%>
        </div>
        <div class="collapse navbar-collapse">
          <ul class="nav navbar-nav">
            <li><%= link_to "Entrate", 'entrate' %></li>
            <li><%= link_to "Uscite", 'uscite' %></li>
            <li><%= link_to "Categorie", categories_path %></li>
            <li><%= link_to "Registro", logs_path %></li>
          </ul>
        </div>
      </div>
    </div>
    <div class="container">
      <% flash.each do |name, msg| %>
        <%= content_tag :div, :class => "alert alert-#{ name == :error ?
"danger" : "success" } alert-dismissable" do %>
          <button type="button" class="close" data-dismiss="alert" aria-
hidden="true">&times;</button>
          <%= msg %>
        <% end %>
      <% end %>
      <div class="row">
        <div class="col-md-8"><%= yield %></div>
        <div class="col-md-1"></div>
        <div class="col-md-3">
          <h2>Come funziona</h2>
          <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
```

```
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex
ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id
est laborum.</p>
        </div>
      </div>
    </div>
  </body>
  </html>
```

Per visualizzare i grafici non ci resta che copiare

- il file **visualize.jQuery.js** all'interno della cartella **app/assets/javascripts**
- il file **visualize.css** all'interno della cartella **app/assets/stylesheets**

Sarà rails ora ad includere i file nel progetto

(*notare le righe che contengono* **'*require_tree .*'** *in* **app/assets/javascripts/application.js** *e* **app/assets/stylesheets/application.css.scss**)

... rullo di tamburi ...

Digitiamo

rails s

apriamo il browser e visitiamo la pagina **http://localhost:3000/**

Se tutto è andato a buon fine dovremmo trovarci davanti alla nostra prima applicazione in ruby on rails, con tanto di twitter bootstrap e jquery visualize

Complimenti e grazie !

:feet:

:see_no_evil: :hear_no_evil: :speak_no_evil: