Ruby on Rails: La prima vera applicazione (Parte 2)

Progetto di Sistemi Web

AA 2025/2026

Filippo Poltronieri filippo.poltronieri@unife.it

Edoardo Di Caro edoardo.dicaro@unife.it

Action_text e Active Storage per la gestione di Contenuti

- Fino a questo momento, la nostra applicazione è molto sempice, non sono presenti immagini.
- Per la gestione di `attachements` è possibile utilizzare `action_text` che oltre a fornire un editor integrato, include anche `active_storage`
- Utilizzando `active_storage`, la gestione delle immagini, upload, caricamento, ridimensionamento è altamente semplificata.

Action_text

- Installiamo `action_text` nella nostra applicazione e proviamo a dare la possibilità un'immagine di presentazione del film.

```
bin/rails action_text:install
```

```
Installing JavaScript dependencies
LENGTHY OUTPUT ..........
VEDIAMOLO ASSIEME
```

Action_text

- Possiamo vedere ha aggiunto delle componenti javascript e ci viene indicato di utilizzare `bundle install` per completare le parti mancanti dell'installazione.

`$ bundle install` --> per installare le gemme di image_processing

`$ bin/dev` --> per riavviare l'applicazione

- Va inoltre eseguita la migration per apportare le modifiche al DB.

Action_text : Models

- A questo punto possiamo procedere ad aggiungere un'immagine alla risorsa `movie`
- Per fare questo dobbiamo modificare il model di `movie`, specificando (con ActiveRecord), che il film ha una immagine allegata.

```ruby
# app/view/models/movie.rb
class Movie < ApplicationRecord
  has_many :castings
  has_many :actors, through: :castings
  has_one_attached :poster
end
```

Action_text : Views

- Dobbiamo modificare le rispettive view e controller per consentire l'upload.
- Ci basterà aggiungere l'helper per l'upload del file e la rispettiva visualizzazione
- Cosa va specificato nel controller?

```
# app/view/movies/_form.html.erb

  <div>
    <%= form.label :poster, style: "display: block" %>
    <%= form.text_field :poster, accept: "image/*" %>
  </div>
```

Esercizio

Come esercizio, proviamo a implementare la stessa cosa anche per la risorsa `actor`

- Modificare il model di `actor` per consentire l'upload di una immagine
- Modificare le rispettive view e controller per consentire l'upload e la visualizzazione dell'immagine
- Testare l'upload di immagini per entrambe le risorse `movie` e `actor`

Autenticazione - 1

- L'autenticazione è un **aspetto fondamentale di qualsiasi applicazione web moderna**
- A differenza delle versioni precedenti del framework, Rails 8 fornisce direttamente un layer di authentication, che sfrutta un generatore opportuno
- In passato venivano utilizzate gemme come `devise`, ora deprecate e non più utilizzate
- Rails ci semplifica il processo di autenticazione, creandoci i rispetti model per gestire la sessione mediante funzionalità di `login` e `logout`
- Il framework non fornisce tuttavia le funzionalità per implementare la registrazione, **che vanno implementate autonomamente**

Autenticazione - 2

- Il layer di autenticazione crea una tabella per l'utente `user`
- Possiamo sfruttare questo nuovo model per aggiungere delle informazioni, come l'anagrafica dell'utente, un username e sfruttare queste informazioni all'interno dell'applicativo
- Per esempio, una volta aggiunte le funzionalità dell'autenticazione, possiamo sfruttare questa opzione per aggiungere delle review ai film.

Hands-on Autenticazione

`$ bin/rails generate authentication`

```
      invoke  erb
         --- LENGTHY OUTPUT ---
```

- Dall'output possiamo vedere come il generator ha creato le rispettive view per la gestione delle `password`
- Vengono creati 3 nuovi model: `session` , `user` , `current`
- `current` ci permette di trovare le informazioni dell'utente corrente
- Vengono create i rispetti controllers, i mailers, e inserite delle regole nell'application controller
- Oltre a eseguire la migration, dobbiamo anche installare le gemme correllate
- `$ bundle install`

- Ricordiamoci delle migration, in particolare la tabella utente avrà il seguente aspetto:

```ruby
class CreateUsers < ActiveRecord::Migration[8.0]
  def change
    create_table :users do |t|
      t.string :email_address, null: false
      t.string :password_digest, null: false

      t.timestamps
    end
    add_index :users, :email_address, unique: true
  end
end
```

```
bin/rails db:migrate
== 20250827145419 CreateUsers: migrating ======================================
-- create_table(:users)
   -> 0.0020s
-- add_index(:users, :email_address, {:unique=>true})
   -> 0.0005s
== 20250827145419 CreateUsers: migrated (0.0026s) =============================

== 20250827145420 CreateSessions: migrating ===================================
-- create_table(:sessions)
   -> 0.0028s
== 20250827145420 CreateSessions: migrated (0.0028s) ==========================
```

Mi sono dimenticato di aggiungere alcuni dettagli alla tabella user, come posso fare?

- Semplice, la risposta è generare una nuova migration che andrà a integrare nuove colonne nello schema di `user`
- Inseriamo `first_name`, `last_name`, `username`

```
└─$ bin/rails generate migration AddInfoToUsers first_name:string last_name:string username:string
      invoke  active_record
      create    db/migrate/20250827151434_add_info_to_users.rb
```

- Meglio non abusarne, ricordiamoci di utilizzare gli strumenti di gestione del codice come git ogni volta che iniziamo a lavorare su una nuova feature

```ruby
class AddInfoToUsers < ActiveRecord::Migration[8.0]
  def change
    add_column :users, :first_name, :string
    add_column :users, :last_name, :string
    add_column :users, :username, :string
  end
end
```

- Possiamo applicare la migration, aggiungere validation ai model e sistemare il controller.
-

$ bin/rails db:migrate

```
---
### Creazione di un Utente di Sviluppo

- Dato che non abbiamo ancora implementato una funzionalità di `sign_up`, possiamo aggiungere un utente o attraverso la console di Rails o sfruttando `db:seeds`
- Vediamo il secondo metodo.
- Al file `db/seeds.rb` va aggiunta la seguente linea:
  `User.create!(email_address: "pltfpp@unfe.it", password: "your_password", first_name: "Filippo", last_name: "Poltronieri", username: "pltfpp")`
- Poi va invocato il comando `bin/rails db:seed`
- Sarà così possibile autenticarsi all'interno dell'applicazione, ovviamente questo funziona solamente quando siamo in ambiente di sviluppo.
- *In alternativa* possiamo usare la console di Rails per eseguire la stessa operazione.

---
## Review

- La popolarità di *IMDB* dipende dalla presenza di recensioni, che vengono aggregate in una valutazione globale del film
- Inoltre, gli utenti hanno la possibilità di lasciare una recensione.
- Modelliamo quindi una nuova risorsa `review` che dia la possibilità di inserire una recensione specificando un `content` e un `rating`
```

- Vediamo il contenuto della migration:

```ruby
class CreateReviews < ActiveRecord::Migration[8.0]
  def change
    create_table :reviews do |t|
      t.references :movie, null: false, foreign_key: true
      t.references :user, null: false, foreign_key: true
      t.text :content
      t.integer :rating

      t.timestamps
    end
  end
end```

- Eseguiamo la migration e apportiamo le opportune modifiche ai model di `user` e `movie`
---

- Inoltre ci conviene inserire le rispettive validation per il voto, che vogliamo in un range da 1 a 10.

```ruby
# app/models/review.rb
class Review < ApplicationRecord
  broadcasts_to :movie
  belongs_to :movie
  belongs_to :user

  validates :rating, presence: true, inclusion: { in: 1..10 }
  validates :content, presence: true, length: { min: 10, maximum: 750 }
end
```

- Cosa dobbiamo ricordarci? In questo caso vogliamo specificare che la risorsa
```

Reviews: Views

- Per quanto riguarda la creazione di `Views`, in questo caso possiamo procedere con la creazione di sole partial.
- Vogliamo caricare le revisioni direttamente nella view associata a `movie`, allo stesso modo che abbiamo fatto per casting.
- In questo caso abbiamo bisogno di una view `_review.html.erb` per la visualizzazione della singola review, e `_reviews.html.erb` per la visualizzazione del blocco di review.
- E di un view `form.html.erb` per l'inserimento della review stessa.
- In seguito possiamo pensare di modificare la view `_moview.html.erb` per visualizzare un voto medio.

```
# app/views/reviews/_form.html.erb
<%= form_with model: [ movie, Review.new ] do |form| %>
  Your review:<br>
  <%= form.text_area :content, size: "20x5" %>
  <br>
  <!-- specify rating here-->
  <%= form.label :rating, style: "display: inline;" %>
  <%= form.number_field :rating,
                        in: 1..10,
                        step: 1,
                        style: "width: 50px; display: inline;" %>
  <br>
  <%= form.submit "Comment" %>
<% end %>
```

- Vediamo come il model specificato nel form comprenda due oggetti `[ movie, Review.new]`, come mal?
- Proviamo ad aprire la console Rails e provare qualche inserimento manuale.

```
# app/views/reviews/_reviews.html.erb
<div id="reviews">
  <%= render movie.reviews if movie.reviews.any? %>
</div>
<%= render "reviews/form", movie: movie %>
```

- Invece, la visualizzazione della singola review andrà a stampare il contenuto, il voto e l'username associato

```
# app/views/reviews/_reviews.html.erb
<div id="<%= dom_id(review) %>">
  <%= review.content %>
  –
  <%= review.rating %>
  / 10 –
  <%= Current.user.username %>
  –
  <%= time_tag review.updated_at, "data-local": "time-ago" %>
  –
  <%= button_to "Delete review",
  [review.movie, review],
  method: :delete,
  data: {
    turbo_confirm: "Delete this review?",
  } %>
</div>
```

Esercizio: Review Views

- Dove andranno visualizzate queste review?
- Modificare il codice per visualizzare e consentire l'inserimento di review.
- Modificare i rispettivi controller per consentire *la creazione di review* e l'eventuale *modifica*