



NOTTEC

Proyecto de página web desarrollada con Ruby on Rails

Descripción breve

Especificación de los diferentes puntos de la práctica 3 de DAW1, la cual consiste en un proyecto de Ruby on Rails, creando un portal de noticias sobre temas relacionados con la tecnología.

JUAN RAMÓN BETANCOR OLIVARES | NÉSTOR DOMÍNGUEZ BOLAÑOS – GRUPO 7

ULPGC – EII – DAW1 – TRABAJO DE CURSO Nº 3.

Título y acrónimo del proyecto.

notTec, portal de noticias sobre tecnología.

Resumen de la implementación.

Nos encontramos frente a un proyecto que consiste en implementar un portal de noticias sobre tecnología, informática, series, es decir, en general del mundo geek. Para ello debemos usar Ruby on Rails, así como la utilización de otros lenguajes de programación web como HTML5, CSS3, Bootstrap, JavaScript, etc.

Dicho portal consta de un índice donde cualquier tipo de usuario puede ver las últimas noticias publicadas en la web. Si hace clic a una de esas noticias puede acceder al contenido de esta, que contendrá el cuerpo de la noticia, categorías pertenecientes, fecha de publicación, usuario que la ha escrito, etc. Por otra parte, también podrá acceder a la sección de comentarios y ver los comentarios de los demás usuarios, así como acceder a sus perfiles, y, si el usuario está registrado en la web e iniciado sesión en la misma, también podrá realizar comentarios. Si el usuario prefiere buscar las noticias de una categoría específica, puede hacerlo a través de la barra de navegación, que mostrará todas las categorías creadas en la página. Estas funciones son las básicas que cualquier rol puede hacer en la página.

En la página hay 4 roles. Un usuario no logeado en la página, usuario registrado y logeado, editor y administrador. El usuario no logeado podrá hacer únicamente lo nombrado anteriormente. Un usuario logeado, además de ver las noticias podrá realizar comentarios en las noticias, así como suscribirse a la newsletter.

El editor además podrá crear noticias nuevas, editar las noticias publicadas y no publicadas y crear nuevas categorías en la página, editarlas y eliminarlas. Y, por último, el administrador podrá eliminar las noticias, ya sea publicadas o no publicadas. Además, podrá publicar las noticias de los editores, así como editarlas y crearlas. Respecto a las categorías, también podrá crearlas, eliminarlas y editarlas. También tendrá a su disposición un panel de control de usuarios, el cual podrá ver los usuarios que actualmente están en la página, su nivel de permiso y un botón de eliminarlo en caso de que se requiera.

Cada usuario tendrá un perfil, en el que se puede ver cierta información y la actividad en la página tales como los comentarios que han hecho como los artículos publicados en caso de que sean editores o administradores.

En todas las páginas aparecerá una barra de navegación en la cual el usuario en todo momento podrá acceder a las noticias, su perfil, categorías, contacto, información de la empresa, etc. Así como un footer con las redes sociales del portal.

Descripción de los elementos obligatorios.

ActiveRecord

- Modelos de entidades del dominio:

En el portal se encuentran varios modelos. Estos son:

- Modelo User, con las especificaciones de los usuarios, atributos, etc.
- Modelo Article: el modelo que representará a las noticias.
- Modelo Comment: modelo que representa a los comentarios.
- Modelo Welcome: modelo que contendrá las vistas restantes, tales como el perfil del usuario, panel de control, perfil, etc.
- Modelo Suscriber: que representa a los usuarios suscritos al newsletter del portal.

- Modelo de usuario:

Como hemos nombrado antes, se dispone de un modelo de usuario. La gestión de los usuarios y de las sesiones en gran medida fueron generadas con la gema “devise”.

- Asociaciones entre modelos:

Se dispone de varias asociaciones entre los modelos anteriormente mencionados. Son los siguientes:

- Un usuario tiene una relación de 1 a muchos con comentarios y artículos, ya que un usuario puede haber creado muchos comentarios y muchos artículos. Pero cada comentario y cada artículo tan sólo está creado por un usuario. Cuando se hace un delete de un usuario, los artículos permanecen, así como los comentarios, ya que se ha puesto como clausura de borrado “nullify”.
- Una categoría puede tener muchos artículos, así como un artículo varias categorías por lo que es una relación muchos a muchos. Si una categoría fuese borrada, los artículos no se borrarían ya que está con “nullify”.
- Un artículo puede tener muchos comentarios, pero cada comentario está asociado a un artículo por lo que tenemos una relación 1 a muchos. Si un artículo fuese borrado, entonces los comentarios si serían borrados ya que se borran en cascada.

- Validaciones:

Se ha implementado varias validaciones tales como en el formulario de un artículo, ningún campo puede estar vacío, el título de una noticia debe ser único, el tamaño del contenido de la noticia no puede ser menor de 20 caracteres, el cuerpo de la noticia no puede estar vacío, las categorías deben de tener un nombre, etc. En los formularios de inicio de sesión y registro también se especifica varias validaciones.

- Scopes:

Se han utilizado principalmente dos scopes para el filtrado de las noticias. Las noticias se muestran en orden decreciente de fecha de creación de tal manera que las noticias más recientes aparecen arriba. Por otra parte, los artículos tienen dos estados, "modo borrador" y modo publicado. Pues se ha creado un scope para que sólo se muestren a los usuarios las noticias publicadas. Dichos estados se han implementado con una gema llamada AASM, que consiste en una máquina de estados, el cual se le puede asociar a un objeto varios estados.

ActionPack

- Templates

Hay varios templates como los de usuarios, artículos, comentarios, etc.

- Rutas con nombre

Se ha creado varias rutas con nombre, por ejemplo: dashboard hace referencia a welcome/dashboard, about que es la vista con información de la web hace referencia a welcome/about, al igual que contact welcome/contact, entre otras.

- Controladores para los modelos definidos

Hay un controlador para cada modelo mencionado anteriormente (article, suscriber, comment, user, etc.).

- Controladores de sesiones y usuarios

El controlador de las sesiones y usuarios se autogeneran con la gema devise que es la encargada de gestionar los usuarios.

- Formularios con tipos básicos

Hay varios formularios de tipos básicos como son text_field, tex_area en los campos de cuerpo y titulo de las noticias, así como los del cuerpo de los comentarios e inicio y registro de sesión.

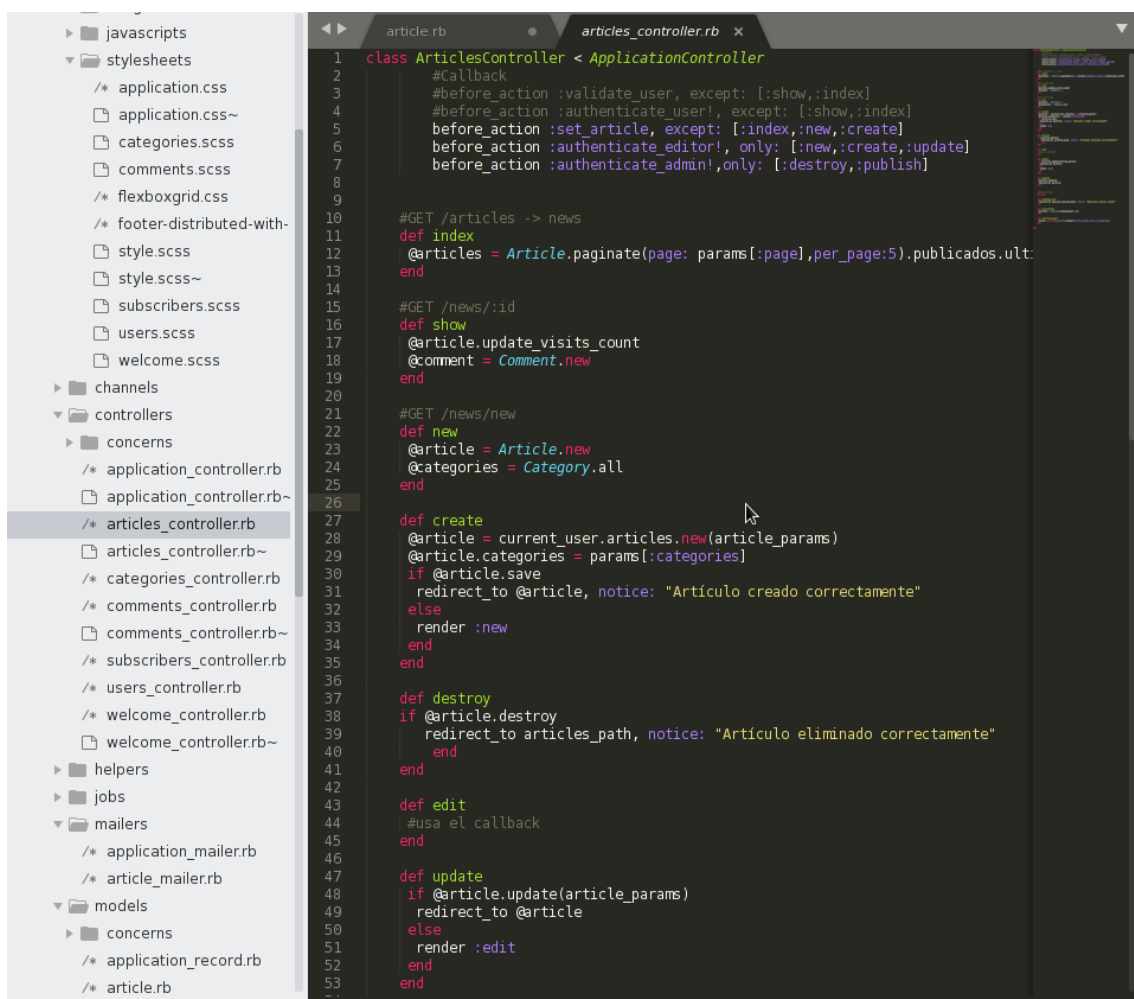
- JQuery UI

Se ha implementado con JQuery UI un datepicker para seleccionar la fecha de creación de las noticias.

Elementos opcionales utilizados.

- Callbacks en modelos

Se ha usado varios por ejemplo en el modelo article, antes de guardar “before_save”, se pone el número de visitas a 0. También se hace un before_create que consigue que cuando se cree el artículo, también se cree dicho artículo en las categorías propuestas. Además se ha creado varios callbacks en los controladores tales como “before_action :authenticate_admin!”, el cual establece que los administradores pueden hacer cierto tipo de cosas que otros roles no pueden hacer.



```
1 class ArticlesController < ApplicationController
2   #Callback
3   #before_action :validate_user, except: [:show,:index]
4   #before_action :authenticate_user!, except: [:show,:index]
5   before_action :set_article, except: [:index,:new,:create]
6   before_action :authenticate_editor!, only: [:new,:create,:update]
7   before_action :authenticate_admin!, only: [:destroy,:publish]
8
9
10  #GET /articles -> news
11  def index
12    @articles = Article.paginate(page: params[:page], per_page: 5).publicados.ult
13  end
14
15  #GET /news/:id
16  def show
17    @article.update_visits_count
18    @comment = Comment.new
19  end
20
21  #GET /news/new
22  def new
23    @article = Article.new
24    @categories = Category.all
25  end
26
27  def create
28    @article = current_user.articles.new(article_params)
29    @article.categories = params[:categories]
30    if @article.save
31      redirect_to @article, notice: "Artículo creado correctamente"
32    else
33      render :new
34    end
35  end
36
37  def destroy
38    if @article.destroy
39      redirect_to articles_path, notice: "Artículo eliminado correctamente"
40    end
41  end
42
43  def edit
44    #usa el callback
45  end
46
47  def update
48    if @article.update(article_params)
49      redirect_to @article
50    else
51      render :edit
52    end
53  end
54 end
```

- Rutas complejas

Hay varias rutas complejas que se han usado, por ejemplo, para borrar un comentario se accede a `"/articles/:article_id/comments/:id"` o a `"/articles/:id/publish"`.

Si ejecutamos `"rake routes"`, podemos ver las rutas del proyecto:

```

new_user_session GET    /users/sign_in(.:format)
user_session POST   /users/sign_in(.:format)
destroy_user_session DELETE /users/sign_out(.:format)
new_user_password GET    /users/password/new(.:format)
edit_user_password GET    /users/password/edit(.:format)
user_password PATCH  /users/password(.:format)
PUT    /users/password(.:format)
POST   /users/password(.:format)
cancel_user_registration GET    /users/cancel(.:format)
new_user_registration GET    /users/sign_up(.:format)
edit_user_registration GET    /users/edit(.:format)
user_registration PATCH  /users(.:format)
PUT    /users(.:format)
DELETE /users(.:format)
POST   /users(.:format)
users GET    /users(.:format)
POST   /users(.:format)
new_user GET    /users/new(.:format)
edit_user GET    /users/:id/edit(.:format)
user GET     /users/:id(.:format)
PATCH  /users/:id(.:format)
PUT     /users/:id(.:format)
DELETE  /users/:id(.:format)
subscribers GET    /subscribers(.:format)
POST    /subscribers(.:format)
new_subscriber GET    /subscribers/new(.:format)
edit_subscriber GET    /subscribers/:id/edit(.:format)
subscriber GET     /subscribers/:id(.:format)
PATCH  /subscribers/:id(.:format)
PUT     /subscribers/:id(.:format)
DELETE  /subscribers/:id(.:format)
welcome_index GET    /welcome/index(.:format)
article_comments POST   /articles/:article_id/comments(.:format)
article_comment PATCH  /articles/:article_id/comments/:id(.:format)
PUT     /articles/:article_id/comments/:id(.:format)
DELETE  /articles/:article_id/comments/:id(.:format)
articles GET     /articles(.:format)
POST    /articles(.:format)
new_article GET    /articles/new(.:format)
edit_article GET    /articles/:id/edit(.:format)
article GET     /articles/:id(.:format)
PATCH  /articles/:id(.:format)
PUT     /articles/:id(.:format)
DELETE  /articles/:id(.:format)
root GET    /dashboard(.:format)
contact GET    /contact(.:format)
about GET    /about(.:format)
PUT     /articles/:id/publish(.:format)
infouser GET    /infouser(.:format)
devise/sessions/new devise/sessions/new
devise/sessions/create devise/sessions/create
devise/sessions/destroy devise/sessions/destroy
devise/passwords/new devise/passwords/new
devise/passwords/edit devise/passwords/edit
devise/passwords/update devise/passwords/update
devise/passwords/create devise/passwords/create
devise/registrations/cancel devise/registrations/cancel
devise/registrations/new devise/registrations/new
devise/registrations/edit devise/registrations/edit
devise/registrations/update devise/registrations/update
devise/registrations/destroy devise/registrations/destroy
devise/registrations/create devise/registrations/create
users#index
users#create
users#new
users#edit
users#show
users#update
users#destroy
subscribers#index
subscribers#create
subscribers#new
subscribers#edit
subscribers#show
subscribers#update
subscribers#destroy
welcome#index
comment#create
comment#update
comment#destroy
articles#index
articles#create
articles#new
articles#edit
articles#show
articles#update
articles#destroy
welcome#dashboard
welcome#contact
welcome#about
welcome#publish
welcome#index
  
```

Fichero routes:

```

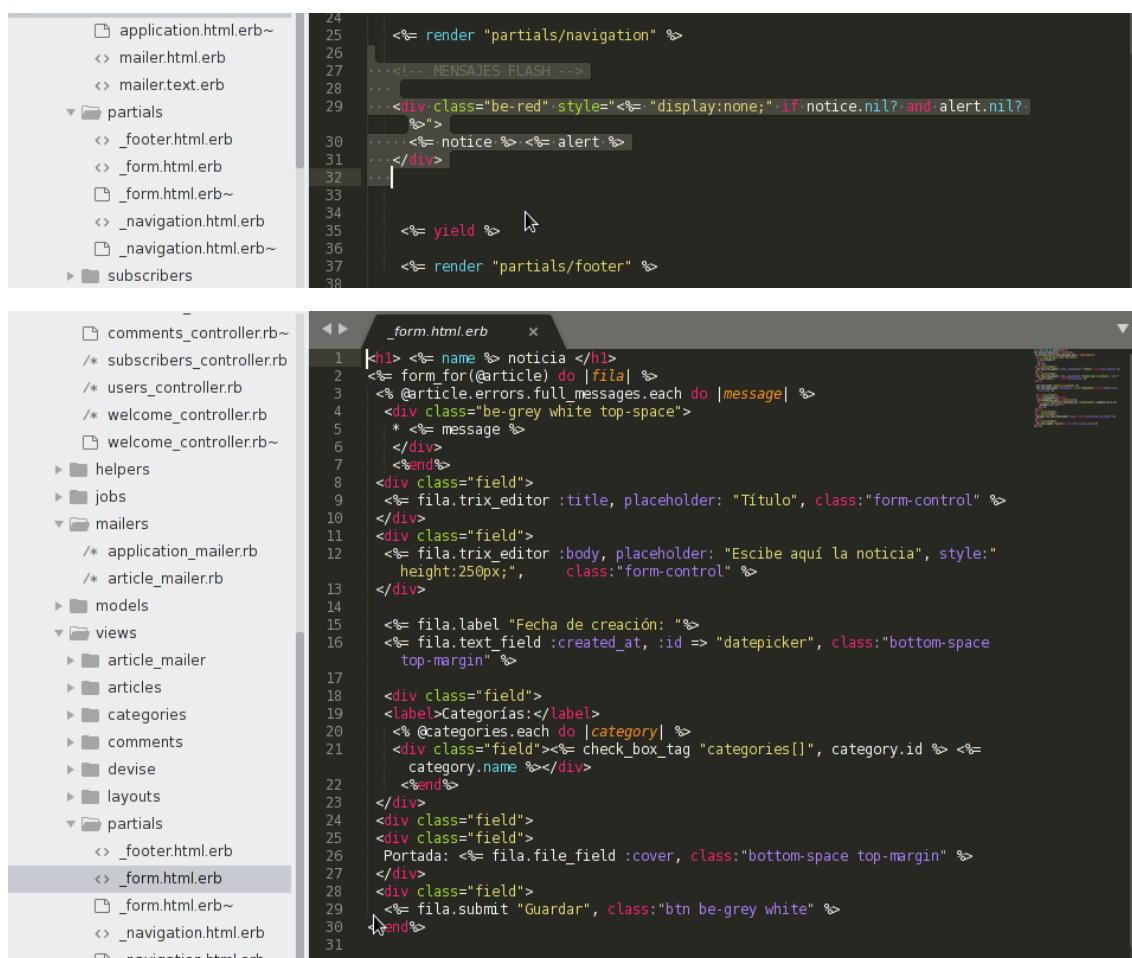
1  Rails.application.routes.draw do
2
3    get 'subscribers/index'
4
5    get 'users/show'
6
7    resources :categories
8    devise_for :users
9    resources :users
10   resources :subscribers
11   get 'welcome/index'
12   #get "paginicio", to: "welcome#index" -> etiquetas
13
14   # NESTED RESOURCES, recursos anidados
15   resources :articles do
16     resources :comments, only: [:create, :destroy, :update]
17   end
18
19   # Ruta raíz
20   root 'articles#index'
21
22   get "/dashboard", to: "welcome#dashboard"
23   get "/contact", to: "welcome#contact"
24   get "/about", to: "welcome#about"
25
26   put "/articles/:id/publish", to: "articles#publish"
27
28   get "/infouser", to: "welcome#index"
29
30   # For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html
31 end
32
  
```

- Recursos anidados

Se encuentra que comentarios está anidado con artículos. Podemos verlo en el bloque resources :articles do ... de la imagen anterior.

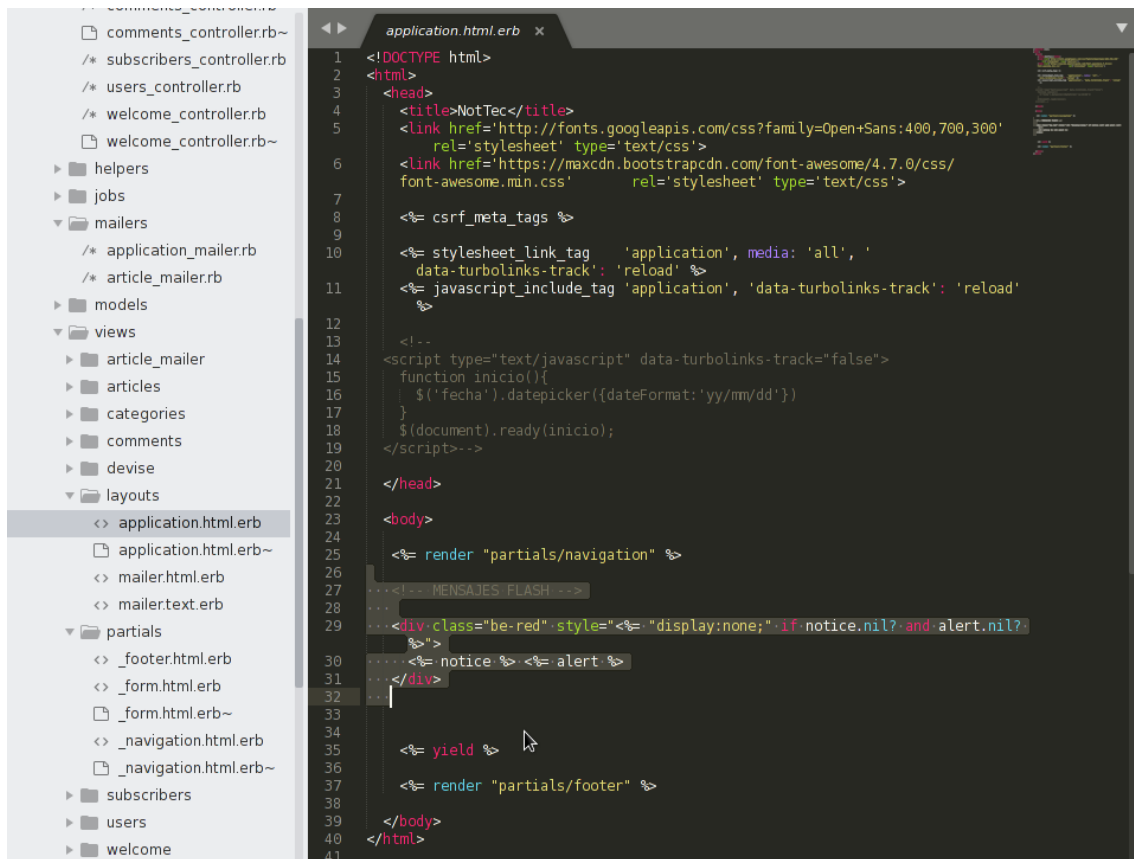
- Partials

Se ha creado varios partials, como el formulario de edición y creación de noticias, la navegación y el footer, entre otros. Se muestran algunos ejemplos, por ejemplo, en el layout se ha utilizado dos partials, uno para la navegación y otro para el footer, así como un partial para el formulario de creación y edición de noticias, ya que usan el mismo código.

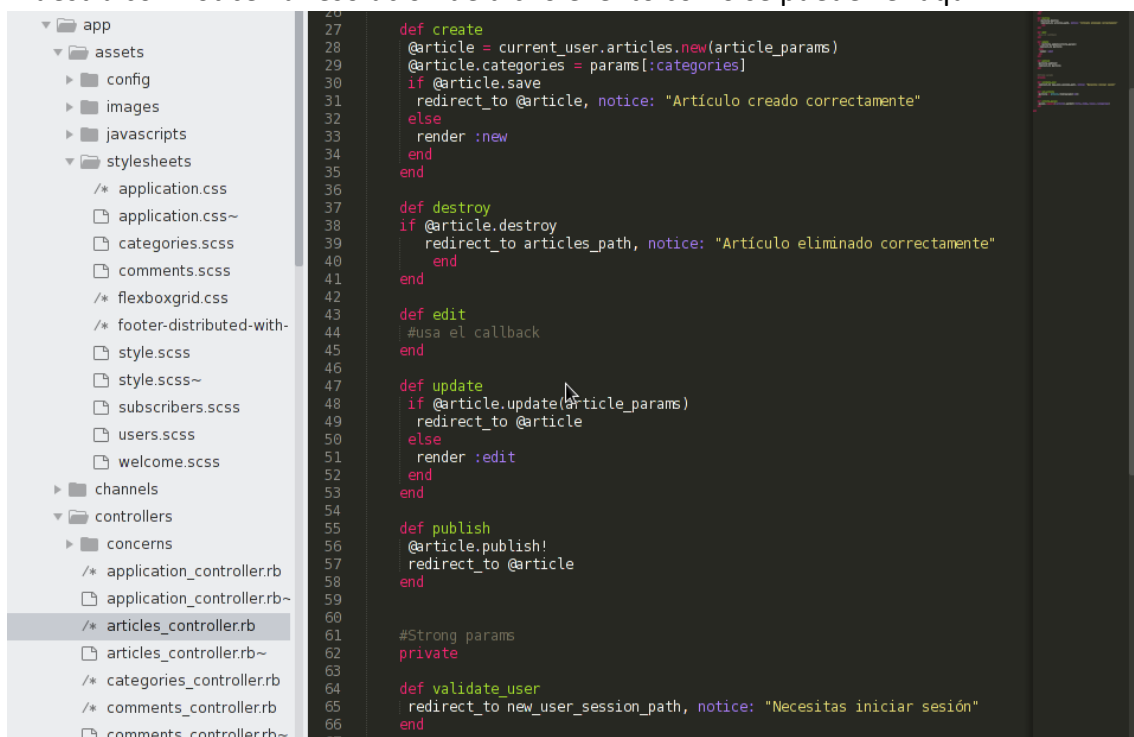


- Utilización del objeto flash:

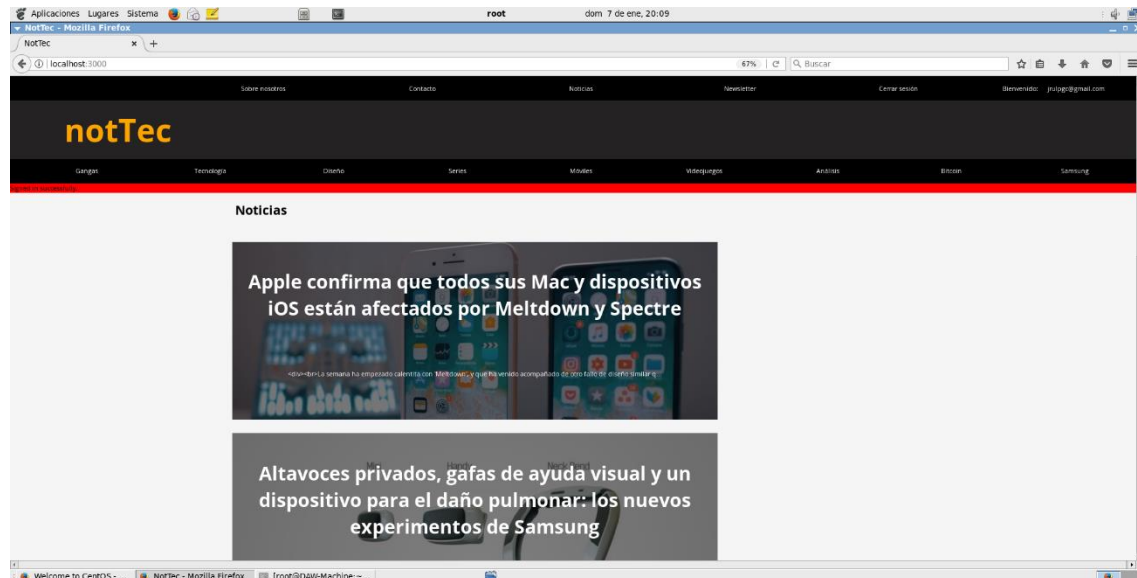
Para mostrar al usuario mensajes e informarle de posibles errores o que todo va bien se ha implementado en el layout la sección de mensajes flash (notice y alert) como podemos ver en la siguiente imagen:



Cuando se produzca algún evento en el controlador de artículos por ejemplo, se muestra con notice: la resolución de dicho evento como se puede ver aquí:

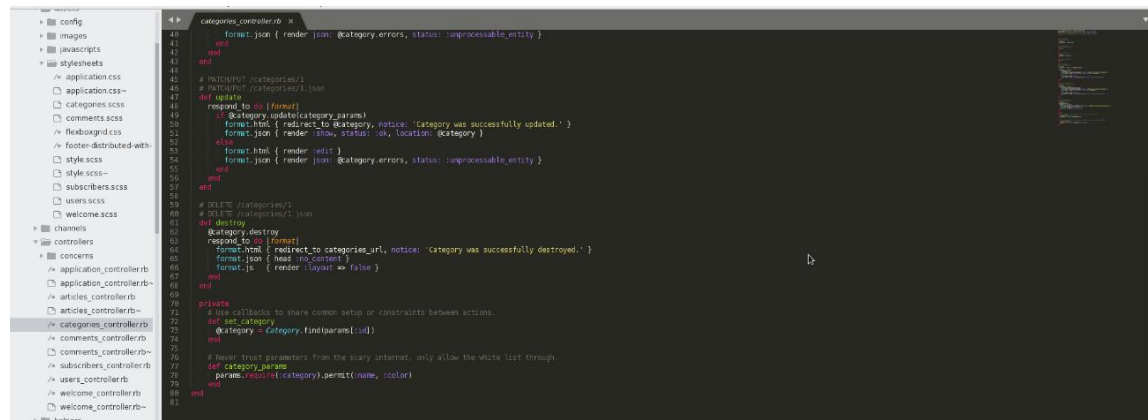


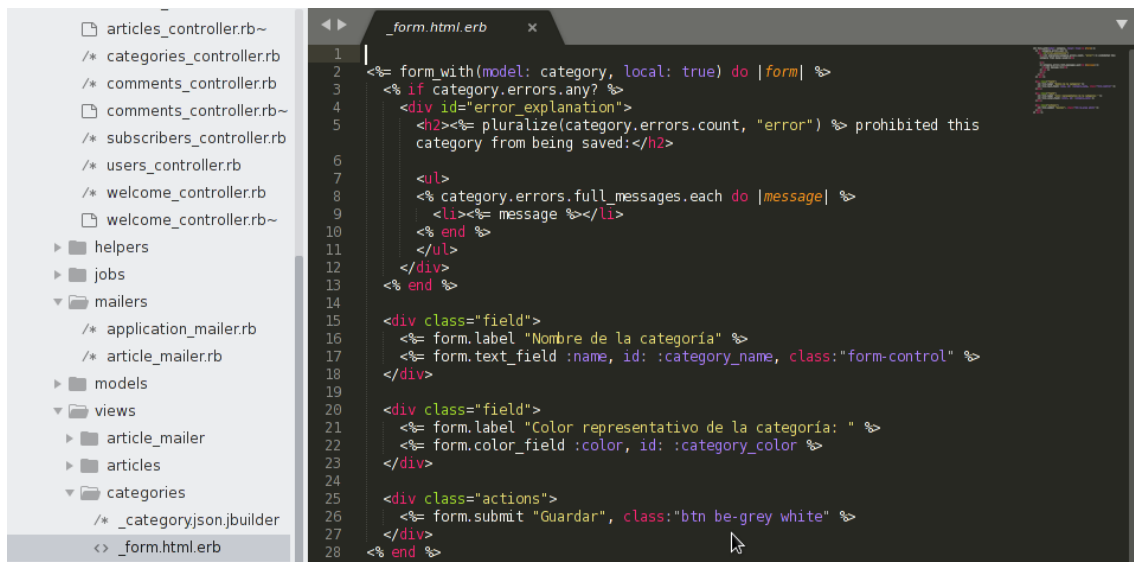
Quedaría de esta manera:



- Ajax:

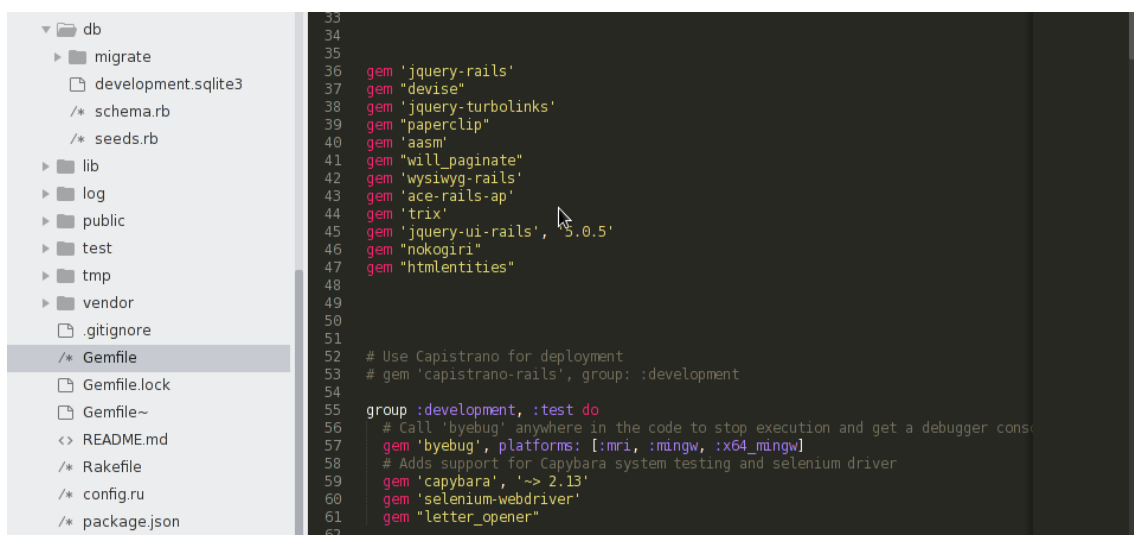
Varios formularios están preparados para ejecutarlo usando Ajax (“remote= true”), por ejemplo, cuando borramos un artículo se borra haciendo una petición Ajax como podemos ver aquí:





- Otras gemas que aumenten la funcionalidad.

Se ha utilizado “devise” para la gestión de usuarios y sesiones. Paperclip para la subida de imágenes junto con el software Image Magic instalado en la MV. La gema AASM para simular los estados de las noticias (borrador y publicado). WillPaginate para mostrar por páginas las noticias. Trix para ofrecer al usuario una interfaz WYSIWYG a la hora de escribir los artículos. JQuery UI y JQuery para el datepicker. Además, se puso en su momento LetterOpener para visualizar el correo enviado, pero no llegó a funcionar.



- Otras funcionalidades del lenguaje
 - Se ha implementado un HELPER llamado `article_cover` el cual se le pasa la URL de la imagen que se desea, así como de unas opciones para crear las caratulas de las noticias.

```

1 module ApplicationHelper
2   def article_cover url, options = {}
3
4     html_class = options[:class]
5     html_style = "background:url({url});\n
6                 width:100%;height:400px;background-size: cover;"
7     html = "<header style='{html_style}' class='{html_class}'>\n
8           </header>"
9     html.html_safe
10  end
11 end
12

```

- Se ha intentado implementar el Mailer, pero finalmente por motivos desconocido no llegó a funcionar.
- Strong params: se ha usado algunos como:

```

60 #Strong params
61 private
62
63 def validate_user
64   redirect_to new_user_session_path, notice: "Necesitas iniciar sesión"
65 end
66
67 def set_article
68   @article = Article.find(params[:id])
69 end
70
71 def article_params
72   params.require(:article).permit(:title,:body,:cover,:categories)
73 end
74
75 end
76
77

```

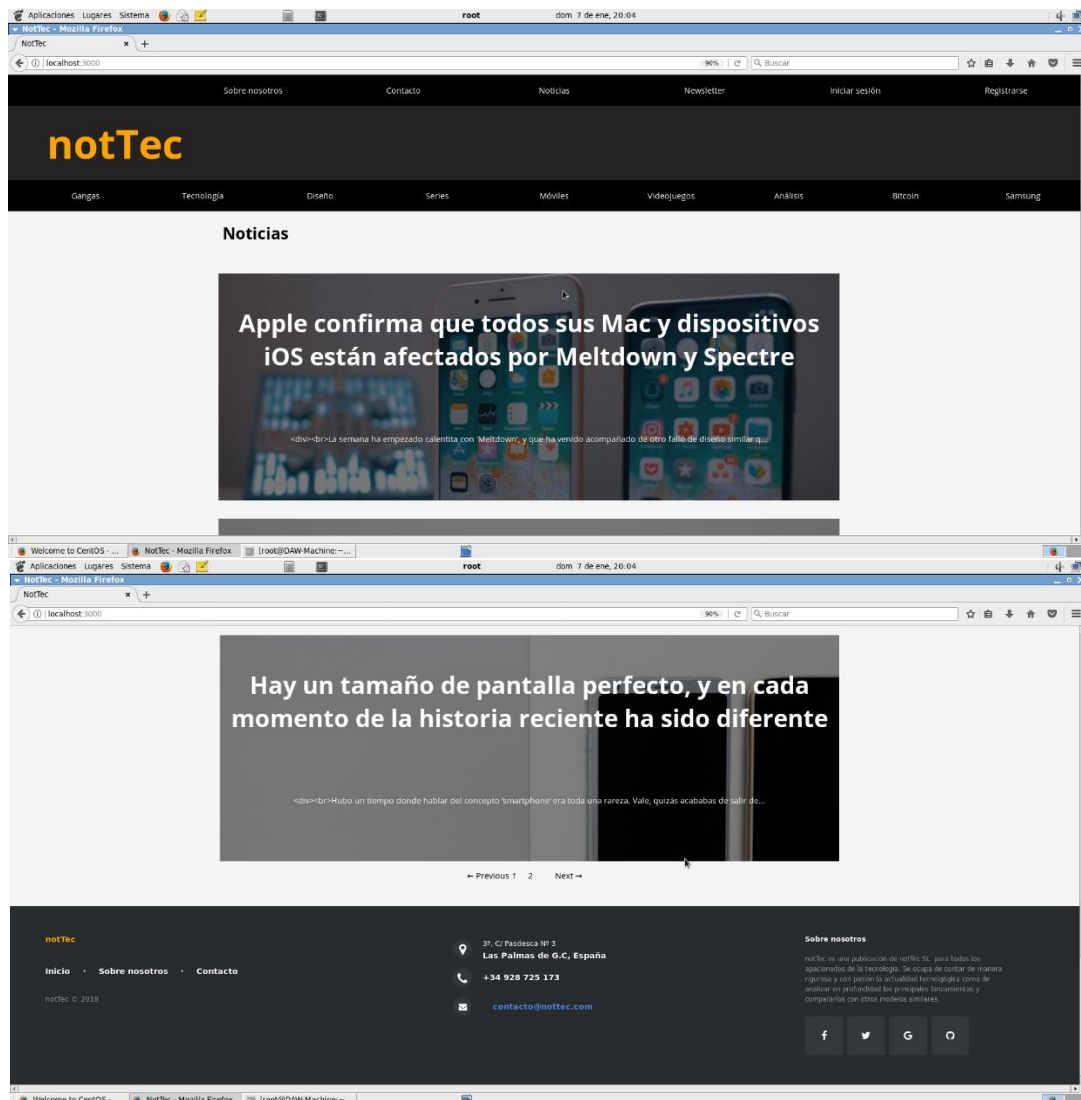
- Utilización de git + bitbucket

Proyecto Ruby on Rails: <https://github.com/JuanRaBetancor/notTec>

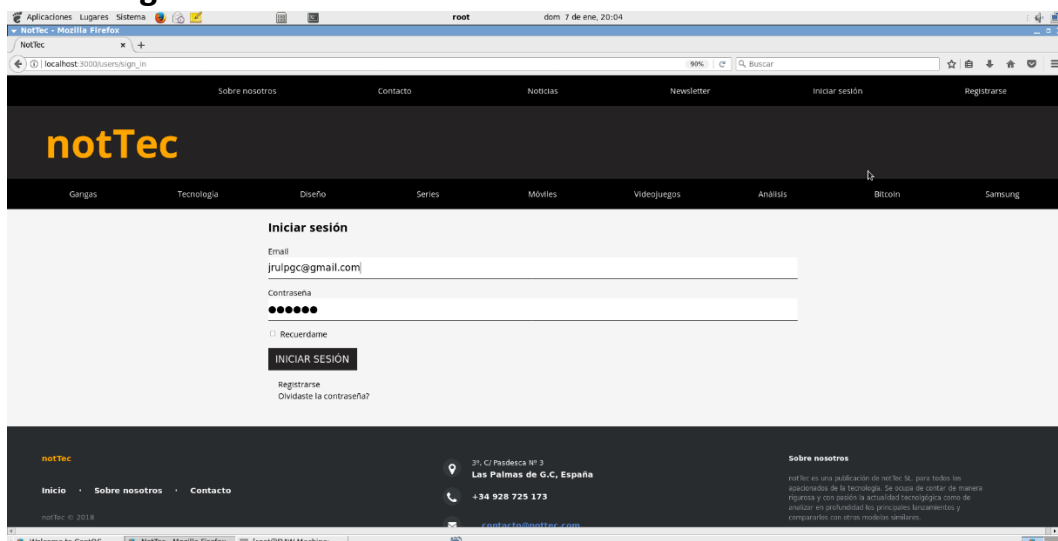
Práctica 2 y práctica 2 corregida: <https://github.com/JuanRaBetancor/DAW1>

Capturas de pantalla de las páginas implementadas.

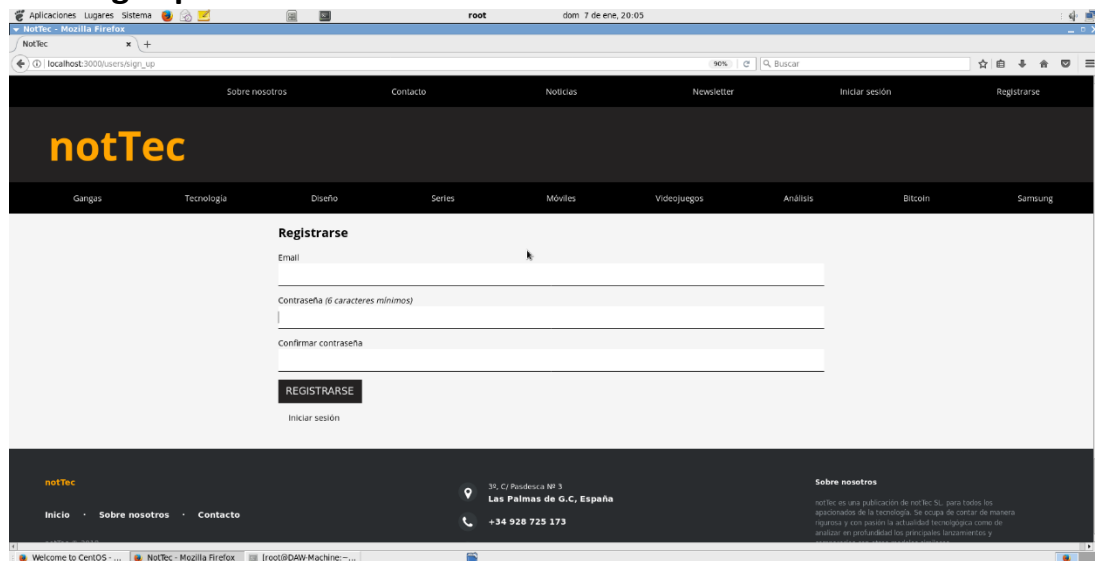
1. Índice



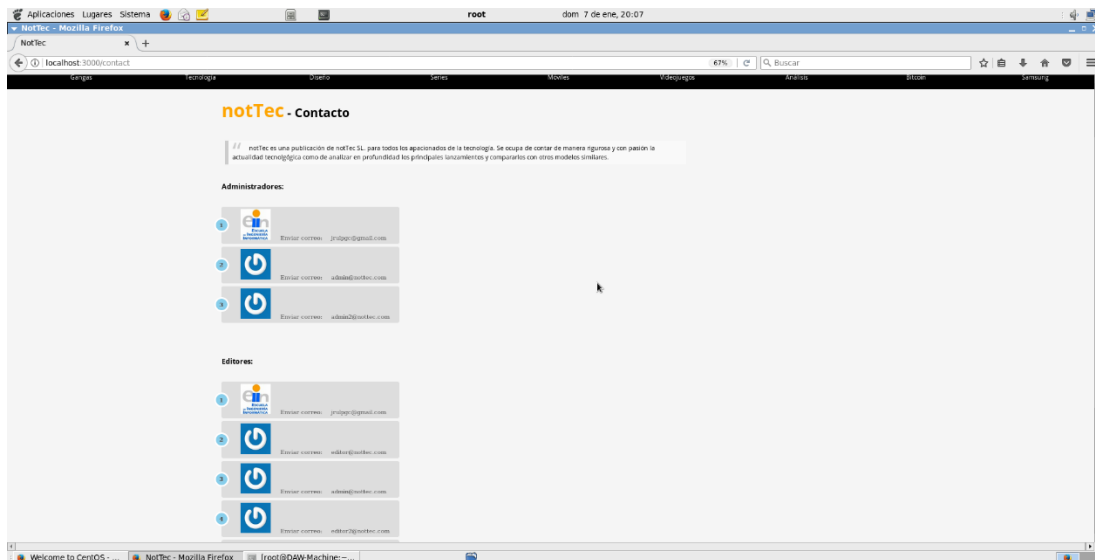
2. Login



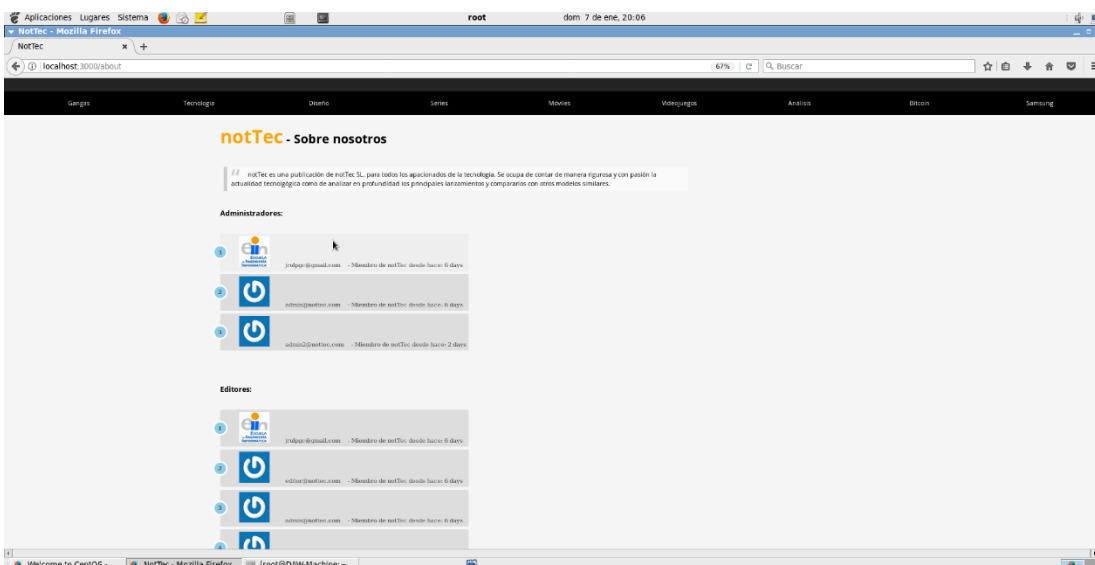
3. SignUp



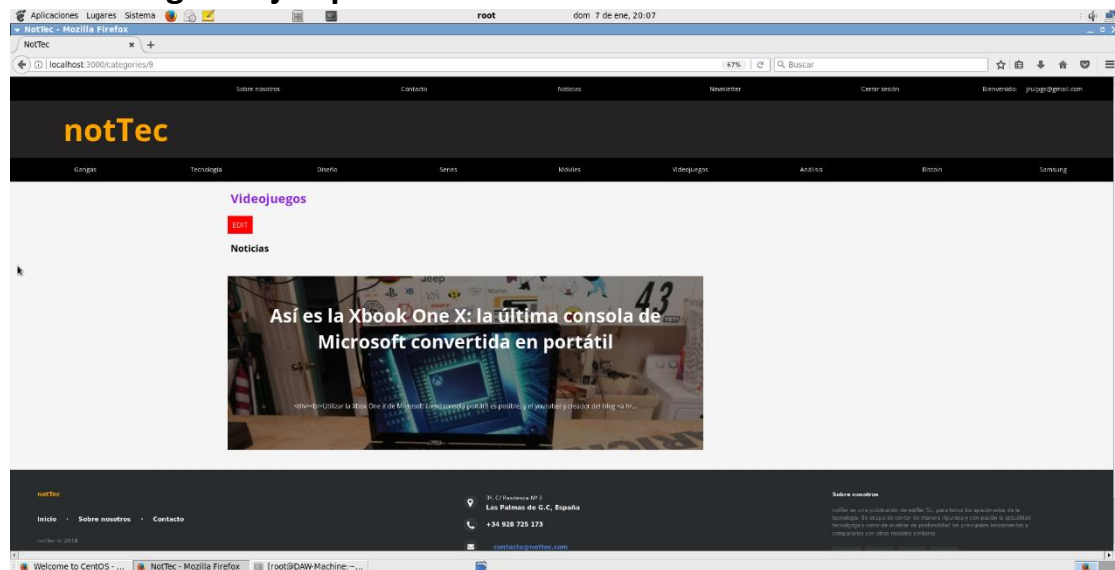
4. Contact



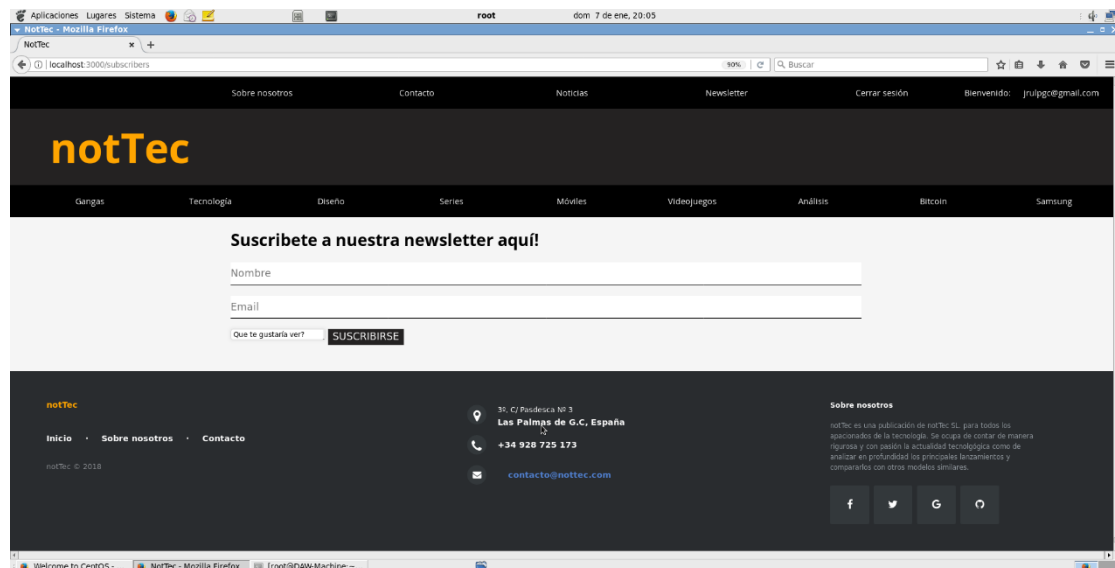
5. About



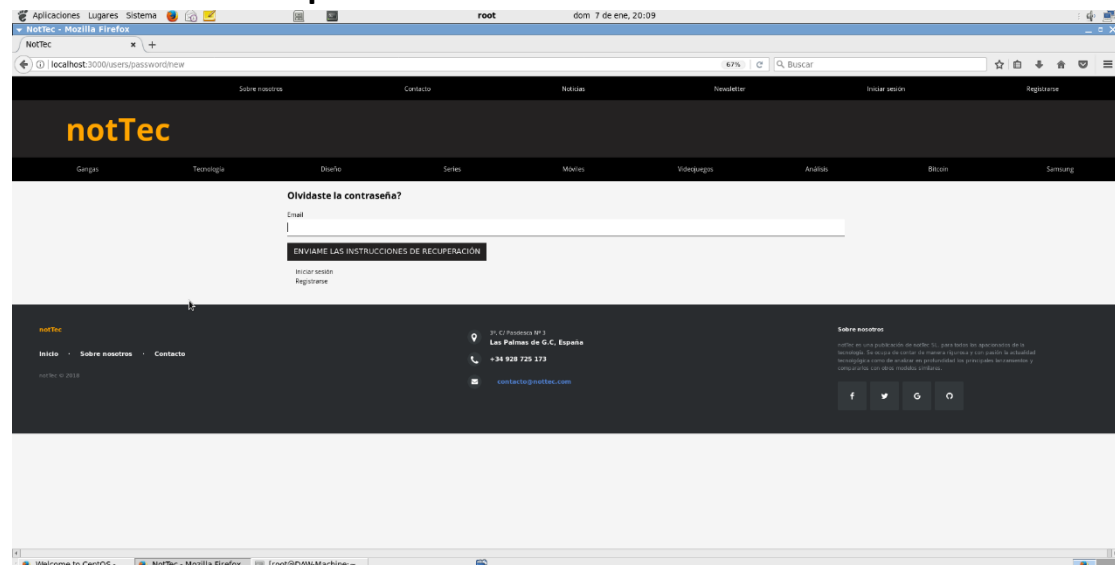
6. Categoría ejemplo.



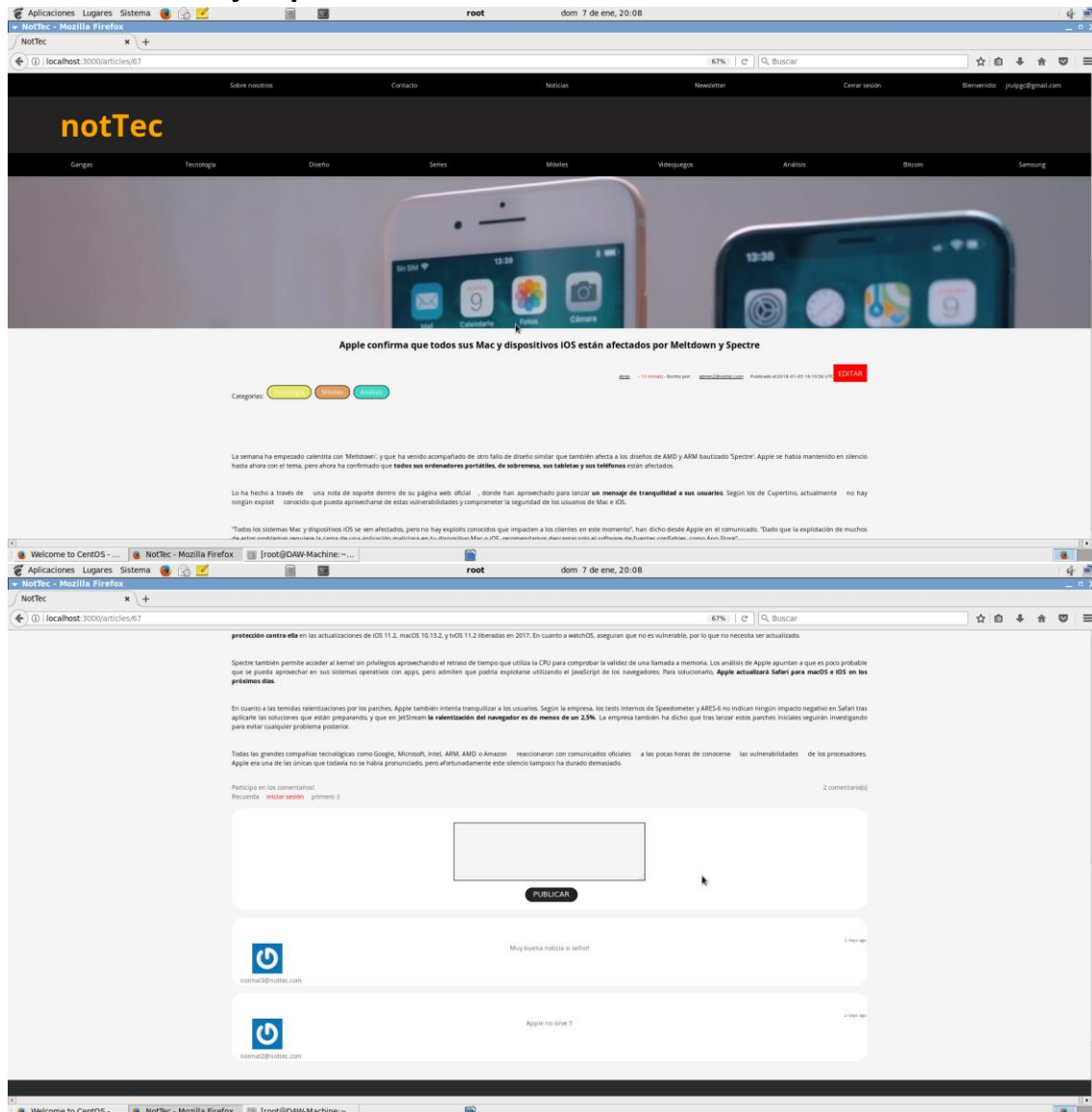
7. Newsletter



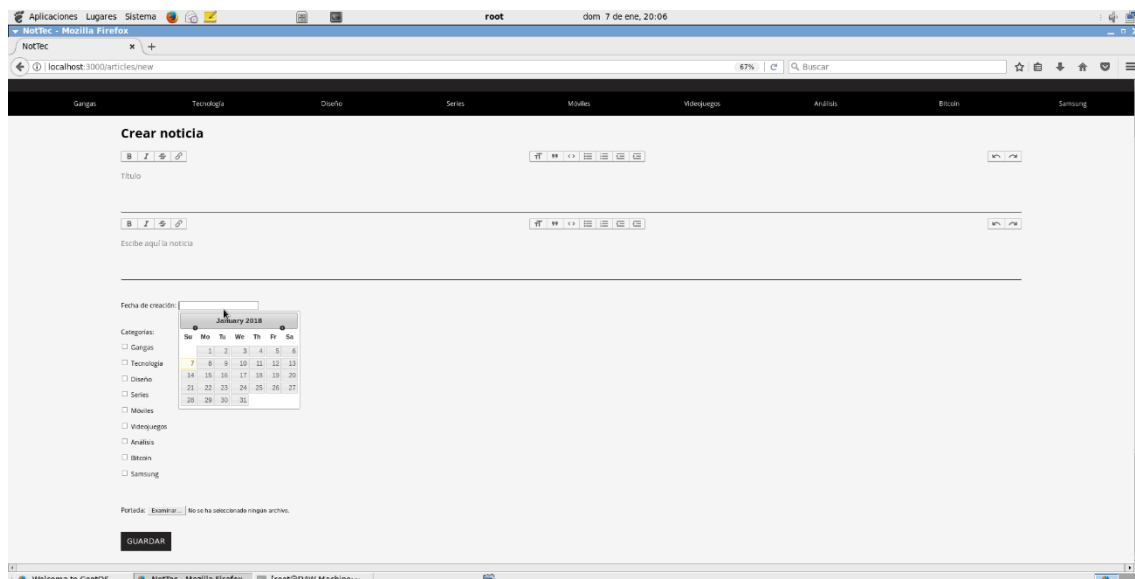
8. Recordatorio password



9. Noticia ejemplo



10. Nueva noticia



11. Nueva categoría

The screenshot shows the 'Nueva Categoría' (New Category) form in the notTec application. The form is located in the center of the page and contains the following fields and elements:

- Nombre de la categoría:** A text input field for the category name.
- Color representativo de la categoría:** A color selection field with a small color picker icon.
- GUARDAR:** A green button to save the new category.
- ATRÁS:** A red button to go back.

The application's header and footer are visible, showing the notTec logo, navigation links, and contact information.

12. Editar Usuario

The screenshot shows the 'Editar Usuario' (Edit User) form in the notTec application. The form is located in the center of the page and contains the following fields and elements:

- Email:** A text input field with the value 'julppg@gmail.com'.
- Contraseña:** A password input field with a strength indicator (6 dots).
- Confirmación de la contraseña:** A text input field for confirming the password.
- Contraseña actual:** A text input field for the current password.
- GUARDAR:** A green button to save the user profile.
- Cancelar la cuenta:** A red button to cancel the account.
- Cancelar cuenta:** A red button to cancel the account.

The application's header and footer are visible, showing the notTec logo, navigation links, and contact information.

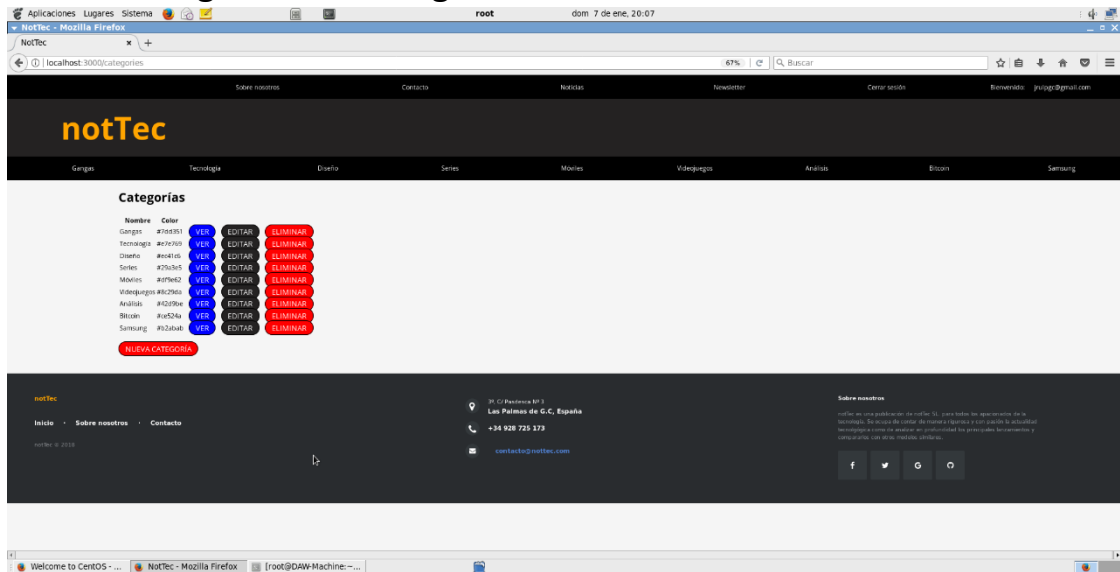
13. Perfil Usuario

The screenshot shows the 'Perfil Usuario' (User Profile) page in the notTec application. The page is divided into several sections:

- Panel de control de categorías:** A red button at the top.
- Panel de control de artículos:** A red button at the top.
- Usuarios registrados:** A red button at the top.
- Perfil de usuario:** A section showing the user's profile information, including the email 'julppg@gmail.com', a profile picture, and a 'Modificar/Borrar cuenta' link.
- Actividad:** A section showing the user's activity, including the number of articles (3) and comments (2).
- Noticias hechas:** A section showing the user's news articles, including a featured article titled 'El problemón de los procesadores de Intel: a quién afecta, a quién no y cómo solucionarlo'.

The application's header and footer are visible, showing the notTec logo, navigation links, and contact information.

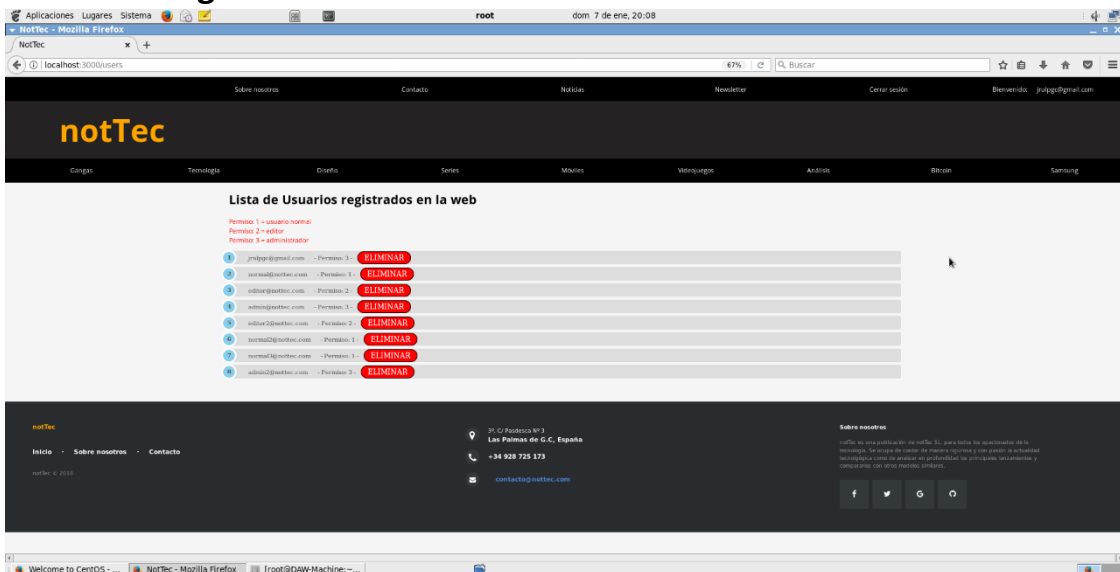
14. Panel gestión de categorías



15. Panel gestión de noticias



16. Panel gestión de usuarios



Estas son las principales páginas implementadas en el trabajo, aunque hay alguna más, pero con menor contenido. En todas se ha utilizado el grid de FlexBox (<http://flexboxgrid.com/>).

Referencias utilizadas.

La referencia principal utilizada a la hora de hacer el proyecto es la siguiente lista de reproducción de tutoriales sobre Ruby on Rails:

- https://www.youtube.com/watch?v=LMD3P97gXa0&list=PLpOqH6AE0tNiQ-ofrDlbAUsc1r67r_AWv

Gema trix para los campos de texto:

- [Let's Code - WYSIWYG on Ruby on Rails 5 using Basecamp's Trix - YouTube](#)

Libro de Ruby on Rails: http://librosweb.es/libro/introduccion_rails/

Ejemplo de Ajax con Rails: [Simple Ajax example with Ruby on Rails - YouTube](#)

Suscriber Model: <http://www.mattmorgante.com/technology/newsletter>

Datepicker: <http://www.binarywebpark.com/how-to-add-jquery-ui-datepicker-to-your-rails-4-application-with-formtastic-and-cocoon/>

Diferentes dudas: <https://stackoverflow.com/questions/tagged/ruby-on-rails>

Truncate para el preview: <https://stackoverflow.com/questions/4320160/is-there-an-html-safe-truncate-method-in-rails>

Proyecto Rails completo.

Proyecto Ruby on Rails: <https://github.com/JuanRaBetancor/notTec>

Modificaciones de los trabajos anteriores.

Práctica 2, práctica 2 corregida y práctica 1 corregida:

<https://github.com/JuanRaBetancor/DAW1>