Pontificia Universidad Católica de Chile
Escuela de Ingeniería
IIC3582 – Tecnologías y aplicaciones del WWW

Documentación página web EventMaker

Nombres: Marco Antonio Arias Figueroa
Carlos Molina Avendaño
Profesor: Jaime Navon
Fecha: 3 / Julio / 2012

1.- Introducción

EventMaker es una página dedicada a que personas de todo el mundo puedan organizar sus eventos, ya sean cumpleaños, aniversarios, matrimonios, fiestas, etc. Pueden ser fiestas públicas, donde todo el mundo puede ir, o fiestas privadas, en donde se necesita una invitación de la persona que creó el evento para tener acceso a él.
La aplicación permite loguearse solo usando el email, a la vez de inscribirse en unos pocos pasos. Luego aparece un menú con todos los eventos del usuario ordenados por fecha, además de permitir crear nuevos eventos, eliminarlos, invitar gente, crear grupos de amigos, y mucho más.

2.- Arquitectura

EventMaker está desarrollado en Ruby a través del framework Ruby on Rails y JavaScript por medio de jQuery, con un enfoque MVC (modelo, vista y controlador) en donde se basa en que cada parte de la página web se encarga de una cierta área sin mezclar el código entre ellas, es decir:

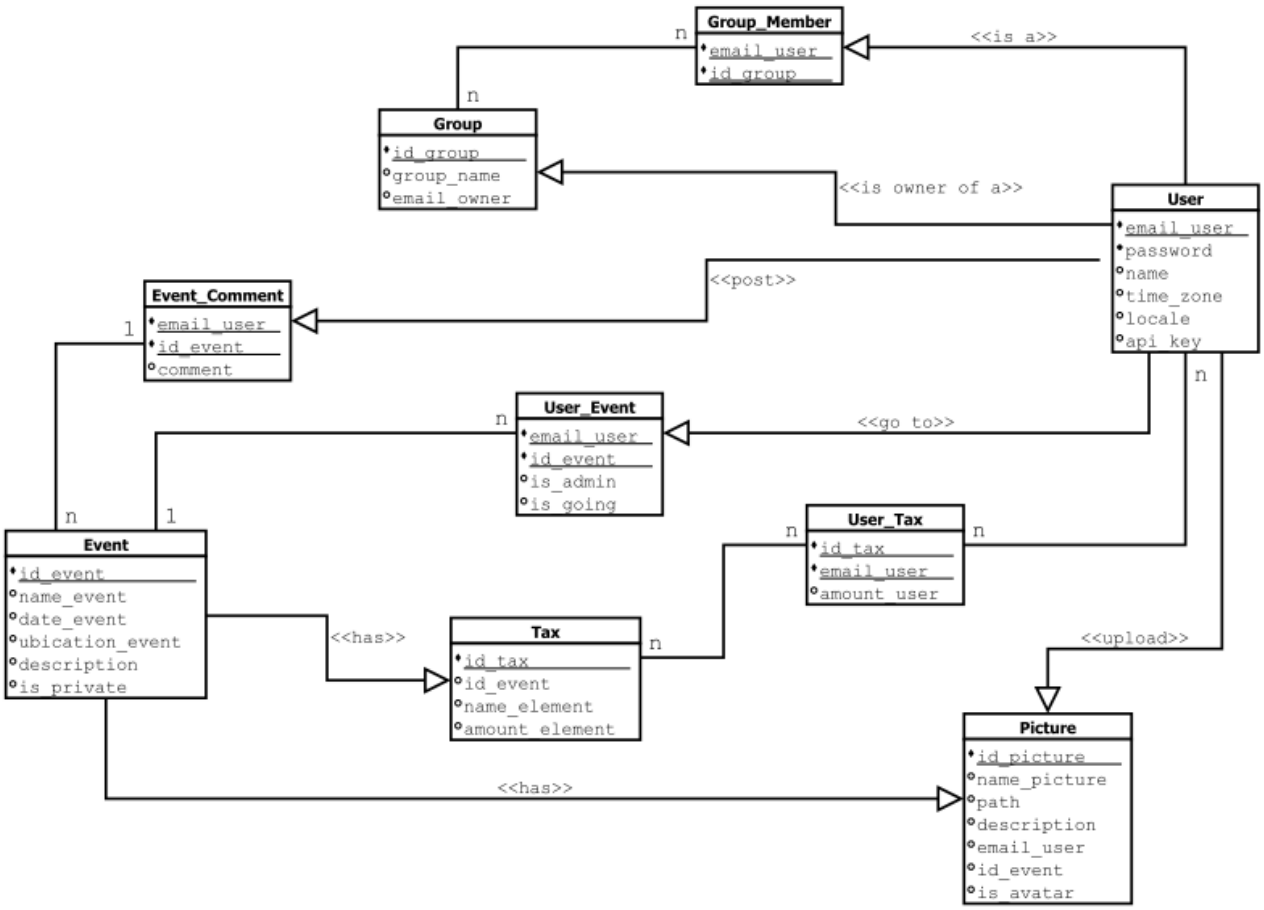
- El **modelo** es el que interactúa directamente con la base de datos (BD), y permite obtener los datos para la aplicación sin interactuar con directamente con ella.
- La **vista** contiene principalmente la producción de código html a través de Ruby, (los estilos css están en una carpeta aparte) y los archivos que producen el JSON de la API. Es decir, se encarga de la interfaz que muestra los datos a los usuarios.
- El **controlador** es el que permite la interacción entre el modelo y la vista. En general se trata de métodos desarrollados en Ruby que llama la vista, busca datos en el modelo, los procesa y luego entrega un resultado final a la vista, la que se encarga de mostrarlos finalmente.

Cada parte MVC tiene su propia ruta de archivos correspondiente. En el caso de nuestra aplicación cada modelo importante se generó de la siguiente manera:

- Tanto los *eventos*, como los *usuarios*, los *grupos*, las *fotos* y las *cosas que llevar* (taxes) fueron implementados por scaffolding (comando de rails), que crea carpetas estándar con operaciones ya hechas (como crear, modificar y eliminar), las cuales fueron modificadas en nuestra aplicación.
- Los controladores se ubican en la ruta */app/controllers* ; los modelos están en */app/models* y las vistas están en */app/views*.
- El resto de partes de la aplicación las colocamos de manera directa en las carpetas estándar, por lo que no generamos una carpeta adicional en nuestra aplicación.

3.- Modelo de Dominio

Para comenzar, nuestro modelo de dominio se basó en el actual sistema de eventos de Facebook, pero sólo enfocándonos en el manejo de eventos y mejorándolo para el usuario. Nuestro modelo de dominio se basa en lo siguiente:



- Un actor (usuario) entra a la pagina
    - Si la persona está registrada, se loguea
    - Si la persona no está registrada, entra a registrarse
- El usuario entra a su perfil
    - Usuario crea eventos
    - Usuario crea grupos
    - Usuario revisa eventos creados
    - Usuario asiste a evento
- Si el evento es creado
    - Evento se muestra en el menú correspondiente
    - Evento permite agregar fotos
    - Evento permite agregar objetos a la CheckList
    - Evento permite mandar invitaciones
    - Evento permite agregarle un avatar

- Grupo es creado
    - Grupo permite agregar gente al mismo
- Usuario entra a evento creado
    - Usuario manda invitaciones a otros usuarios
- CheckList aparece en el evento
    - El usuario puede agregar un objeto a la checklist
    - Otro usuario puede decir si va a llevar o no ese objeto
- Aparece galería de fotos desde el evento creado
    - Usuario sube fotos a la galería
    - Se pueden eliminar fotos
- Si usuario es aadmin del evento
    - Puede editar el evento
    - Puede eliminar el evento

4.- Modelo de Base de Datos:

La base de datos de la página tiene 5 tablas importantes: Users, Events, Groups, Pictures y Taxes, además de otras tablas anexas que ayudan en la interacción de éstas, las cuales son Event_comments, Guests, Group_members y User_taxes. Cabe destacar que cada elemento de la BD tiene su ID correspondiente para poder reconocerlo. Explicaremos cada una de las tablas a continuación:

a.- Events: Quizás es la tabla más importante de la página web. Guarda toda la información relevante del evento. Sus atributos son:

- Name (string): Nombre del evento
- Date (datetime): Fecha del evento
- Place (string): Lugar del evento (este parámetro es pasado al JS que muestra el mapa del lugar, que se explicará más adelante.
- Description (text): Descripción del evento, dodne el creador puede dejarles un mensaje a los asistentes al evento.
- Is_private (boolean): Indica si el evento es público o privado, para poder ordenarlo respectivamente. False indica que es público y true que es privado.

b.- Users: Clase que maneja todos los atributos importantes de un usuario.

- Email (string): Importante y tiene que ser verídico. Guarda el e-mail del usuario que le permite loguearse en la página.
- Password (string): Guarda la password del Usuario
- Name (string): Es el nombre con el cual el usuario aparecerá en el sistema, es decir, el nombre público que muestra.
- Time-Zone (string): Indica en qué región vive el usuario. Esto se le pregunta al momento de registrarse. Muy importante ya que permite
- Locale (string): Indica si el usuario habla inglés o español. Muy importante ya que la página soporta ambos idiomas.
- Api_key (string): Guarda la Api_key de cada usuario, la cual le permite al desarrollador obtener el login correspondiente para usar nuestra API.

c.- Groups: Clase que guarda la información relacionada al grupo de amigos.

- Group_name (string): Guarda el nombre del grupo.
- User_id (integer): Guarda el id del dueño del grupo, ya que asumimos que cada grupo sólo puede verlo el usuario que lo creó (similar al sistema de listas de facebook). Para agregar personas al grupo usamos otra tabla explicada más adelante.

d.- Pictures: Guarda la información de las fotos que suben los usuarios.

- Name (string): Nombre de la foto
- Path: Path en donde se guarda la foto subida a la página
- Description: Descripción de la foto, que la persona que la sube puede agregar a ella.
- Is_Avatar (boolean): Indica si la foto se subió como avatar o como foto normal. True si es un avatar.
- User_id (integer): ID del usuario que subió la foto
- Event_id (integer): ID del evento al cual corresponde la foto

e.- Taxes: Corresponde a las "Cosas que llevar" al evento, que tienen sus propios atributos.

- Name (string): Nombre del objeto que se va a agregar a la lista.
- Amount (integer): Cantidad del objeto que se necesita en el evento.
- Event_id (integer): ID del evento al que corresponde el objeto.

f.- Event_Comments: Relaciona un usuario con un evento, en donde se guarda el comentario que un usuario deja en un evento determinado (similar a facebook).

- User_id (integer): ID del usuario
- Event_Id (integer): ID del evento
- Coment (string): Comentario que deja el usuario en el evento.

g.- Guests: Tabla importante, ya que relaciona un usuario con un evento y permite el manejo de las invitaciones ya que indica si un usuario ha sido o no invitado a un evento por otro usuario.

- User_id (integer): ID del usuario
- Event_id (integer): ID del evento
- Is_admin (boolean): Indica si el usuario es admin del evento determinado.
- Is_Going (boolean): Muy importante, ya que indica si el usuario asistirá o no al evento. No es lo mismo ser invitado a un evento que indicar que realmente va a asistir.

h.- Group_members: Relaciona un usuario con un grupo e indica si un usuario pertenece a un grupo o no.

- User_id (integer): ID del usuario
- Group_id (integer): ID del grupo

i.- User_Taxes: Relaciona un usuario con un tax, indicando que ese usuario se ofrece a llevar ese tag al evento.

- Tax_id (integer): ID del tax
- User_id (integer): ID del usario
- Amount_user (nteger): Cantidad que el usuario ofreció llevar de ese objeto

5.- Código Detallado

Como ya hemos dicho, varias de las partes anteriores fueron desarrolladas con scaffolding (con arreglos más adelante modificando un poco las clases) pero existen partes del programa que no utilizaron código estándar, como las siguientes:

a.- Implementación de selección de lenguajes:

Mediante un atributo en el Usuario es posible *customizar* el idioma de la página web de forma completa (esto es gracias a I18n implementado en Rails, y una extensión similar disponible en jQuery). Cuando el usuario aún no se registra y no ha proporcionado explícitamente su idioma de preferencia, la página web carga el idioma de preferencia del explorador que esté utilizando. Hasta ahora están disponibles los idiomas inglés y español, pero es fácilmente expandible a otros idiomas por medio de archivos "yml".

b.- Registro y fotos en pop-up:

Implementamos que tanto la ventana de registro de usuarios y la ventana de la foto se mostraran en un pop-up respectivo, para lograr mayor comodidad al usuario. Esto se hizo utilizando jQuery Dialog, de jQuery UI.
El detalle de este código se encuentra en */app/assets/javascript/application.js*

c.- Invitaciones a eventos y a grupos para los usuarios:

Esto se realiza mediante un plugin de jQuery llamado TokenInput en el cuál permite realizar consultas vía AJAX a algún método del controlador. En este caso, el controlador fue *User*, y el método *search*. Al escribir un email, un nombre de una persona o de un grupo, se retorna un objeto en formato JSON con todos los resultados posibles, desplegando en la vista una lista desplegable.

d.- Javascript para indicar si se asiste o no al evento:

Esto se realiza de manera simple por medio de JQuery, el cual revisa el párrafo que indica si el usuario va o no al evento, si el usuario aceptó la invitación, muestra en un párrafo que irá al evento. En el caso que este usuario pase por el párrafo, éste cambiará a "No asistir" mientras el mouse está sobre él, permitiéndole la opción de que si lo presiona, deje de asistir al evento (similar a como funciona facebook con su pestaña de asistencia). En el caso que el usuario aún no contesta la invitación, aparecerá "Asistir" en el párrafo.

e.- Mostrar eventos por fecha en partes diferentes del menú

Esto lo hacemos desde el controlador de Event, donde tenemos varios eventos (Own, Past, Public, etc) que seleccionan de todos los eventos de la página, sólo los necesarios dependiendo de las condiciones del ítem respectivo.

f.- Fechas de eventos son localizadas

El usuario, al crear su cuenta, puede elegir su zona horaria. De esta forma los eventos muestran su fecha de forma relativa para cada usuario. Este método, *set_locale_and_time_zone*, que también se encarga de configurar el idioma, se encuentra en *application_controller.rb.*

6.- Implementación de API externa

Para nuestra aplicación utilizamos la **API Google Maps** de Google, par a poder agregar un mapa a los eventos, indicando el lugar exacto en donde ocurriría el evento (similar a Facebook). Para la implementación de esta API se utilizó lo siguiente:

- No se utilizó una gem adicional, sino que manipulamos directamente las llamadas a la API utilizando javascript y mostrándolo directamente en la página web.
- Se siguió el siguiente tutorial de la página de Google Developers (indicado en las Referencias) para las llamadas a la API, lo que explicaremos a continuación:
    - Se creó una función *initialize()* que es la que llama al mapa.
    - Esta función genera una variable *map*, a la cual se le pasa el id del div donde se va a mostrar el mapa en la página web (en nuestro caso *map_canvas*) y una serie de opciones configurables, en la cual se le pasa el tamaño, el zoom, el tipo de mapa, si se quiere estático, etc.
    - También genera una variable *geocoder*, la cual se encarga de pasarle los parámetros necesarios a map sobre las coordenadas que se quieren mostrar en el mapa.
    - Para evitar poner las opciones default, se coloca la opción *disableDefaultUI* en true.

- Además necesitamos que el mapa muestre las coordenadas que nosotros queremos, por lo cual utilizamos otra función de la API (*geocode*) que nos entrega la latitud y la longitud que queremos. Para llamarla, nos creamos una función llamada *codeAddress()* que se llama justo después de llamar a initialize(), que hace lo siguiente:
    - Lee el div donde se encuentra nuestra dirección del evento (a través de jQuery) y se la pasa a esta función de la API, que devuelve las coordenadas del lugar que le pedimos.
    - Si no encuentra las coordenadas, le pedimos que no muestre el mapa, ya que interpretamos que el usuario no entrego una dirección precisa (por ej. puso “Mi Casa”).

- Hay que tener cuidado que mientras más preciso se es al entregar una dirección, mejor será el resultado de *codeAddress()*, ya que puede que a veces por ambigüedades entregue una dirección errónea.

7.- Referencias

- Implementación de la API: (https://developers.google.com/maps/documentation/javascript/tutorial?hl=es)
- Documentación de las clases de IIC3582 1er semestre 2012
- StackOverflow: http://stackoverflow.com/
- W3School: www.w3schools.com/
- jQuery http://jquery.com/
- jQuery UI http://jqueryui.com/
- jQuery TokenInput http://loopj.com/jquery-tokeninput/
- jQuery Validation plugin http://docs.jquery.com/Plugins/Validation
- Rails i18n http://guides.rubyonrails.org/i18n.html
- i18n support for JavaScript with Rails support https://github.com/fnando/i18n-js/