Pontificia Universidad Católica de Chile

Diseño Detallado de Software IIC2113

Agosto 2014

# ENTREGA 2: SUBGRUPOS

# COMPONENTE FRONTEND

Integrantes

Tomas Dussailant

Guillermo Morales

Macarena Peralta

# DESCRIPCIÓN

Para el desarrollo del proyecto Mashup se tomó la decisión de separar la aplicación en dos. Una está encargada del backend, que posee varios componentes, y la otra, únicamente en el frontend. Esta última corresponde a todo el componente frontend. Las razones de esta decisión de diseño nacen de reunir en un sólo lugar todo lo que tiene el mismo objetivo y responsabilidad, y así desacoplar el componente de los otros. También permite la sustitución del frontend de forma directa para responder a diferentes interfaces de usuario. Por otro lado, permite replicar componentes que se vean con mucha demanda de forma independiente. Así se asegura escalabilidad y extensibilidad de la aplicación.

El componente fronted es el encargado de mostrar al usuario los resultados de todo lo que realiza la aplicación. Esto contempla el mashup general, es decir el cloud tag y el resultado de la búsqueda en las diferentes fuentes. Además permite al usuario administrar sus mashup guardados, crear una cuenta o conectarse con facebook y visualizar los mashup de sus amigos.

Está compuesto por las vistas html con las que el usuario interactúa directamente. Existen controladores encargados de ellas y del manejo de las rutas correctas, y finalmente los modelos. Estos últimos no son los modelos directos de la base de datos, no se replican los de la aplicación backend, sino que son sólo los necesarios para el manejo de las vistas, como los usuarios o el mashup. En vez de conectarse con la base de datos estos modelos llaman directamente a la aplicación backend sin saber qué hace ella, aun así recibe los objetos que están en su base de datos y sus modelos como metadata en formato JSON que se manejan como elementos de ActiveRecords a través de la gema ActiveResources. Esta misma conexión permite actualizar los modelos cuando sea necesario, por ejemplo al editar un usuario o al guardar un nuevo mashup.

Es posible justificar todas estas decisiones de diseño a partir de lo siguiente:

- **Abstracción:** La separación en dos aplicaciones permite abstraer completamente el componente visual del modelo y sus funciones. Sin embargo, el componente de frontend depende del modelo y está diseñado para sus especificaciones por lo que no es funcional por sí solo. Por lo tanto, la decisión consideró priorizar la abstracción del modelo antes que la vista para permitir posibles vistas diferentes como vistas en apps móviles.

- **Ocultamiento:** Al ser una aplicación independiente, tanto el funcionamiento del modelo como el de las vistas están ocultas tras una serie de interfaces bien definidas de comunicación. El frontend sólo solicita recursos y solicita acciones sobre recursos. El modelo realiza acciones sobre los recursos sin revelar el proceso.

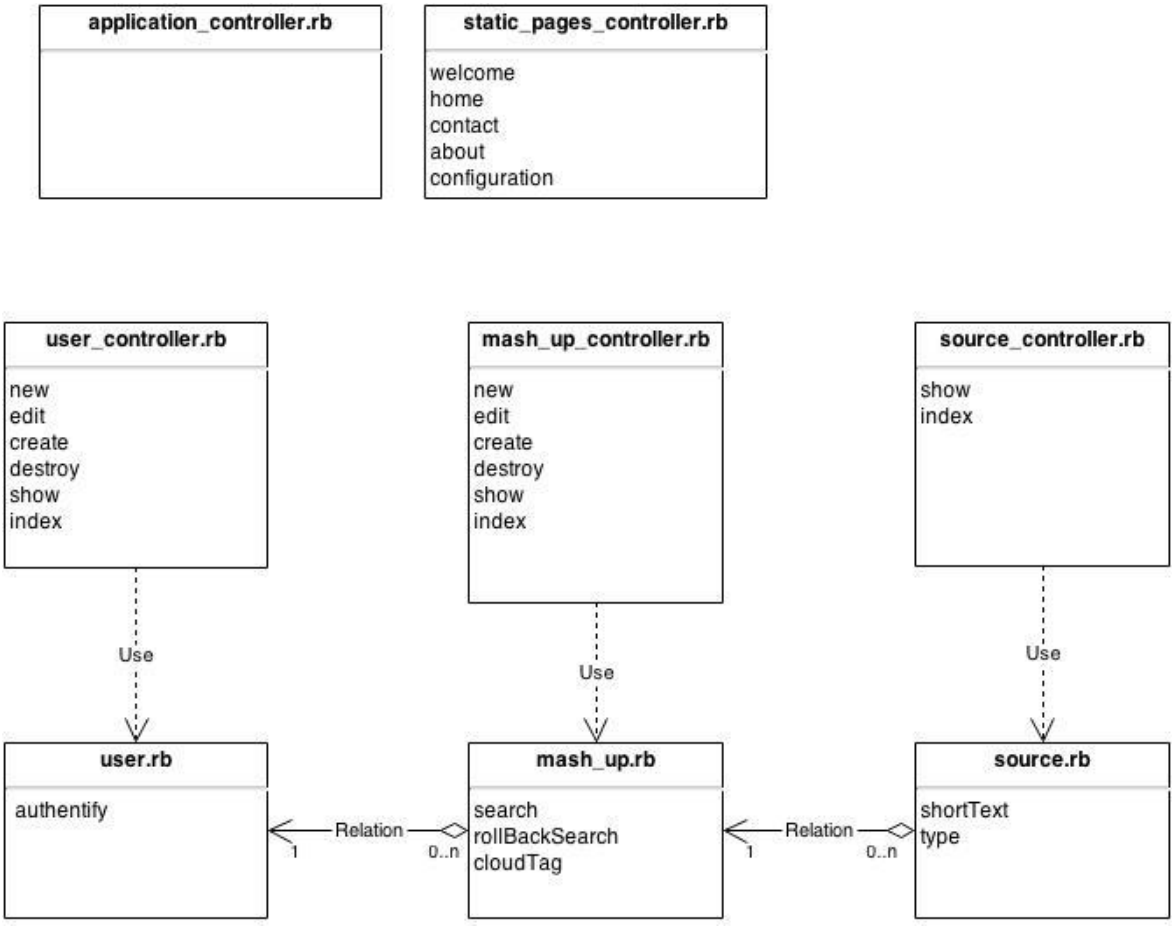
- **Cohesión y acoplamiento:** La cohesión aumenta al dividir este componente ya que tiene comportamientos bien definidos de los que no depende el resto del programa. El acoplamiento,

por otro lado, aumenta en la separación pero se estimó que las ventajas de la división superaban el overhead de la comunicación entre aplicaciones. Esto se debe a que, para aplicaciones implementadas en servidores cercanos, el costo de comunicación es bajo y los beneficios antes descritos lo justifican.

- **Code smells:** Cambios divergentes y Shotgun surgery no serán un problema debido a la clara separación entre las aplicaciones, sin embargo, habrá que estar atento a posibles “olores” de Feature envy que se puedan producir por una responsabilidad mal asignada. Lo más probable es que este problema se detecte primero en desacuerdos entre subgrupos antes que en el mismo código. Se deberá ser claro en las discusiones sobre a quién le corresponde qué, especialmente con los componentes que utilizan las interfaces.

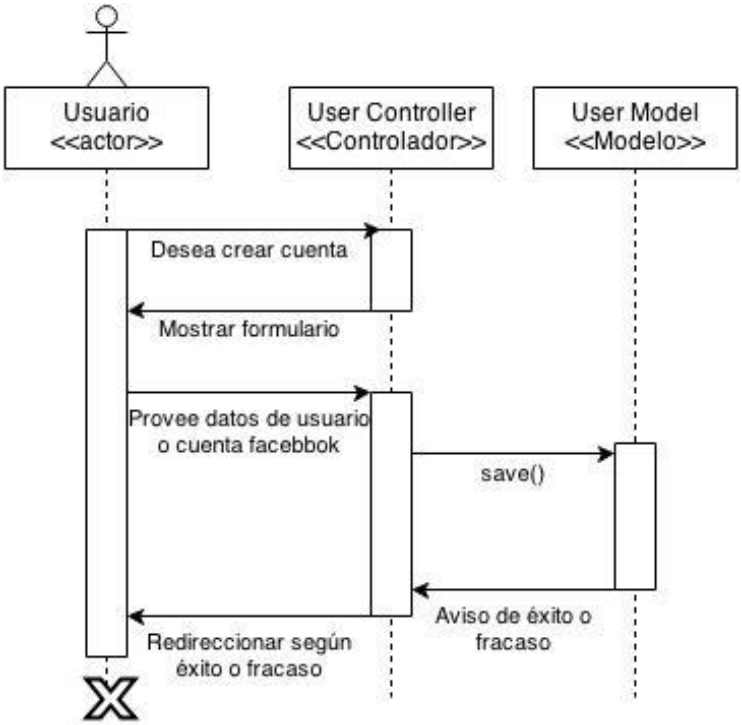
# DIAGRAMAS

## DIAGRAMA DE CLASE



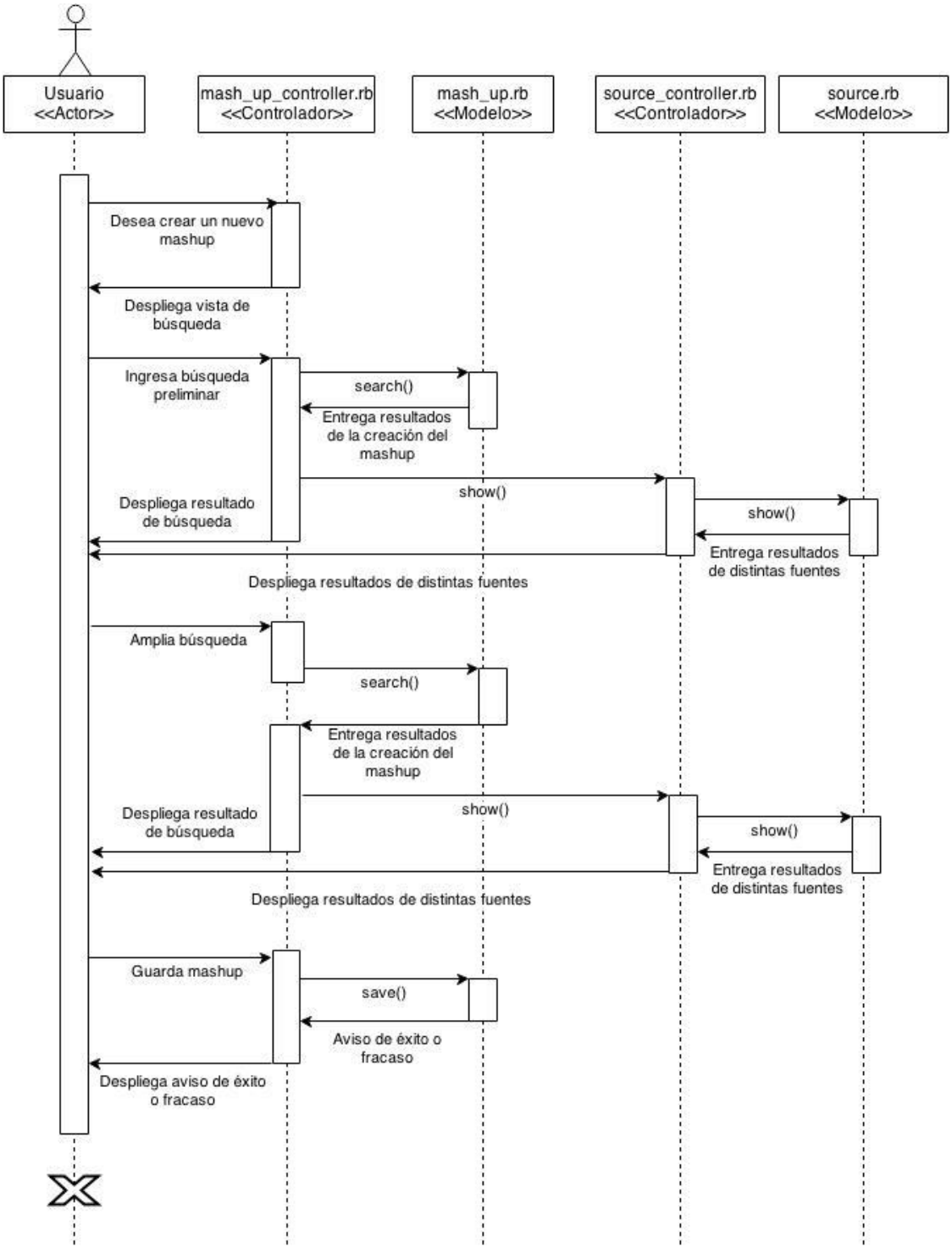
## DIAGRAMAS DE SECUENCIA

### 1. CREAR UN NUEVO USUARIO



2. CREAR UN MASHUP DESDE CERO Y AMPLIAR LA BÚSQUEDA UNA VEZ.

Esta acción consiste en crear un mashup con una búsqueda cualquiera, pero se puede ampliar si el usuario hace click sobre uno de los links del cloudtag mostrado. Finalmente el usuario guarda su mashup.



## PATRONES GOF EN EL DISEÑO:

Gracias al análisis realizado durante la creación de los diagramas, podemos identificar los siguientes patrones GoF.

1. **Cadena de responsabilidades:** Este patrón está presente en el controlador mash_up. particularmente se puede observar en el diagrama de secuencia de creación de un mashup. Aunque este controlador se encargue de desplegar las vistas y llamar la función search, la cual obtiene de la aplicación backend todo lo necesario para visualizar el mashup, delega la visualización de las fuentes al controlador Source. Con esto pasa responsabilidades a otros controladores.
2. **Sigletón:** Este patrón se observa en application_controller, ya que es una clase que permite tener los métodos y variables globales. Es una única instancia para todo este componente.
3. **Intérprete:** El patrón intérprete se encuentra presente en la implementación de los modelos. El modelo de este componente usa la gema ActiveResource, que es la encargada de recibir la información de la aplicación backend en formato json y la interpreta como un objeto de ActiveRecord. También permite la comunicación en sentido contrario, es decir ante cualquier cambio que se guarda en el modelo la gema se encarga de traducirla a un objeto json para enviarlo a la aplicación backend.