# Documentación proyecto Alma

## Equipo ToPa

### Modelo de datos

Primero están los usuarios, que tienen *username* y *password_digest* para autenticarse, y otros datos personales: nombre, apellido, email, dirección. Además tienen un *remember_token* para manejar la sesión.

Después están los almacenes, con nombre y dirección. Para reconocer su dueño, hay una tabla intermedia de privilegios que relaciona usuario con almacén. Además tiene un nivel de privilegio: por ahora solo está implementado el de administrador, pero se podrían agregar, por ejemplo, cajeros o reponedores, además de múltiples dueños.

Cada almacén puede publicar noticias (llamadas *reports* por la aplicación ya que *news* no singulariza correctamente). Estas tienen título y contenido, y opcionalmente pueden estar asociadas a un producto.

Las noticias pueden tener comentarios (*report_comments*), los que tienen un usuario dueño y un texto.

Además cada usuario puede seguir (*follow*) a un almacén a través de la tabla *followers*. Esto le da rápido acceso a ellos y además puede ver un *feed* de noticas de los almacenes a los que sigue.

Cada almacén puede tener muchos productos. Estos tienen nombre, precio y visibilidad, para ocultarlos en vez de borrarlos. Además, a través de la tabla inventario, los productos tienen *stock*.

Opcionalmente pueden tener una categoría para clasificarlos. Las categorías tienen solo nombre. Aunque pueden tener solo una categoría, los productos pueden tener infinitas etiquetas (*tags*). También solo poseen nombre.

Los usuarios pueden valorar los productos *(review*). Recibe un texto y una puntuación entre 1 y 5 estrellas. En cada *review* se crea un modelo *star* que guarda la puntuación asignada, y cada producto tiene un *star_count* que cuenta las valoraciones recibidas según estrellas. Además cada *review* puede ser comentado.

Para hacer compras está la tabla de *purchase_orders*. Cada compra está asociada a un usuario y un almacén. Para que pueda contener varios productos junto a la cantidad deseada, contiene instancias del modelo *order_line*, con la columna *amount.*

Por último, hay modelos para manejar las imágenes de almacén, producto y usuario. Estos son *grocery_image*, *product_image* y *user_image*. Cada uno tiene un *string* que guarda el nombre del archivo de la imagen.

## Arquitectura de la Aplicación

Como buena aplicación de Rails, nuestra aplicación tiene una arquitectura de Modelo, Vista y Controlador o MVC.

## Detalles interesantes de implementación

1. Manejo de assets

   Todo el css y el javascript se encuentra en archivos apartes. La única excepción es el uso de <script> tags en html cuando se necesita inicializar variables con datos dinámicos que no pueden ser harcodeados en archivos js estáticos, pero todos los métodos que trabajan sobre esos datos están debidamente ubicados en archivos javascript.

   Todo el css, javascript e imágenes que nosotros manejamos por nuestra cuenta se encuentran en el directorio app/assets. Otros archivos como jquery y jquery-ui son invocados por medio de gemas. En el caso de plugins de jquery que no fueron incorporados por medio de gemas, estos se tuvieron que incluir en el proyecto ubicándolos en el directorio vendor/assets. En el caso de plugins que dependen de imágenes, éstas tuvieron que ser ubicadas en el directorio public/images con el fin de eludir el sistema de precompilación de rails para evitar que los nombres de los archivos fueran modificados (ya que rails agrega un hash md5 a cada archivo que precompila).

   Los assets se pueden considerar en 2 grupos, aquellos que son precompilados junto con Application.js y Application.css al ser referenciados por estos, y aquellos archivos que se precompilan aparte especificando sus nombres y extensiones en el archivo assets.rb. Nuestra idea es que los archivos precompilados junto con Application.js/css estén siempre disponibles a nivel de aplicación, y por lo tanto estos son incluídos en cada layout. El segundo grupo es más versátil ya que estos se pueden incorporar a nivel de layout o bien a nivel de vista. Si se quiere incluir cierto archivo css y/o cierto archivo js en una determinada vista específica, entonces se puede definir un content_for :head y el layout respectivo los incluirá por medio de yield :head.

2. Implementación del Dashboard

   El Dashboard funciona parecido a un Single Page Application. Para acceder a él se debe ingresar al index de purchase_orders para una grocery particular. A grandes rasgos funciona teniendo arreglos de datos en javascript que guardan información

relevante para la página, y cuando se necesita interactuar con el servidor para actualizar algo de la misma página, se utiliza ajax. Hay 2 instancias de uso de ajax: 1) cuando se hace una búsqueda de órdenes de compra por período de tiempo y 2) cuando se quieren los detalles de una orden de compra en particular. En el primer caso lo que se recibe es un archivo json, el cual es procesado a nivel de javascript para que pueda ser interpretado por un plugin que renderea un gráfico de curva. En el segundo caso lo que se recibe es un archivo javascript que actualiza un  con los contenidos de los detalles de la orden de compra indicada. Por motivos de optimización se cachean las órdenes de compra en un map a nivel de javascript para evitar redundancia de queries vía ajax.