regalar

Diseño de Arquitectura

Fecha: 06/2016

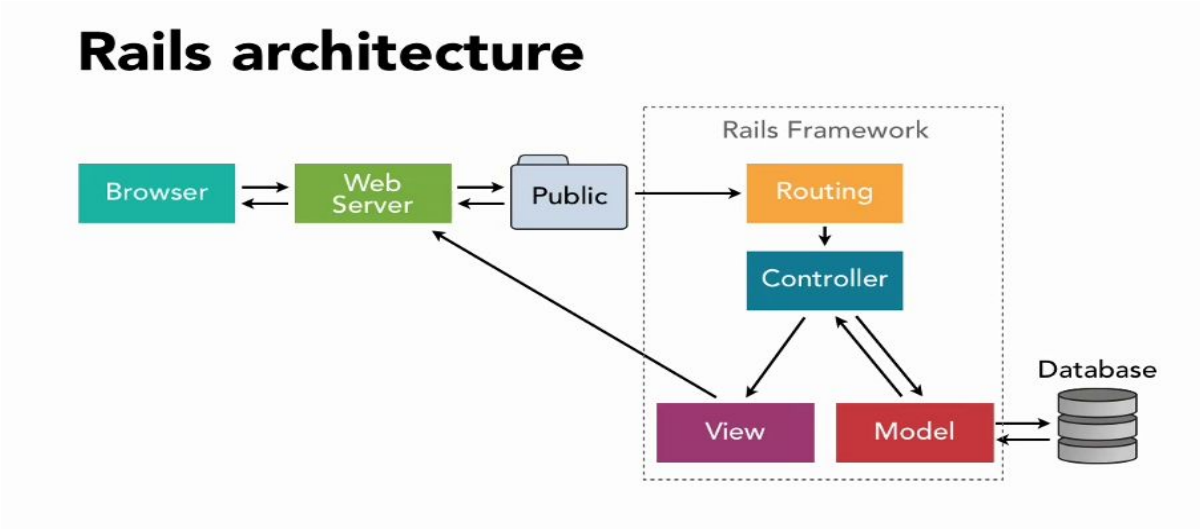
Responsables:

- Luciana Reznik
- Maria Victoria Mesa Alcorta
- Federico Ezequiel Di Nucci

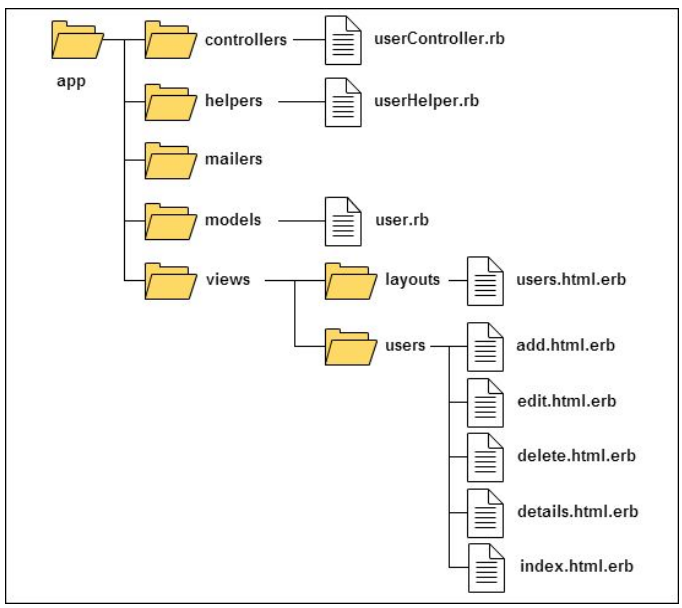
Objetivo

El objetivo de este documento es brindar al lector una visión global y comprensible del diseño general de la arquitectura de Regalar

Diseño Arquitectónico



Se planteó una arquitectura cliente servidor donde este último, es una aplicación programada en Ruby on Rails, la rapidez en el desarrollo de proyectos con Rails está fundamentada en la idea de construir la aplicación separando de forma clara las capas de Modelo, Vista y Controlador para reducir el acoplamiento entre la lógica de negocios y la de presentación. Para simplificarlo, provee por default una organización de directorios la cual simplifica la separación entre ellos de la siguiente manera :



Otra gran ventaja de Ruby son las gemas, que son plugins con funcionalidades específicas para nuestros proyectos. En el caso de regalar, las gemas utilizadas más significativas son las siguientes:

Pg: Permite utilizar postgreSQL como base de datos para el proyecto.

Carrierwave/fog: Simplifica el ABM de imágenes integrándose con el servicio de almacenamiento de amazon S3 y persiste en la base de datos solo el path a la misma.

Devise: es una solución completa para la autenticación de usuarios que nos permite guardar hashes de las contraseñas dentro de la base de datos para autenticar al usuario, confirmar un usuario con mail de confirmación, resolver el olvido de contraseña con envio de mail y expirar la sesión luego de un tiempo determinado.

Lenguajes Involucrados

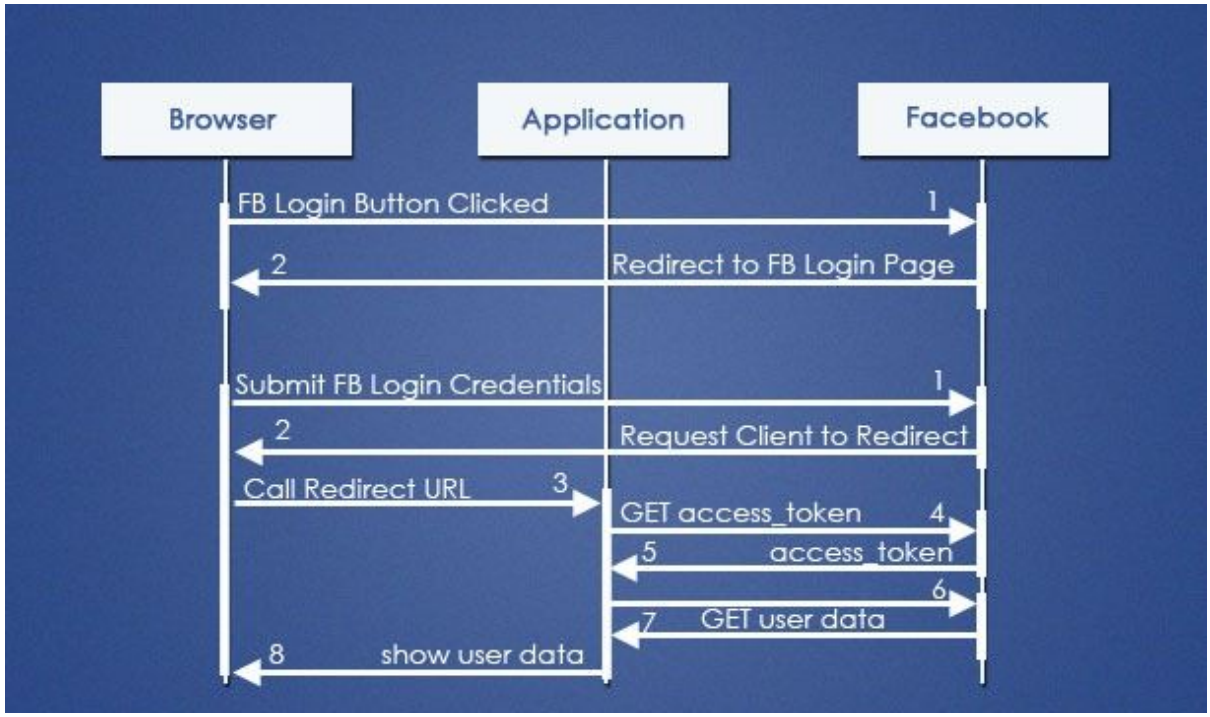
Para el desarrollo del proyecto se utilizaron los siguientes lenguajes:

- Ruby on Rails
- HTML + Slim
- Javascript
- CSS
- SQL

Integración con servicios externos

Durante el transcurso del proyecto se realizaron integraciones con los siguientes servicios externos:

1. **Facebook:** Para autenticar al usuario se utilizó la API provista por facebook que implementa el protocolo Oauth2. El mismo es un protocolo de autorización que permite a terceros (clientes) acceder a contenidos propiedad de un usuario (alojados en aplicaciones de confianza, servidor de recursos) sin que éstos tengan que manejar ni conocer las credenciales del usuario de donde se extrajo datos como el mail, nombre, apellido y foto de perfil para ser usada en RegalAR y poder logear de una manera rapida y sencilla sin el uso de una contraseña. El proceso para autentificarse con facebook se puede ver a continuación en el siguiente diagrama:



El protocolo cuenta con 2 etapas

Etapa A:

1. Primero se contacta a una URL para validar el Application ID.
2. Si el mismo es válido se redirige a la pagina de login.

Etapa B:

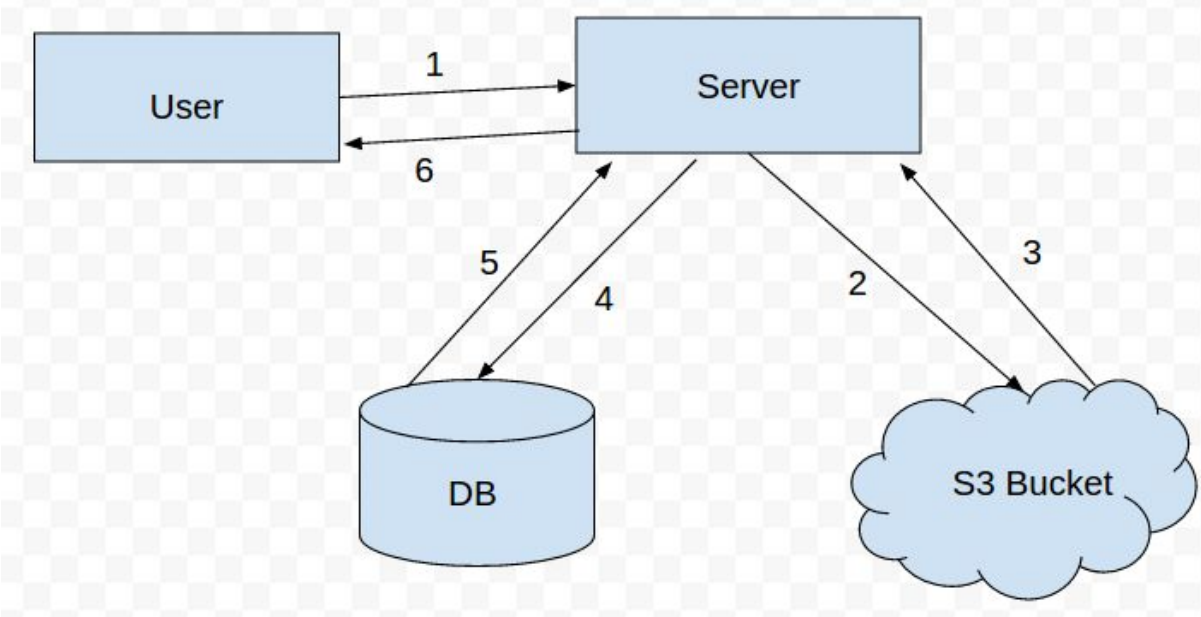
1. El usuario envía el formulario con sus credenciales.
2. Facebook valida las credenciales ingresadas por el usuario y redirige a la redirect_url que se tenga configurada en la Facebook App creada en la pagina de Developers.
3. El browser llama a la URL de redirección.
4. Se llama de vuelta a la página de facebook para pedir el token de acceso.
5. Facebook responde con el token de acceso.
6. Se pide la información del usuario utilizando el token de acceso.
7. Facebook valida el token y responde con la información pedida.
8. Nuestro servidor recibe la información y crea/loguea el usuario según sea el caso.

También se agregó la funcionalidad para poder “likear” y compartir las publicaciones en la red social.

2. **Google:** Al igual que en facebook se utilizó su API pública para resolver la autenticación de los usuarios, también implementa Oauth2.

3. **Twitter:** Se agrego la posibilidad de poder twittear los deseos de las diferentes organizaciones para asi poder obtener donaciones y nuevos usuarios.

4. **Amazon S3:** Se utilizó para resolver el problema del almacenamiento de las imágenes subidas por los usuarios se utilizó un servicio de Amazon llamado S3 y se integró de forma tal que en nuestra base de datos solo se guarda la ruta al archivo dentro de nuestro servicio.



1. El usuario envía el formulario con la imagen al servidor
2. El servidor procesa el pedido y envía la imagen a Amazon para ser almacenada
3. Amazon responde que la imagen ha sido almacenada correctamente
4. Se crea en la base de datos una nueva entrada para la imagen donde no se guarda el contenido de la imagen en sí, sino sólo una referencia a cómo encontrarla en el bucket de S3
5. La base de datos responde que la nueva tupla ha sido creada
6. El servidor responde al usuario que la imagen fue guardada exitosamente

URL Semánticas

Se implementó un esquema de URLs semánticas para generar direcciones más amigables y entendibles para el usuario, para eso se reemplazaron los ids numéricos en las URLs por los nombres representativos de las entidades **Usuario** y **Organización**. Gracias a eso se

logró que el path cambie, por ejemplo, de /users/1/organizations/25/administrate a /users/dinu/organizations/sabf-south-american-busines-forum/administrate.

Las URLs semánticas también ayudan al SEO del sitio permitiendo rankear mejor en las búsquedas orgánicas de Google atrayendo más tráfico al sitio.

Calidad de Código

Lograr mantener una buena calidad de código a través del tiempo es una labor complicada, más cuando hay muchas personas aportando al mismo proyecto y este crece en tamaño. Para combatir este problema se utilizaron las siguientes soluciones:

Rubocop: Es un analizador de estilo de código configurable el cual nos permite mantener un standard de calidad. Entre las cosas que analiza se encuentra la longitud máxima de caracteres por renglón, complejidad de métodos, indentación y convenciones básicas de Ruby.

Rspec: Es una librería de testeos de unidad que nos sirvió mucho para mantener las funcionalidades estables a medida que se iteraba o agregaban otras.

Travis: Es un servicio de integración continua usado para buildear y testear proyectos de software hosteados en GitHub. Lo utilizabamos para correr automáticamente los tests y los chequeos de estilo en cada pull request, asi sabíamos que cuando algo era mergeado a master no rompería funcionalidad pre existente.

Decisiones Tomadas

Lenguaje de Programación

Como se mencionó anteriormente, se optó por utilizar como lenguaje de programación para el servidor Ruby on Rails, esto fue asi porque es facil de implementar, tiene una gran comunidad open source por detrás, lo cual favorece el encuentro de soluciones a la hora de enfrentarse a problemas y a diferencia de otros lenguajes cuenta con un aprendizaje mucho más rápido, lo cual nos ayudo mucho ya que no todos los integrantes del grupo lo manejábamos. Aparte de esto, facilita y motiva a utilizar el patrón de diseño MVC con muchas convenciones que vienen dadas por el lenguaje.

Base de Datos

Teniendo en cuenta los requerimientos que necesitábamos cumplir, se decidió por utilizar una base de datos relacional sobre una no relacional ya que para comenzar, teníamos un modelo bien marcado y pensado para cubrir las necesidades de nuestros modelos, por lo que nos venía bien tener un esquema. Además, las base de datos relacionales son mucho más performantes para realizar consultas con JOINs lo cual usamos mucho y para evitarlas deberíamos haber replicado mucha información y habria momentos durante los cuales la base no sería consistente.

Metodología de Trabajo

Como metodología ágil de desarrollo se utilizó algo muy parecido a Kanban ya que nos basamos en instancias incrementales del producto y contábamos con una lista de tareas de las cual se tomaban requerimientos on demand. Para la organización del proyecto se utilizó Trello que es un organizador de tareas, el cual posee Listas de Cards donde para nosotros, cada card refería a un trabajo a realizar, y cada lista representa un estado de dicho trabajo (Backlog, Doing, Done), esto tiene como ventaja que tenemos en mente el scope de las cosas que estamos realizando con una descripcion de lo que debe hacer y además se puede asignar un responsable, lo cual ayuda a que dos personas no realicen el mismo trabajo. Cada funcionalidad que se agregaba debía estar debidamente testeada y cumplir con el código de estilo definido. Aparte de eso, se creaba un Pull Request en github que era examinado por otro miembro del equipo y si el mismo no era aprobado, debían hacerse los cambios pertinentes para que asi lo fuese y recien ahi se podía mergear a la línea base del proyecto.