Para lograr un diseño arquitectónico que pudiese satisfacer la calidad de producto esperada, hicimos uso del método ADD (Attribute-Driven Design). De hecho, con dicho método, hemos podido generar una arquitectura de software, y luego un sistema, que cumple satisfactoriamente con lo estipulado.

El método como ya sabemos, propone recursivamente, descomponer el sistema tratando de asegurar en cada iteración, los atributos de calidad de mayor prioridad entre los que aún no fueron atendidos. Al final, a través de la elección de tácticas y patrones, todas las cualidades deberán haber sido satisfechas.

De este modo, comenzamos evaluando los requerimientos de calidad, requerimientos funcionales, descriptos antes. También se tuvieron en cuenta algunas restricciones de diseño que nos impusimos. El análisis necesario generó un listado priorizado de los mismo, necesarios como entrada para comenzar el proceso recursivo.

Entre las restricciones mencionadas, elegimos como lenguaje de programación PHP versión 5 debido a que contamos con experiencia suficiente para poder sacar provecho del mismo. Adicionalmente, dado que hemos optado por hacer uso de una API de Georreferencia y localización provista por HTML5, entonces es necesario que dicha tecnología sea compatible con las decisiones que se tomen.

Comenzando con el proceso de priorización propiamente dicho, consideramos que el atributo más importante a tener en cuenta, es el de portabilidad. Es prioridad primera de PlaceOn, poder ser accedido desde la mayor parte de los dispositivos disponibles hoy en día. La forma de expresar estos atributos y lo que se necesita para su cumplimento, es definiendo seis factores para el denominado escenario de calidad. Para el requerimiento mencionado como prioridad, el escenario de calidad será:

**Fuente**: Usuario final

**Estímulo**: Desea acceder al sistema desde cualquier dispositivo

**Artefacto**: Sistema

**Entorno**: Runtime

**Respuesta**: El sistema es completamente accesible

**Medida de respuesta**: Acceso 100% funcional desde cualquier dispositivo que cuente con un browser compatible con html5 y API de Geolocalización

Acordamos tomar como segundo en el orden de prioridades, lo referido a la usabilidad. Buscamos apuntar a una experiencia de usuario adaptada a los dispositivos móviles, los cuales poseen espacio de pantalla acotado, y generalmente sensores táctiles. Entonces el escenario que deseamos satisfacer es:

**Fuente**: Usuario final

**Estímulo**: Desea utilizar eficientemente el sistema desde un dispositivo móvil

**Artefacto**: Interfaz de usuario del sistema

**Entorno**: Runtime

**Respuesta**: El sistema provee una interfaz cómoda para un dispositivo móvil

**Medida de respuesta**: La información del sistema debe ser completamente visible y accesible en pantallas de 4'' o más

Como tercer atributo de calidad deseable, añadimos la necesidad de poseer un sistema modificable, especialmente en el punto clave de lo que ahora denominaremos "información compartida", que no es más que la unidad de información que deseamos que la red PlaceOn, permita hacer circular entre sus usuarios. Como objetivo de este requerimiento, se halla el deseo de poder fácilmente, añadir o modificar tipos de información. Sólo por citar ejemplos, actualmente nuestro sistema permite compartir textos, enlaces con vista previa e imágenes, pero bien podríamos desear en alguna otra instancia, proveer soporte para videos, mensajes con múltiples destinos, etc.

Definimos nuestro escenario:

**Fuente**: Desarrollador

**Estímulo**: Desea añadir una nueva clase de "información compartida"

**Artefacto**: Sistema

**Entorno**: Tiempo de desarrollo

**Respuesta**: Se realiza el cambio sin afectar otras funcionalidades

**Medida de respuesta**: Se consigue el resultado final luego de 1 día de trabajo

Estos tres requerimientos de calidad, son según el criterio propio, nuestros tres Architectural Drivers. Esto es, del conjunto entero de requerimientos funcionales y de calidad, observamos en estos la posibilidad más alta de tener impacto en el diseño de la arquitectura.

Entonces nos hallamos ante la primera decisión arquitectónica a ser tomada. A priori, es necesario hacer una subdivisión de lo que podemos llamar "Sistema", que no es más que la idea menos detallada de una arquitectura.

El primer requerimiento en el orden de prioridades, indica que deseamos que la aplicación sea accedida desde múltiples dispositivos. Es bastante lógico pensar que una aplicación web, que necesita disponer de datos centralizados, información compartida entre usuarios, y que deseamos se comporte como una aplicación web, nos lleve a decidir como primer medida arquitectónica, que será necesario poseer componentes del tipo Cliente y Servidor.

Las razones y los razonamientos que nos hacen optar por este tipo de arquitectura, están basados sobre todo en la experiencia tanto personal, como ajena, ya que está comprobada la efectividad de sistemas de este tipo, a lo largo del tiempo y el espacio, especialmente en aplicaciones que son accedidas a través de la Internet.

Lo que esperamos entonces a través de esta estructura, es poder garantizar el acceso al Servidor, desde múltiples clientes. Cabe destacar aquí, que también al apegarnos a las restricciones impuestas, también dejamos sentado el hecho de utilizar protocolos HTTP/HTTPS como esquema de comunicación, favoreciendo así el acceso al cliente masivamente, necesitando para ello solamente de un navegador o browser compatible con HTML5 (prácticamente cualquier navegador conocido en sus últimas versiones).

Cliente

Servidor

Sistema

Los mecanismos e interfaces de comunicación entre estos dos componentes, como ya hemos mencionado, se apegan al protocolo HTTP/HTTPS con el fin de garantizar la compatibilidad con navegadores web, siendo ese un punto clave en nuestro sistema.

En este punto, consideramos cubierto el primer Architectural Driver y de esta manera, pudimos concluir una primera descomposición que resultó en una arquitectura general establecida, nos hallamos en condiciones aptas para seguir adelante en el refinamiento y descomposición subsiguiente.

Buscando alcanzar el objetivo de brindar una experiencia de usuario adaptable a los diversos potenciales Clientes (haciendo referencia al dispositivo físico propiamente dicho, desde el cual se accede al sistema, volviéndolo una instancia del artefacto arquitectónico "Cliente") hemos analizado un par de alternativas de las que daremos cuenta a continuación.

Aquí las dos alternativas que consideramos, eran por un lado, el diseño de aplicaciones Cliente específicas para cada plataforma que deseábamos alcanzar, por el otro una estructura arquitectónica genérica que permitiera un desacople entre la lógica que se utiliza para alcanzar los requerimientos funcionales, y la forma en que esa información es presentada a los usuarios. En otro apartado del presente informe, haremos una comparación más detallada entre las denominadas App Nativas y las Web Apps. Más allá de las cuestiones que han hecho inclinar la balanza para uno de los lados, es importante no perder de vista un par de consideraciones. Primero, la restricción impuesta del lenguaje PHP, que vuelve complejo sino imposible, el hecho de programar aplicaciones nativas (que suelen estar atadas a un lenguaje específico según en que dispositivo o sistema operativo se busque ejecutar la aplicación), y segundo, la decisión tomada en el paso anterior, de apuntar a generar un cliente capaz de correr sobre navegadores.

Con estos hechos a la vista, llegamos a concluir que será beneficioso el empleo de una arquitectura Model-View-Controller (MVC), que nos facilite el desarrollo por separado de los componentes lógicos pertinentes al modelo de negocio, y de la presentación de la información a los usuarios. También en un apartado posterior, se dará cuenta de un framework MVC hecho en PHP, elegido para el desarrollo del sistema.

Cliente

Servidor

Modelo

Controlador

Vista

Vista

Sistema

Aquí cabe destacar los mecanismos de comunicación entre los componentes. Dentro del artefacto Modelo, tenemos que reflejar todas responsabilidades pertinentes a la lógica de negocio del sistema, control de integridad de datos, mapeo hacia la base de datos, y obtención y guardado de información. El Controlador claro, el quien en efecto deberá recibir, responder y atender las peticiones propiamente dichas, que llegarán a través de las interfaces dispuestas por el protocolo HTTP, desde los clientes. También se mantiene desde el Controlador, comunicación bidireccional con el Modelo, ya que enviará los pedidos tanto de guardado como de búsqueda de datos.

La denominada Vista, como se ve, se encuentra dividida tanto en el Cliente como en el Servidor. Esto es debido a que parte de la información que será mostrada a los clientes, será enviada de regreso ante las peticiones HTTP realizadas. Es decir, el Servidor generará buena parte de las vistas, y las retornará para ser mostradas por los navegadores webs que corren en los clientes. Por otro lado, considerando como parte de la Vista, el código Javascript y HTML que están en el navegador, es que dividimos dicho componente de este modo. El cliente, al interactuar con el navegador, y ejecutar peticiones a través del mismo, vía AJAX y usando el mismo protocolo HTTP, se comunica con el Controlador que estará atento a dichos mensajes entrantes.

Luego de haber cubierto el segundo Architectural Driver, con el resultado arquitectónico mostrado, procedimos al análisis del considerado como tercer y último requerimiento.

Con el fin de favorecer la facilidad a la hora de modificar lo que abstractamente denominamos "información compartida", es que decidimos valernos de varias tácticas que son pertinentes en este caso.

Ya aquí estamos un paso adelante. Sabemos que éste, es un punto crítico en el cual es muy probable que sean necesarias modificaciones, agregados, etc. Es una de las tácticas más valiosas para prevenir efectos en cadena (cambios que conllevan otros, y otros, aún en sitios "lejanos" arquitectónicamente hablando).

En ese mismo sentido, separar la pequeña sección de modelo que refleja dicha información en un módulo abstracto más genérico, supone también que los puntos de conflicto se hallen localizados, es decir se espera así poder acotar los lugares que deban ser cambiados ante la eventual necesidad.

Estas tácticas que esperan favorecer en buena medida la modificabilidad del sistema en el punto clave, nos hacen pensar en un módulo que se desprenda del modelo general.

Cliente

Servidor

Modelo

Controlador

Vista

Vista

Módulo de Información

Sistema

En cuanto al modo de interacción con el nuevo módulo, deseamos proveer similares interfaces a las que existen entre Controlador y Modelo que hemos mencionado antes.

Habiendo provisto al sistema de características que solucionan y resuelven las necesidades planteadas como prioridad, definidas como Architectural Drivers, hemos podido iterativamente, definir una arquitectura que se ajusta a la medida.

Todo este proceso que hemos atravesado, nos permitió no solamente obtener la arquitectura previa al desarrollo, sino que nos ayudó enormemente para poder observar los problemas, las diversas soluciones alternativas, como así también constituye una experiencia invalorable personal y profesional para nosotros. No es del todo cotidiano, a pesar de ser ambos desarrolladores avanzados, tener esta posibilidad de tener injerencia directa en las decisiones arquitectónicas de los proyectos en los cuales nos hemos involucrado en nuestra carrera profesional, de manera que esa es sino la más rica de las experiencias obtenidas del presente trabajo.