Juan Manuel Lizarazo

Daniela Mariño Rodríguez

Andrés Felipe Moreno

María Camila Hernández

# Experimento 3 - Grupo 3

## Patrón Arquitectónico

**Singleton**: Para la implementación que realizamos de seguridad en la aplicación, se usó de Deadbolt, el cual es un mecanismo de autorización para la definición de los derechos de acceso a ciertos métodos del controlador y parte de la vista. Este acceso le permite determinar los permisos que tendrán los diferentes componentes del proyecto, generando escenarios de autorización específicos y genéricos. En nuestro caso, usamos escenarios de autorización genéricos, y en especial para dos casos: El primero es para los médicos y los pacientes quienes cada vez que deseen acceder al sistema deberán obligatoriamente loguearse mientras que hay un tercer actor, las urgencias, quienes pueden ser un médico o el mismo sistema quienes en una situación de emergencia de un paciente pueden acceder al sistema sin necesidad de utilizar un log in para hacer uso de la aplicación. Por ejemplo, la clase "MyHandlerCache" define los nombres de los controladores utilizados en la aplicación, para asegurarnos de que usamos la clave del controlador predeterminada necesario para que cada uno de los actores representados en los controladores efectivamente pueda reutilizar está clase cada vez que haga uso de su clave. Por lo que podemos afirmar que en este caso se puede restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto, como son las claves de ingreso.

**MVC (Modelo, vista, controlador):** Al utilizar Play para desarrollar el proyecto, estamos haciendo uso de este patrón cuyo funcionamiento está basado en separar los datos y la lógica de negocio de la aplicación de la interfaz de usuario, lo que nos permite reutilizar el código y la separación de conceptos lo que a la vez facilita la tarea de desarrollo de aplicaciones y de su posterior mantenimiento.

***Decorator***: En el caso de la interfaz, con el main podemos ver este patrón debido qué esta página tiene el menú principal y las dependencias. Así, las otras páginas solo llaman al main y le ponen al cosas adicionales que cada una contiene.

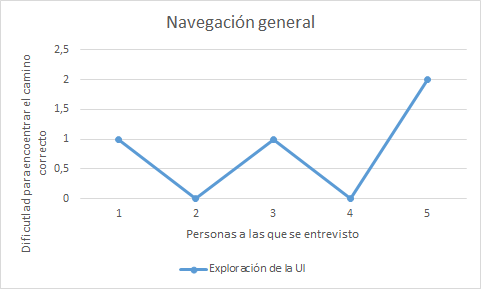
***Mediator:*** El router que tiene el proyecto contiene todas las rutas que son consumidas por los diferentes servicios que ofrece que la aplicación. Así que el router no solo las administra sino las maneja ya que toda la lógica desde que se quiere empezar a consumir un servicio.

***Wrapper:*** Para este patrón creamos la clase MedicoEnfermo, la cual es usado en el caso en que un médico que ya se encuentra en el sistema como médico se enferme. Por lo anterior, el sistema también debería poder ofrecerle los servicios que un paciente posee, pero para no crear un perfil nuevo del médico, es decir, tener en nuestro sistema al médico como dos personas diferentes sea hace uso del Wrapper, garantizando que el medico tenga toda la atención necesaria para recuperarse y luego posteriormente seguir solo como un médico en el sistema.

***Responsive:*** Debido a que la página web funciona tanto en computadores como en celulares, lo que le da las herramientas a los diferentes usuarios de hacer uso de la aplicación en cualquier momento y lugar. Haciendo que la usabilidad sea utilizada desde la perspectiva que el usuario prefiera. Mientras que la modificabilidad se ve beneficiada ya que se podrá obtener información en tiempo real de todo lo que ocurre ya que los usuarios pueden hacer uno de la app en cualquier lugar.

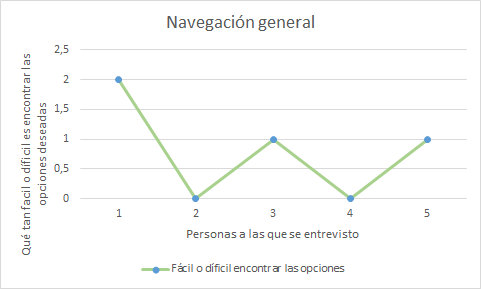
## Instrumento de crítica constructiva aplicado al menos a 5 usuarios

* ***Pregunta 1:*** ¿Qué tanto deben explorar los usuarios la UI para encontrar las opciones que ellos desean utilizar?



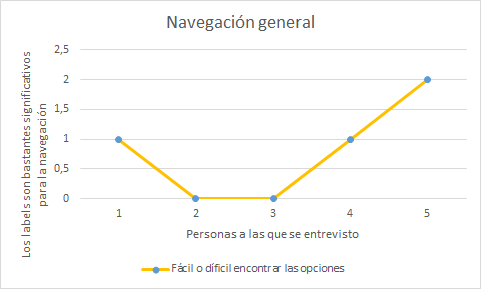
Donde 0 significa que la mayoría de los usuarios conocen el camino correcto mientras que 5 significa que la mayoría de los usuarios deben explorar muchas opciones para encontrar lo que quieren hacer.

* ***Pregunta 2***: En general, ¿ Qué tan fácil o difícil será para los usuarios encontrar las opciones que ellos desean al utilizar el sitio?



Donde 0 quiere decir que es muy fácil mientras que 5 es muy difícil.

* ***Pregunta 3***: ¿Los labels de la UI son bastantes significativos para insinuar qué opciones de navegación ofrece cada uno?



Donde 0 quiere decir que los labels son muy insinuantes mientras que 5 quiere decir que son poso insinuantes.

* ***Pregunta 4***: ¿ Qué problemas sobre el sitio contribuyen a las calificaciones anteriores?

El menú superior no es claro como cambiará dependiendo del usuario. No hace mucho sentido que la página sea igual antes y después de log in.

## Análisis de los requerimientos, diseño de elementos de arquitectura, diseño de la solución, implementación y pruebas. Se cuenta con artefactos que soportan el proceso y con un depósito de las fuentes.

**Diseño de Elementos de Arquitectura:**

La arquitectura manejada fue la misma utilizada en entregas pasadas, con la aplicación en el framework de play funcionando con una base de datos implementada en PostgreSQL. Para este Experimento, se buscó implementar polticas de auto-escalamiento y ejecutar las pruebas de carga sobre la arquitectura para garantizar el auto-escalamiento. Adicional a esto, se implementó una interfaz web para el manejo de la aplicación por parte de los medicos, en donde puedan manejar su información, enviar consejos y ver la información de los médicos que hacen parte del hospital. Con esto en mente, también se desarrolló una interfaz de uso para el paciente, en donde puede ver las ltimas mediciones de su artefacto, editar su información y leer los consejos que el médico tratante envía. Para el auto-escalamiento, la aplicación fue subida a Heroku, el cual garantiza esta métrica implementando la aplicación en un sistema bajo demanda. Otra herramienta que ayuda bastante con esta metrica es Hire-Fire, la cual implementa las políticas de los dynos de Heroku y se utiliza para monitorear diferentes características de la aplicación desplegada en Heroku. Esta aplicación se encarga de monitorear métricas sobre los dynos desplegados en Heroku, para aumentarlos y disminuirlos dependiendo de la necesidad y la estrategia de auto-scaling propuesta. Por último, Loader.io debió ser implementada, la cual permite realizar las pruebas de carga a la aplicación web de forma distribuida, garantizando que heroku las adopte y las permita.

## Decisiones de arquitectura

Debido a que el proyecto se creó desde un principio son Play se han podido ir garantizando varias características que han favorecido al proyecto:

1. ***Stateless:*** Da una mejor escalabilidad horizontal de las aplicaciones. Ya que con nuestro balanceador logramos que cuando llega una petición esta pueda ser enviada a cualquier nodo, sin importar en cual se ha dado respuesta en ocasiones anteriores.
2. ***Peticiones asíncronas***: En lugar de esperar , el thread puede continuar procesando otras peticiones, y sólo volverá al actual thread cuando la petición de I/O hay finalizado. Esto permite la realización de muchas llamadas concurrentes sin tener que esperar como ocurre con los threads.
3. ***Auto-Escalamiento:*** Como se mencionó anteriormente, implementamos esta métrica con aplicaciones como Heroku y Hire-Fire. Lo primero que se hizo fue subir la aplicación a Heroku a partir del repositorio de Github, se agregaron y modificaron archivos necesarios para garantizar que la aplicación pudiera compilar en la página. Otro paso a seguir fue crear una cuenta en Amazon Web Services, la cual va a proveer la base de datos de ahora en adelante creando Buckets de datos y garantizando la seguridad por medio de llaves, las cuales son secretas en cada cuenta y garantizan que la información no vaya a ser adulterada.

Además al tener una base de datos relacional, podemos acceder a la información de diferentes actores de una manera más sencilla ya que solo es necesario hacer los filtros adecuados sobre las búsquedas.

# Bibliografía

http://codecriticon.com/play-framework/