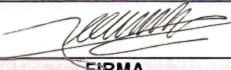
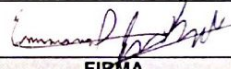



REALIZADO POR	FECHA	FIRMA
Facundo Salmerón		
REVISADO POR	FECHA	FIRMA
Emmanuel Rojas Fredini		
REVISADO POR	FECHA	FIRMA
Horacio Sagardoy		
APROBADO POR	FECHA	FIRMA
Lucila Romero		

**Nombre del Proyecto: DESARROLLO DE APLICACIÓN MÓVIL PARA EL DISEÑO Y PROCESAMIENTO DE ENCUESTAS PÚBLICAS SOBRE SALUD.**

**Periodo del Informe: 13/04/2018 – 31/05/2018**

**Alcance: Etapa 3**



Proyecto Final de Carrera Ingeniería Informática	Informe de estado del proyecto	FICH	UNL
---	--------------------------------	------	-----

Riesgos	Riesgo	Se efectivizó		Impacto	Mitigación
	R001: Falta de disponibilidad del director.	Si		Debido a cuestiones laborales del director del proyecto se produjo un leve retraso en el desarrollo del web Service y la corrección del informe de avance.	Continuar con tareas simultáneas hasta obtener las respuestas solicitadas.
	R002: Retrasos en los entregables.	Si		Debido que el ejecutor del proyecto tomó un empleo de 8 horas diarias y debido a situaciones de cursado en la carrera, el desarrollo de la etapa de diseño se vio postergado.	
	R004: Pérdidas en la base de datos obtenida.		No		
	R003: Indisponibilidad de los recursos.		No		
	R006: Cambios en los requerimientos.		No		
	Riesgos futuros			Probabilidad	Impacto
	Falta de disponibilidad del recurso humano del proyecto			Alta	Medio
Notas	<p><b>Como mitigación para solventar los retrasos producidos en el inicio del proyecto, se comenzó a trabajar más horas de las previstas por día y tiempo completo los fines de semana y feriados.</b></p> <p><b>Cabe aclarar que la actividad 3.4 fue agregada actualmente, teniendo en cuenta que la misma no había sido estimada en el ante-proyecto.</b></p>				



**UNIVERSIDAD NACIONAL DEL LITORAL**  
Facultad de Ingeniería y Ciencias Hídricas

PROYECTO FINAL DE CARRERA  
INGENIERÍA EN INFORMÁTICA

**Desarrollo de aplicación móvil para el  
diseño y procesamiento de encuestas  
públicas sobre salud.**

## **ETAPA 3: DESARROLLO DEL WEB SERVICE**

Alumno: Salmerón Facundo

Director: Rojas Fredini Emmanuel

Co-Director: Sagardoy Horacio

Santa Fe, Junio de 2018

## INFORME DE AVANCE N° 3:

En el siguiente anexo se detallan los resultados obtenidos de las tareas realizadas dentro de la tercera etapa que constituye el ciclo de vida del proyecto, denominada Desarrollo del Web Service.

### INSTALACIÓN/CONFIGURACIÓN DE HERRAMIENTAS DE TRABAJO:

Las herramientas que se utilizaron para el desarrollo del Web Service son las siguientes:

- IDE Java (Integrated Development Environment): Eclipse Oxygen.
- Gestor de proyectos y dependencias: Maven 3.5.0.
- Gestor de bases de datos: MySQL 5.7.
- Servidor: Apache Tomcat v9.0 Server.
- Frameworks:
  - Hibernate: 4.3.5.Final.
  - Spring: 4.0.3.Release.

Inicialmente se procedió a crear un **Maven Project** dentro del IDE Java Eclipse. Maven es una herramienta de software utilizada para la gestión y construcción de proyectos Java. El mismo tiene un modelo de configuración de construcción a través de un archivo en formato XML. Dicho archivo es el **POM (Project Object Model)** donde se describe el proyecto de software a construir, las dependencias de otros módulos y los componentes externos, como por ejemplo la utilización de librerías y frameworks. Mediante dichas anotaciones, se realizan descargas dinámicas de los repositorios permitiendo no tener que instalar manualmente cada dependencia a utilizar.

Por lo tanto, dentro del POM se incluyeron todas las dependencias necesarias para el desarrollo del Web Service, como por ejemplo para la utilización de Spring, Hibernate, JSON, MySQL Connector, etc. El formato de las dependencias es el siguiente:

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>4.3.5.Final</version>
</dependency>
```

Es pertinente realizar una breve descripción de los frameworks escogidos para el desarrollo. Por un lado, se tiene **Hibernate**. El mismo es una herramienta de **mapeo objeto-relacional (ORM)**, es decir logra mapear los atributos entre una base de datos relacional y el modelo de objetos de la aplicación. Estas relaciones se establecen mediante anotaciones en los beans de las entidades. Algunas de las anotaciones de relevancia que provee el framework son por ejemplo, el `@Table(name = tabla)` donde se especifica la tabla que representará la clase Java, `@GeneratedValue` la forma en que se generará el ID de la tabla en cuestión, `@Column(name = nombre)` mapeo de un objeto a una columna en base de datos, `@OneToOne`, `@OneToMany`, `@ManyToOne` y `@ManyToMany` para el mapeo de relaciones.

Por otro lado, se tiene el framework **Spring**. Dicha herramienta es de gran utilidad para

el desarrollo de aplicaciones web, permitiendo entre otras cosas la **inversión de control (ioc)**, es decir el cambio de flujo de ejecución y vida de los objetos con respecto a la programación tradicional, invirtiendo la forma en que se controla la aplicación, donde ahora es el framework quien se encarga de administrar los componentes (crearlos en orden correcto, conectarlos entre sí, configurarlos) sin depender del programador. Además, el framework provee la **inyección de dependencias (di)**, que es un tipo de inversión de control, donde en lugar de que cada clase tenga que instanciar los objetos que necesite, Spring se encargará de inyectar dichos objetos mediante los setters o el constructor en el momento que se cree la clase y cuando se la quiera utilizar la misma ya estará lista, a diferencia del método tradicional donde se debía instanciar cada vez que se quería utilizar una clase. Para ello, dentro de Spring hay un contenedor que se encarga de inyectar a cada objeto, los objetos que necesita según se le indique, mediante un archivo de configuración xml o anotaciones sobre los beans (*@Autowired*).

## DESARROLLO DE LA BASE DE DATOS:

Teniendo en cuenta el diagrama de base de datos diseñado en la etapa anterior, en esta ocasión se llevó a cabo la creación de la misma en el gestor MySQL. Para ello se creó una conexión de instancia local, manipulada mediante el puerto 3306 de la computadora personal. Luego se llevaron a cabo las distintas consultas SQL para dar de alta todas las tablas especificadas, junto a las características de cada columna. Se realizaron *inserts* manuales dentro del gestor, a fin de probar la integridad y respuesta de la base de datos.

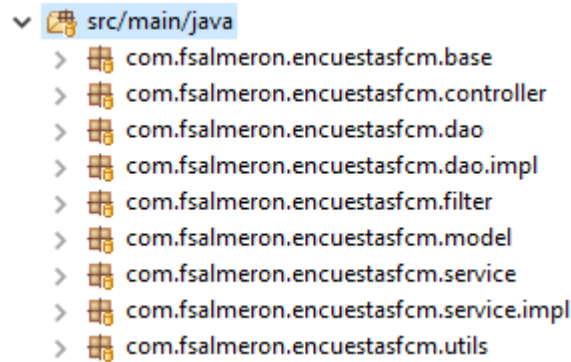
Finalmente, se realizó la configuración de la base de datos en el IDE Eclipse, a fin de sincronizarlo con el Web Service. Para ello, se llevaron a cabo las anotaciones pertinentes en formato xml dentro del archivo *servlet-context.xml*, provisto por dicho entorno, que a su vez posee todas aquellas configuraciones relacionadas al servidor y Hibernate.

## IMPLEMENTACIÓN DE LAS FUNCIONALIDADES:

Una vez realizadas todas las configuraciones en las herramientas de trabajo y con la base de datos corriendo de manera local, se llevó a cabo el desarrollo del Web Service, el cual posee una arquitectura *REST* y realiza envío de datos a la aplicación en formato *JSON*, según lo establecido en etapas anteriores.

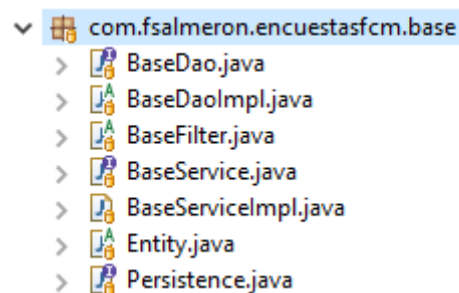
Al igual que la base de datos, el Web Service se encontrará corriendo en un entorno local, teniendo una URL de acceso al mismo mediante el puerto 8080 de la computadora personal, el cual permite aplicar el **protocolo de transferencia de hipertextos** (HTTP).

Con el fin de separar todas las clases de acuerdo a sus utilidades y siguiendo las buenas prácticas, se crearon distintos paquetes:



(Fig.1 – División de funcionalidades por paquetes)

Como se puede ver y de acuerdo al diagrama de Web Service establecido en la etapa anterior, se encuentra el paquete *controller* en relación a la capa de presentación, el *dao* y su implementación para la capa de datos, y luego el *Service* junto a su implementación, el *filter* y el *utils* relacionados a la capa de servicios.



(Fig.2 – Paquete base)

Dentro del paquete *base* se encuentran todas aquellas interfaces y clases genéricas que serán extendidas según correspondan. Cabe destacar que la clase *Persistence* representa una entidad del dominio que puede ser persistida, la cual es extendida por las demás “bases”. A modo de ejemplo, dentro del *BaseDao* se encuentran funciones genéricas como filtros, traer objetos por id, guardar o eliminar objetos en base de datos, etc.

Por su parte, en el paquete *model* se encuentran todas las clases java que representan el mapeo de objetos en relación a la base de datos, con sus respectivas anotaciones Hibernate. Las mismas extienden de la clase base *Entity*, la cual define genéricamente un id con sus respectivos *getters* y *setter*. Con respecto al id de cada clase, el mismo se generará automáticamente de forma secuencial cada vez que se cree un objeto en base de datos.

```
@Override
@Id
@Column(name = "IDUSUARIO", nullable = false, unique = true)
@GeneratedValue(strategy = GenerationType.AUTO, generator = "Usuario_Generator")
@SequenceGenerator(name = "Usuario_Generator", sequenceName = "Usuario_Generator")
public Integer getId() {
    return id;
}
```

(Fig.3 – Ejemplo de anotaciones Hibernate en relación al id de la clase “Usuario”)

Luego en el paquete *controller* se definieron los distintos controladores que se encargan de recibir y procesar las peticiones. Con la implementación del framework *Spring*, se establece un controlador frontal conocido como Dispatcher Servlet (Servlet

Despachador) cuya funcionalidad principal es la de encapsular y enrutar todos los controladores definidos posteriormente. El mismo es configurado en el archivo *web.xml*, que posee toda la información de configuración y despliegue para los componentes web que conforman la aplicación. En dicho archivo se estableció que la ruta será */*. Por lo tanto, todas aquellas peticiones dirigidas al entorno local (*localhost:8080/*), serán atendidas por este Dispatcher Servlet y dirigidas a los controladores que se hayan creado. A su vez, en la ruta de peticiones se tiene en cuenta el nombre del proyecto (*EncuestasFCM*) para la creación de la url.

Uno de los controladores creados es por ejemplo el “UserController” el cual recibe peticiones por parte de la aplicación para realizar la creación de un nuevo usuario o para loguear el mismo. Ésta clase controlador, está mapeada por una anotación **@RequestMapping(value = “/usuarios”)**, en la cual se coloca el nombre con el que se realizará el mapeo de las peticiones y será enrutado por el Dispatcher Servlet. A su vez, cada una de las funciones dentro de dicho controlador tendrán su propio @RequestMapping.

Para el caso de creación de nuevo usuario el mapeo de la solicitud se hace a través del String *“/saveUser”*, seguido de los parámetros a guardar en base de datos, como el nombre de usuario, password, etc. Entonces, la url de la petición será de la siguiente forma:

*http://localhost:8080/EncuestasFCM/usuarios/saveUser?nombre=...&password=...&fechaNacimiento=...&mail=...&activo=1&sexo=...&tipoUsuario=...*

Todos los parámetros que conforman la url serán obtenidos a través de la aplicación.

Cuando el Web Service recibe dicha petición, se ejecuta la función relacionada con este controlador. Dentro de ésta función, se utilizan unas clases de la capa de servicios, la cual se comunica con el objeto de acceso a datos (*DAO*), ya sea para realizar las validaciones y el posterior almacenamiento. Se utilizan los servicios de *“TipoUsuarioService”* para determinar el tipo de usuario a almacenar y luego *“UsuarioService”*, donde se validará por ejemplo, si el nombre de usuario o mail escogido ya se encuentra en uso, si la contraseña cumple con los requisitos pre-establecidos, etc. Para dichas validaciones, el DAO recupera la información de la tabla Usuarios y se construye un JSON para enviar la respuesta adecuada, incluyendo los errores cometidos (si es que los hay).

A continuación, se presenta un ejemplo de una respuesta en formato JSON, cuando se producen errores en la validación:

```
{
  "exito":false,
  "error":"El nombre de usuario escogido ya se encuentra en uso."
}
```

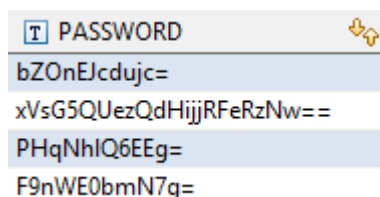
En el caso de superar las validaciones presentadas, dentro del servicio de usuarios se invocará a la función genérica del BaseService: *saveOrUpdate*, la cual se comunica con la capa de acceso a datos para persistir en base al nuevo usuario, retornando a la aplicación mediante un JSON:

```
{
  "exito":true
}
```



}

Cabe aclarar que dentro del paquete *utils*, se desarrolló una clase denominada **EncryptionUtils**. La misma contiene dos funciones para llevar a cabo la codificación y decodificación de un determinado String bajo el sistema **Base64**. Éste sistema numérico, utiliza 64 como base que es la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII, a la hora de encriptar códigos. Con el desarrollo de las mencionadas funciones, a la hora de crear un nuevo usuario, cuando se pasa la contraseña a almacenar por el parámetro de la petición, se procede a encriptar la misma como un mecanismo de seguridad a nivel base de datos.



PASSWORD
bZOnEJcdUjc=
xVsG5QUezQdHijjRFeRzNw==
PHqNhIQ6EEg=
F9nWE0bmN7g=

(Fig.4 – Forma de almacenamiento de las contraseñas en base de datos).

Para el caso de logueo de un usuario, el mapeo de la solicitud se hace a través del String *“/loginUser”*, seguido de los parámetros correspondientes como nombre de usuario y contraseña. La url de la petición se determina de la siguiente forma:

*http://localhost:8080/EncuestasFCM/loginUser?nombre=...&password=...*

Cuando el Web Service recibe dicha petición, mediante los servicios de Usuario, se encarga de verificar que los datos ingresados sean correctos, devolviendo a la aplicación un JSON informando el éxito o no de la transacción.

Ejemplo JSON logueo exitoso:

```
{
  "exito":true,
  "tipoUsuario":1,
  "id":2,
  "sexo":1,
  "nombre":"facusalmeron"
}
```

Logueo con errores:

```
{
  "exito":false,
  "error":"La contraseña ingresada es incorrecta"
}
```

De manera similar, se procede con los demás controladores implementados en el Web Service.

Para el caso de “EncuestasController” se poseen las siguientes funciones con sus determinadas url de peticiones:

- *http://localhost:8080/EncuestasFCM/encuestas/getAll* → retorna una lista de todas las encuestas que se encuentren activas dentro de la tabla *Encuestas*, en

formato JSON, para presentarlas en la sección correspondiente dentro de la aplicación.

Ejemplo:

```
{
  "response": [
    {
      "descripcion": "Descripción encuesta número 1",
      "titulo": "Primer encuesta",
      "id": 1
    },
    {
      "descripcion": "Descripción encuesta número 2",
      "titulo": "Segunda encuesta",
      "id": 2
    }
  ]
}
```

- <http://localhost:8080/EncuestasFCM/encuestas/openEncuesta?idEncuesta=...> → retorna todos los elementos que componen a la encuesta pasada por id, en formato JSON, utilizado para abrir una determinada encuesta en la aplicación.

Ejemplo:

```
{
  "descripcion": "Descripción encuesta número 1",
  "titulo": "Primer encuesta",
  "preguntas": [
    {
      "descripcionPregunta": "Pregunta 1 ",
      "respuesta": [
        {
          "descripcionRespuesta": "Respuesta 1.1",
          "idTipoRespuesta": 2,
          "idRespuesta": 1
        },
        {
          "descripcionRespuesta": "Respuesta 1.2",
          "idTipoRespuesta": 2,
          "idRespuesta": 2
        }
      ],
      "idPregunta": 1
    },
    {
      "descripcionPregunta": "Pregunta 2",
      "respuesta": [
        {
          "descripcionRespuesta": "Respuesta 2.1",
          "idTipoRespuesta": 2,
          "idRespuesta": 3
        },
        {
          "descripcionRespuesta": "Respuesta 2.2",
          "idTipoRespuesta": 2,
          "idRespuesta": 4
        }
      ],
      "idPregunta": 2
    }
  ]
}
```

```

    }
  ],
  "id":1
}

```

- <http://localhost:8080/EncuestasFCM/encuestas/saveEncuesta?titulo=...&descripcion=...&idUsuario=...> → función utilizada para dar de alta una nueva encuesta dentro de la aplicación.
- <http://localhost:8080/EncuestasFCM/encuestas/updateEncuesta?idEncuesta=...&titulo=...&descripcion=...&idUsuario=...> → función utilizada para realizar modificaciones en la encuesta cuyo id es pasado como parámetro.
- <http://localhost:8080/EncuestasFCM/encuestas/disableEncuesta?idEncuesta=...&idUsuario=...> → función utilizada para inhabilitar una encuesta (activo = false), cuyo id es pasado por parámetro.
- <http://localhost:8080/EncuestasFCM/encuestas/enableEncuesta?idEncuesta=...&idUsuario=...> → función utilizada para habilitar una encuesta (activo = true), cuyo id es pasado por parámetro.
- <http://localhost:8080/EncuestasFCM/encuestas/deleteEncuesta?idEncuesta=...> → función utilizada para dar de baja en base de datos una encuesta, junto a sus respectivas preguntas y respuestas asociadas.

En el caso de “PreguntaController”, se poseen las siguientes funciones para realizar los ABMs correspondientes:

- <http://localhost:8080/EncuestasFCM/preguntas/savePregunta?descripcion=...&idEncuesta=...&idUsuario=...> → con ésta función se da de alta una nueva pregunta para la encuesta cuyo id se mapea en el parámetro de entrada.
- <http://localhost:8080/EncuestasFCM/preguntas/updatePregunta?idPregunta=...&descripcion=...&idUsuario=...> → función utilizada para realizar modificaciones en una determinada pregunta cuyo id es pasado como parámetro.
- <http://localhost:8080/EncuestasFCM/preguntas/deletePregunta?idPregunta=...&idUsuario=...> → función utilizada para dar de baja una determinada pregunta cuyo id es pasado como parámetro.

Luego para el caso de “RespuestaController”, se poseen las siguientes funciones orientadas a realizar los ABMs correspondientes:

- <http://localhost:8080/EncuestasFCM/respuestas/saveRespuesta?descripcion=...&idPregunta=...&idTipoRespuesta=...&idUsuario=...> → función utilizada para dar de alta una respuesta, asociada a una determinada pregunta cuyo id es pasado como parámetro.
- <http://localhost:8080/EncuestasFCM/respuestas/updateRespuesta?idRespuesta=...&descripcion=...&idUsuario=...> → función utilizada para realizar modificaciones

sobre una respuesta cuyo id es pasado como parámetro.

- <http://localhost:8080/EncuestasFCM/respuestas/deleteRespuesta?idRespuesta=...&idUsuario=...> → función utilizada para dar de baja una determinada respuesta cuyo id es pasado como parámetro.

Como se puede observar, en todas las funciones relacionadas a los ABMs, habilitación e inhabilitación, se pasa como parámetro el id del usuario. Esto se debe a que todos los cambios que se vayan registrando en base de datos se almacenará su responsable, a modo de brindar un mayor control.

Finalmente, para el controlador “ResultadoController” se posee la siguiente función, cuyo url de la petición se presenta a continuación:

- <http://localhost:8080/EncuestasFCM/resultados/saveResultado?latitud=...&longitud=...&edadEncuestado=...&sexoEncuestado=...&idUsuario=...&idRespuesta=...> → función utilizada para almacenar el resultado al responder una determinada encuesta en la aplicación.