

Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Reporte Final del Proyecto de Integración

Aplicación Web para incidentes basada en la opinión de la comunidad
de la UAM-Azcapotzalco

Modalidad: Proyecto Tecnológico

Trimestre 2018 Invierno

Rodrigo Lino Osorio
2123001401
royer_lino@hotmail.com

José Alejandro Reyes Ortiz
Doctor en Ciencias de la Computación
Profesor Asociado
Departamento de Sistemas
jaro@correo.azc.uam.mx

10 de abril de 2018

Declaratoria

Yo, Dr. José Alejandro Reyes Ortiz, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



José Alejandro Reyes Ortiz

Yo, Rodrigo Lino Osorio, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Rodrigo Lino Osorio

Resumen

Como miembros de la comunidad de la U.A.M. Azcapotzalco, es importante considerar el hecho de que en la universidad suelen suceder un sinnúmero de acontecimientos, donde solo algún porcentaje de la comunidad puede estar involucrada y por ende solo esas personas se enteran de ello. Algunos de los acontecimientos considerados para este proyecto son robos, bloqueos, agresiones, incendios y fugas. Siguiendo con el contexto, en sí, no hay forma o no es tan considerable el estar informando a toda la comunidad sobre cada instante en el que acontece un nuevo incidente. No obstante, mediante las redes sociales hay ocasiones en la que los mismos alumnos informan o alertan a ciertos miembros de la universidad sobre cierto incidente acaecido para así estar prevenidos, pero la mayoría no se percatan de estas publicaciones.

Por este motivo la aplicación web tiene la finalidad de informar a cualquier usuario que desee saber sobre la dimensión de los incidentes ocurridos. Y no sólo eso, si ha pasado por alguno de los acontecimientos mencionados, si lo considera necesario, puede dar de alta un reporte especificando la ubicación, el tipo de incidente, una breve descripción y su nombre, aunque este último es opcional. Si no fuera el caso, el usuario puede emitir opiniones sobre los incidentes ya registrados, colocando un comentario, una clasificación para determinar la relevancia (mediante la escala en estrellas) y su nombre si así lo desea, siempre y cuando haya presenciado o sido testigo de algún incidente dentro de la escuela. Este sistema es de gran ayuda ya que se así se cuenta con la participación de la comunidad en general y con ello tener información sobre los puntos donde suelen ocurrir incidentes con mayor frecuencia.

La aplicación web, está disponible y alojada en la siguiente liga <https://aisii.azc.uam.mx:8080/IncidentesUAM>, cualquier miembro de la universidad puede acceder a esta página ya sea desde su celular, tablet, laptop o máquina de escritorio con acceso a Internet.

Índice

1. Introducción	1
2. Antecedentes	2
2.1 Proyectos Terminales	2
2.2 Tesis	3
2.3 Software.....	3
3. Justificación	3
4. Objetivos	4
4.1 Objetivo General	4
4.2 Objetivos Específicos.....	4
5. Marco teórico	5
5.1 <i>Google Maps JavaScript API</i>	5
Paso 1. Crear una página <i>HTML</i>	5
Paso 2. Agregar un marcador en el mapa.....	6
Paso 3. Obtener una clave <i>API</i>	7
5.2 <i>Framework Hibernate</i>	10
5.3 <i>Framework Spring MVC</i>	10
5.4 <i>JSP y Servlets</i>	11
5.5 <i>Bootstrap</i>	12
5.6 <i>CSS y HTML5</i>	13
5.7 Incidentes relacionados a la seguridad.....	14
6. Desarrollo del proyecto	15
6.1 Base de datos	15
6.2 Mapeo de entidades	16
6.3 Conexión a la base de datos con <i>Hibernate</i>	17
6.4 Configurar archivo <i>web.xml</i>	17
6.5 Módulo para el registro de incidentes	19
6.5.1 Diseño del módulo registrar	19
6.5.2 Implementación del módulo registrar	22
6.6 Módulo para la consulta de incidentes.....	22
6.6.1 Diseño del módulo consultar (incidente, espacio y fecha).....	23
6.6.2 Implementación del módulo consultar (incidente, espacio y fecha)	26
6.7 Módulo para la opinión de incidentes	27
6.7.1 Diseño del módulo opiniones	27

6.7.2 Implementación del módulo opiniones	30
6.8 Interfaz gráfica de usuario	32
7. Resultados	38
7.1 Resultados de registro	38
7.2 Resultados de consulta.....	38
7.3 Resultados de opiniones.....	41
8. Análisis y Discusión de resultados	43
9. Conclusiones	43
10. Referencias	44
11. Anexos	45
11.1 Script para la base de datos e inserción de registros	45
11.2 Clase <i>java</i> para crear conexión a la base de datos	48
11.3 Clase reporte.java.....	48
11.4 Mapeo clase reporte	50
11.5 Formulario para registrar reporte (simplificado)	50
11.6 Controlador registrar reporte.....	51
11.7 Insertar reporte	52
11.8 Clase ReporteIncidenteEspacio.java	52
11.9 Controlador consultar incidentes	54
11.10 Consultar reporte	56
11.11 Clase opinion.java	60
11.12 Mapeo clase opinion	61
11.13 Formulario para agregar opinión (simplificado).....	62
11.14 Controlador agregar/mostrar opinión	64
11.15 Insertar opinión	65
11.16 Obtener opiniones	65
11.17 Clase vistasReferencias.java	66
11.18 Mapeo clase incidente	67
11.19 Mapeo clase espacio físico.....	67
11.20 Clase incidente.java.....	67
11.21 Clase espaciofisico.java.....	68

Índice de figuras.

Figura 1. Ejemplo básico <i>HTML5</i>	5
Figura 2. Ejemplo para agregar un marcador en el mapa de <i>Google Maps</i>	6
Figura 3. Aceptar términos y condiciones de la <i>API Google Maps</i>	8
Figura 4. Habilitar la <i>API Google Maps</i>	8
Figura 5. Tipo de credencial a utilizar en la <i>API</i>	9
Figura 6. Clave de <i>API</i> obtenida.....	9
Figura 7. Funcionamiento básico del <i>DispatcherServlet</i>	11
Figura 8. Modelo de funcionamiento, <i>Servlet</i> y <i>JSP</i>	12
Figura 9. Modelo Entidad-Relación de la base de datos.....	15
Figura 10. Archivo de mapeo " <i>Reporte.hbm.xml</i> ".....	16
Figura 11. Archivo de configuración <i>hibernate.cfg.xml</i>	17
Figura 12. Configuración de <i>web.xml</i>	18
Figura 13. Configuración de <i>controlador-servlet.xml</i>	19
Figura 14. Diagrama de clases para registrar un reporte.....	20
Figura 15. Formulario para registrar reporte.....	21
Figura 16. Vista para el registro de incidente.....	22
Figura 17. Diagrama de clases para realizar consultas.....	25
Figura 18. Vista para consultar por incidente.....	26
Figura 19. Vista para consultar por espacio físico.....	26
Figura 20. Vista para consultar por fecha.....	27
Figura 21. Diagrama de clases para agregar y mostrar opiniones.....	29
Figura 22. Diagrama de clase para navegar entre páginas <i>JSP</i>	29
Figura 23. Vista para ver el total de reportes registrados.....	30
Figura 24. Vista para agregar una opinión.....	30
Figura 25. Vista para mostrar opiniones de algún incidente.....	31
Figura 26. Vista de opiniones agregadas en un incidente.....	31
Figura 27. Vista versión móvil para el registro del incidente.....	32
Figura 28. Vista versión móvil de la página de inicio.....	33
Figura 29. Vista versión escritorio para el registro de incidente.....	33
Figura 30. Vista versión móvil para la consulta por incidente.....	34
Figura 31. Vista versión móvil para la consulta por espacio físico.....	34
Figura 32. Vista versión móvil para la consulta por fecha.....	35
Figura 33. Vista versión móvil de los reportes registrados. y Figura 34. Vista versión móvil para agregar una opinión.....	35
Figura 35. Vista versión móvil para ver opiniones agregadas.....	36
Figura 36. Vista versión escritorio de la página acerca del proyecto.....	36
Figura 37. Vista versión escritorio de la página proyecto tecnológico.....	37
Figura 38. Vista versión escritorio de la página contacto.....	37
Figura 39. Porcentaje de los reportes de prueba registrados.....	38
Figura 40. Consulta sobre el mapa del incidente Bloqueo.....	39
Figura 41. Consulta sobre la tabla del incidente Bloqueo.....	39
Figura 42. Consulta sobre el mapa del espacio físico Edificio G.....	40
Figura 43. Consulta sobre la tabla del espacio físico Edificio G.....	40
Figura 44. Consulta sobre el mapa de la fecha 28/02/2018.....	41
Figura 45. Consulta sobre la tabla de la fecha 28/02/2018.....	41
Figura 46. Mapa con el <i>total de reportes de prueba registrados</i>	42
Figura 47. Opiniones mostradas en los reportes 1, 27, 38 y 42.....	42
Tabla 1. Precisión de los módulos.....	43

1. Introducción

La universidad es sin duda uno de los lugares donde gran parte de la comunidad universitaria pasa la mayoría de su tiempo, en la cual no se sabe a detalle que pasa alrededor de ella. Hoy por hoy, el tema de la inseguridad se ha vuelto una de las principales preocupaciones en nuestro país, con base en la encuesta elaborada por el INEGI en el año 2017, se tiene que la Ciudad de México es uno de los lugares donde hay mayor incertidumbre, debido a la alta incidencia delictiva que existe en la actualidad [1]. El incidente que ocurre con mayor frecuencia es el robo y este suele suceder en el trayecto a la escuela, al trabajo o al hogar, pero no se puede determinar el momento exacto cuándo ocurre esta situación y el riesgo que puede generar a la sociedad.

Un caso similar, sucede en la UAM Azcapotzalco, donde algunos miembros de la universidad han pasado o vivido diversos incidentes. Aunque, ellos no contaban con un sistema o una aplicación para registrar el suceso o consultar dicha información. Es por eso que es de gran utilidad el saber los tipos de incidentes que se relacione con la seguridad de los miembros, como: robos, agresiones, bloqueos o situaciones que generen peligro como incendios, fugas de gas u otro químico. Tampoco se enteran del lugar en específico donde se llevó a cabo el suceso, al no tener datos seguros que informen a la comunidad sobre estos hechos, se desconoce la magnitud de los incidentes y de cierta forma, está afectando la integridad de la universidad.

Por lo tanto, para este proyecto se elaboró una aplicación web que nos muestra algo que asemeja la magnitud de los incidentes que puedan ocurrir en un futuro y así tener detalles de lo acontecido. Esta aplicación permitió dar de alta, consultar y opinar sobre dichos incidentes. La plataforma utilizada para elaborar esta aplicación web fue *Java EE*, con el entorno de desarrollo Eclipse y me basé en el patrón de arquitectura de software *MVC*, ya que se fundamenta de tres componentes (Modelo - Vista - Controlador), por lo que se aparta la lógica de la aplicación con la presentación hacia el usuario.

El modelo entidad/relación de los datos se llevó a cabo con el manejador de base de datos *MySQL*. Para la persistencia de datos y correspondencia entre las tablas y las clases se utilizó el *framework Hibernate*.

La vista de la aplicación se visualizó con ayuda de páginas *JSP*, además de permitir intercalar código *HTML* con código *Java*, para dar formato se usó hojas de estilo *CSS*. Con ayuda del *framework Bootstrap* se dieron estilos a los formularios, barra de navegación, tablas e iconos. Para llevar cierto control de los datos ingresados en los formularios y considerar las ubicaciones geográficas de la universidad se utilizó *JavaScript*, con ayuda de este lenguaje se inicializó la *API* de *Google Maps*.

Los controladores se implementaron con el *framework Spring MVC*, con este *framework* se cargan los modelos de datos y se invocan las vistas de cada módulo.

2. Antecedentes

2.1 Proyectos Terminales

1. Sistema Web para gestionar incidentes delictivos [2].

En este proyecto, la función principal consiste en gestionar incidentes delictivos de la Ciudad de México con el propósito de informar a la sociedad sobre lo que ocurre día a día y así aportar una mejor seguridad a los ciudadanos. El parecido que tiene el proyecto [2] con este proyecto es el registro de incidentes por parte de los usuarios, pero sin incluir incidentes graves como violación u homicidio. La diferencia se centra en que [2] permite publicar, eliminar, modificar o buscar incidentes de la Ciudad de México, mientras en este proyecto se permite registrar o visualizar incidentes sobre el mapa de la UAM Azcapotzalco para informar a los miembros de la comunidad. Además, se puede añadir una opinión o una valoración del incidente existente por parte de la comunidad.

2. Sistema de información para el análisis de la seguridad social o pública a través de periódicos [3].

La función principal de este proyecto es el análisis de notas periodísticas acerca de la seguridad en la sociedad y que están almacenados en la web. Después, se generan archivos de texto y se examinan haciendo uso de la minería de datos, para así clasificarlos y mostrarlos sobre una interfaz gráfica. La relación que tiene el proyecto [3] con este proyecto, es la consideración de algunos incidentes como robos o agresiones, de la misma forma se mostrarán los incidentes obtenidos sobre una interfaz gráfica. La diferencia de [3] con este proyecto radica en la visualización de incidentes, los cuales serán mostrados sobre un mapa geográfico de la universidad UAM Azcapotzalco. También, otra diferencia es la extracción de incidentes a partir de sitios web, mientras, en este proyecto de integración se extraerán los datos de los registros que darán de alta los miembros de la comunidad. Además, se cuenta con opiniones y valoraciones de los usuarios sobre los registros ya generados.

3. Sistema Web para la geolocalización de incidentes delictivos a partir de publicaciones en *Twitter* [4].

El objetivo de [4] es la visualización de delitos, el cual extrae *tweets* de cuentas oficiales de *Twitter* de ciertos periódicos confiables de la Ciudad de México, dando a conocer los lugares de ocurrencia a través de la geolocalización. La semejanza que tiene [4] con este proyecto, es el hecho de dar a conocer la ubicación de los incidentes sobre un mapa geográfico usando la API¹ de *Google Maps*. La diferencia que tiene con este proyecto, se basa en que [4] no lleva a cabo el registro de incidentes desde el mapa geográfico. Esto se debe a que solo se visualizan los incidentes generados desde los *tweets*, pero en este proyecto existe la manera de registrar incidentes desde el mismo mapa y en la visualización del incidente existente, el usuario puede agregar su opinión.

¹ API (*Application Programming Interface*), es un conjunto de subrutinas, métodos y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

2.2 Tesis

4. Red social para la detección de zonas de riesgo y participación ciudadana para dispositivos móviles y computadoras personales [5].

En esta tesis, el objetivo principal es el desarrollo de una red social para dar a conocer cuáles son los sitios de alto riesgo en la Ciudad de México a través de un mapa digital tanto de manera web y móvil. La relación que tiene esta tesis con este proyecto, es el uso de aplicaciones web y en ambas se visualizan sobre un mapa, aquellos sitios donde han acaecido incidentes (robos) y que fueron registrados por algún usuario. La diferencia se centra en el uso de *Geoserver*², porque contemplaron a la Ciudad de México, mientras que, en este proyecto solo se abarcó a la UAM Azcapotzalco, para así informar a los miembros de la comunidad. Para considerar al mapa geográfico de la universidad, se implementó con la ayuda de la *API* de *Google Maps*.

2.3 Software

5. Semáforo Delictivo [6].

Es una aplicación móvil que permite enviar alertas de manera anónima en tiempo real, sobre incidentes delictivos que ocurren en cualquier parte de México, entre ellos robos, homicidios, secuestros, extorción, lesiones y violación. Esta aplicación comparte la idea de informar a la sociedad los sitios en el cual ocurren ciertos tipos de incidentes para así prevenir o evitar la zona. La diferencia que existe entre la aplicación [6] y este proyecto, es el hecho de informar a todos los estados de la república mexicana, además, por medio de encuestas obtiene estadísticos sobre los incidentes que ocurren en cada estado.

6. CincoD – Vecino Vigilante [7].

Es una aplicación móvil, que sirve para reducir el crimen a través de la participación ciudadana, fue creada para las personas que frecuentan o viven en la delegación Benito Juárez. Si estas personas son testigos de algún delito, incidente o situación peligrosa, podrán tomar un video y enviarlo de manera anónima al Centro de Comando y Control de la delegación, de esta forma enviarán a un elemento de seguridad para brindar apoyo. La similitud se centra en el hecho que los usuarios podrán registrar reportes de ciertos incidentes, pero la diferencia está en el registro, el cual incluye un video del incidente, para apoyar a las autoridades en apoyo a la víctima. Además, solo es para aquellas personas situadas en la delegación mencionada.

3. Justificación

De los incidentes que ocurren diariamente en la UAM Azcapotzalco, solo algunas personas se enteran de lo sucedido, esto se debe a que se encontraban en el instante o el lugar del incidente. En particular, de aquellos incidentes que puedan

² *Geoserver* es un servidor web que permite servir mapas y datos de diferentes formatos para aplicaciones web.

sucedir dentro de la universidad como robo a vehículos, robo a transeúnte, robo en aulas o en instalaciones de la unidad, bloqueos o agresiones por parte de los miembros de la comunidad. También, existen situaciones que puedan generar peligro como incendios, fugas de gas u otro químico en aulas, laboratorios o pasillos de los edificios.

Por el bienestar y la seguridad de la comunidad es importante saber cuáles son los espacios de la unidad donde suelen acaecer los incidentes, ya que jamás se tiene contemplado el hecho de que ocurra durante su estancia en la UAM. En varias ocasiones se ignora o no se le da importancia cuando alguien pasa por esta situación, causando que la gran mayoría de las personas no se percata de lo acontecido. A lo largo de los años y con la aparición de redes sociales, se ha notado como algunos estudiantes han sido víctimas de algún robo o daño en sus vehículos, cuando se encontraban estacionados en la unidad. Las personas que cometen este acto, averiguan cuáles son los puntos ciegos de las cámaras de seguridad de la universidad, para así no ser captados.

Por lo tanto, la aplicación web sirve de ayuda para localizar los diferentes sitios de la UAM Azcapotzalco donde suelen ser más frecuentes los tipos de incidentes ya mencionados, así se puede prevenir a todo aquel personal de la unidad para que no pase por una situación similar. Otro de los beneficios que ofrece este proyecto es que permite a los usuarios agregar una opinión sobre los reportes que se dieron de alta, con esto la aplicación es colaborativa y se sabe las opiniones de la comunidad al respecto. Con esto, la institución y miembros de la comunidad, pueden ser favorecidos al contar con dicha aplicación, además, la vigilancia en dichas zonas puede ser fortalecida.

4. Objetivos

4.1 Objetivo General

Diseñar e implementar una aplicación web para registrar y mostrar incidentes de la UAM Azcapotzalco, considerando las opiniones de la comunidad universitaria sobre los incidentes con la finalidad de informar sobre lo que sucede en la unidad.

4.2 Objetivos Específicos

- Diseñar e implementar un módulo para registrar incidentes en la UAM Azcapotzalco indicando su espacio físico, es decir, el lugar donde sucedió el incidente.
- Diseñar e implementar un módulo para visualizar incidentes registrados, utilizando el mapa geográfico de la UAM Azcapotzalco.
- Diseñar e implementar un módulo para registrar opiniones y valoraciones de los usuarios sobre los incidentes, con el propósito de hacer una aplicación colaborativa.

5. Marco teórico

5.1 Google Maps JavaScript API

Esta API de *Google Maps* permite visualizar y publicar nuestros mapas en la web, está basado en el lenguaje de programación *JavaScript* y está orientados a objetos. A continuación se indican cuáles son los pasos que se necesitan para utilizar esta API en cualquier página web [8].

Paso 1. Crear una página *HTML*.

Se debe crear una página web *HTML5* básica, ver Figura 1.

Este es el código *HTML5*:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      #mapa {
        width: 100%;
        height: 400px;
        background-color: grey;
      }
    </style>
  </head>
  <body>
    <h3>Demo Google Maps</h3>
    <div id="mapa"></div>
  </body>
</html>
```

Figura 1. Ejemplo básico *HTML5*.

A continuación, se explica el código anterior.

Se crea una página *HTML* que contiene un encabezado `<head></head>` y un cuerpo `<body></body>`.

La etiqueta *style* define el tamaño de div del mapa. En este caso, div tiene 400 píxeles de alto y 100% de ancho, para mostrarlo a lo ancho de la página web. El fondo es de color gris *background-color: grey*.

Se agrega un nivel de encabezado tres (*h3*) arriba del mapa.

Se define un área de la página para el mapa de Google.

La etiqueta *div* aparece como un bloque gris porque aún no se agrega un mapa.

Paso 2. Agregar un marcador en el mapa.

Para agregar un marcador se agrega una función `initMap` dentro de un `script`, esta función se inicializa en otro `script`, una vez que se cargue la página. El código completo queda de la siguiente manera, ver Figura 2.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      #mapa {
        height: 400px;
        width: 100%;
      }
    </style>
  </head>
  <body>
    <h3>Demo Google Maps</h3>
    <div id="mapa"></div>
    <script>
      function initMap() {
        var coord = {lat: -25.363, lng: 131.044};
        var mapa = new google.maps.Map(document.getElementById('mapa'), {
          zoom: 4,
          center: coord
        });
        var marcador = new google.maps.Marker({
          position: coord,
          map: mapa
        });
      }
    </script>
    <script async defer
      src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callba
ck=initMap">
    </script>
  </body>
</html>
```

Figura 2. Ejemplo para agregar un marcador en el mapa de *Google Maps*.

El código anterior ya no contiene el CSS que mostraba un `div` de color gris. Esto se debe a que `div` ahora contiene el mapa.

A continuación, se explica cada sección del código anterior.

```
<script async defer
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback
=initMap"
</script>
```

La etiqueta `script` carga la *API* desde la *URL* especificada.

El parámetro `callback` ejecuta la función `initMap` cuando la *API* se carga por completo.

El atributo `async` permite al navegador seguir procesando el resto de la página mientras se carga la *API*.

El parámetro *key* contiene la clave *API* (se obtiene en el siguiente paso).

```
<script>
  function initMap() { }
</script>
```

La función `initMap` inicializa y agrega el mapa cuando se carga la página web.

```
getElementById
```

Se agrega esta función para buscar el id del div del mapa en la página web.

```
new google.maps.Map()
```

Se agrega este nuevo objeto de mapas de Google para crear un mapa en el elemento `div`.

```
{ var coord = {lat: -25.363, lng: 131.044};
  var mapa = new google.maps.Map(document.getElementById('mapa'), {
    zoom: 4,
    center: coord
  }); }
```

Se agregan propiedades al mapa, como el centro y el nivel de zoom. La propiedad `center` comunica a la *API* el lugar en el que debe centrar el mapa. Las coordenadas del mapa se definen en este orden: latitud, longitud.

La propiedad `zoom` especifica el nivel de zoom del mapa. `Zoom: 0` es el valor de zoom más bajo y muestra todo el planeta.

```
var marcador = new google.maps.Marker({
  position: coord,
  map: mapa
});
```

Esta parte del código agrega un marcador al mapa. La propiedad `position` define la posición del marcador, sería la coordenada asignada en la variable `coord`.

Paso 3. Obtener una clave *API*

Se requiere de una clave *API*, que es la llave que permite inicializar la función *JavaScript* donde está alojado el mapa. Los pasos para obtener la clave son los siguientes:

- Acceder a 'Google API Console' con nuestra cuenta de Google:

https://console.developers.google.com/flows/enableapi?apiid=maps_backend,geocoding_backend,directions_backend,distance_matrix_backend,elevation_backend,places_backend&reusekey=true

- Ya iniciada la sesión, aparece una ventana como la Figura 3, se debe seleccionar ‘Sí’ en la segunda opción, con esto se aceptaran los términos y condiciones del uso de la API y después click en el botón ‘Aceptar y continuar’

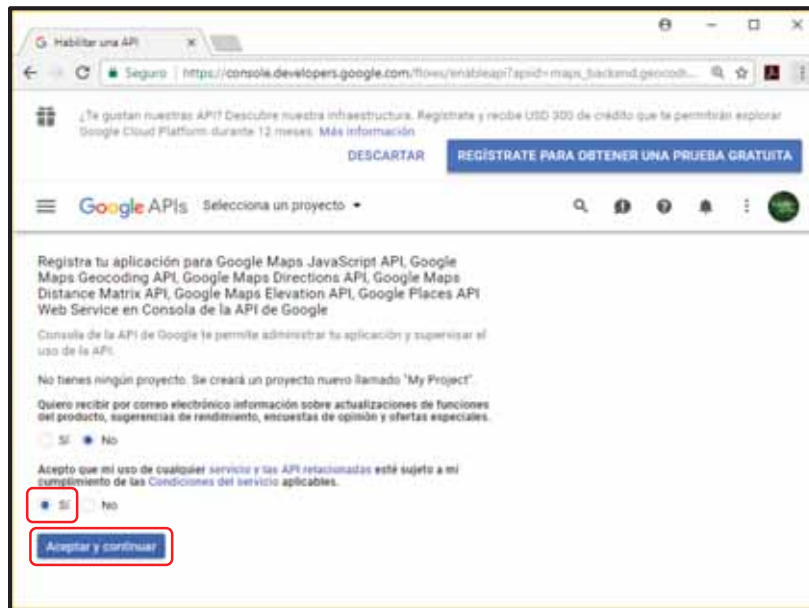


Figura 3. Aceptar términos y condiciones de la API Google Maps.

- Después, se muestra una ventana como la Figura 4, se selecciona ‘Crear un proyecto’ y después click en ‘Continuar’, para así habilitar la API y cualquier servicio relacionado.

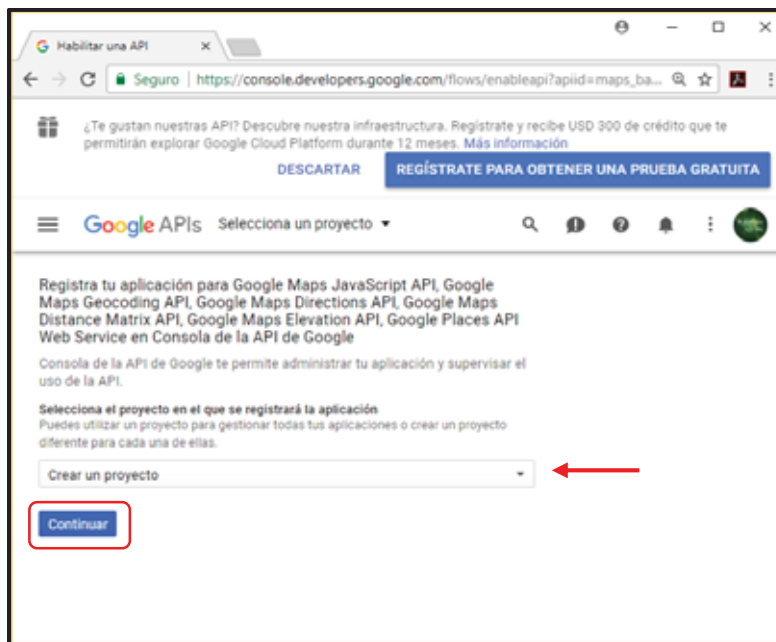


Figura 4. Habilitar la API Google Maps.

- Posteriormente, se muestra otra ventana como la Figura 5, aquí aparecen las credenciales que se pueden ocupar, en nuestro caso, se deja la opción que está 'Google Maps JavaScript API' y dar click en el botón '¿Qué credenciales necesito?'.

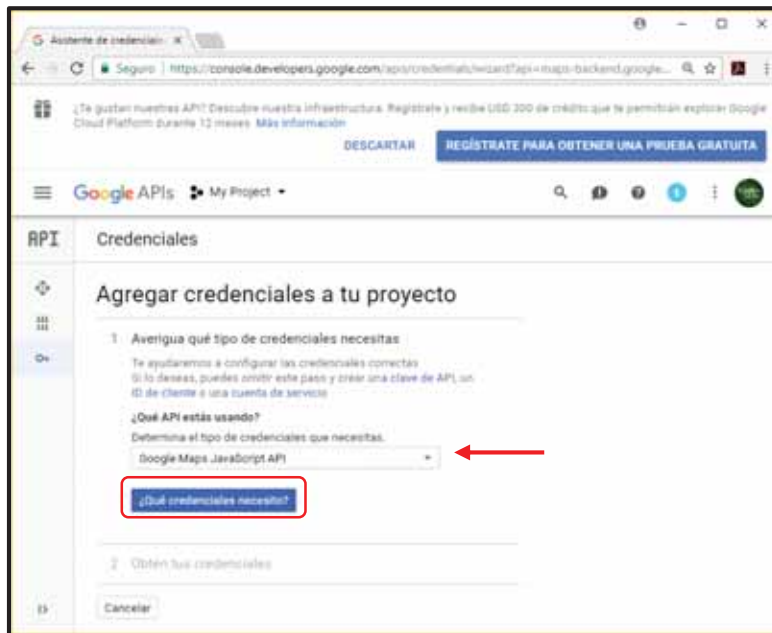


Figura 5. Tipo de credencial a utilizar en la API.

- En automático se genera nuestra clave API, ver Figura 6, copiamos y guardamos esta llave para utilizarla más adelante. Si se desea restringir la clave puede hacerse, para así evitar el uso no autorizado o el robo de cuotas. En nuestro caso no es necesario porque no contrataremos algún plan *Premium*.



Figura 6. Clave de API obtenida.

- La clave obtenida se reemplaza por el valor del parámetro *key* de la *URL* que está en el atributo 'src' del siguiente *script*.

```
<script  
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap"  
async defer>  
</script>
```

Al seguir estos pasos, nuestra página web ya tiene agregado un mapa de *Google*.

5.2 Framework Hibernate

Hibernate es una herramienta de Mapeo objeto-relacional (*ORM*) para la plataforma *Java* que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (*XML*) o anotaciones en los *beans* de las entidades que permiten establecer estas relaciones [9].

En otras palabras, *Hibernate* es un *Framework* que agiliza la relación entre la aplicación y la base de datos. Para poder aprender a utilizarlo es necesario contar con los conocimientos básicos de base de datos y *SQL* así como manejar el lenguaje *Java*.

¿Cómo funciona?

El desarrollador deberá configurar en un archivo *XML* o mediante *annotations* donde corresponde un atributo de una clase, con una columna de una tabla. Es una tarea simple donde existen herramientas que lo hacen por nosotros

Características

Simplicidad y flexibilidad: necesita un único archivo de configuración en tiempo de ejecución y un documento de mapeo para cada aplicación. Este archivo puede ser el estándar de *Java* o un archivo *XML*. También se tiene la alternativa de realizar la configuración de forma programática. El uso de *frameworks* de persistencia, tales como *EJBs* hace que la aplicación dependa del *framework*. *Hibernate* no crea esa dependencia adicional. Los objetos persistentes en la aplicación no tienen que heredar de una clase de *Hibernate* u obedecer a una semántica específica. Tampoco necesita un contenedor para funcionar.

5.3 Framework Spring MVC

Spring Web MVC es un sub-proyecto *Spring* que está dirigido a facilitar y optimizar el proceso creación de aplicaciones web utilizando el patrón *MVC* (Modelo-Vista-Controlador), donde el Modelo representa los datos o información que manejará la aplicación web, la Vista son todos los elementos de la *UI* (Interfaz de Usuario), con ellos el usuario interactúa con la aplicación, ejemplo: botones, campos de texto, etc., finalmente el Controlador será el encargado manipular los datos en base a la interacción del usuario [10].

La pieza central de *Framework Spring MVC* es el *DispatcherServlet*, ver Figura 7, que extiende la clase *HttpServlet* este componente es el encargado de recibir las peticiones *HTTP* y generar la respuesta adecuada a dicha petición, tradicionalmente se configura utilizando el archivo *web.xml* aunque utilizando la especificación *Servlet* 3.0+ es posible configurar programáticamente, su funcionamiento básico se muestra en el siguiente diagrama:

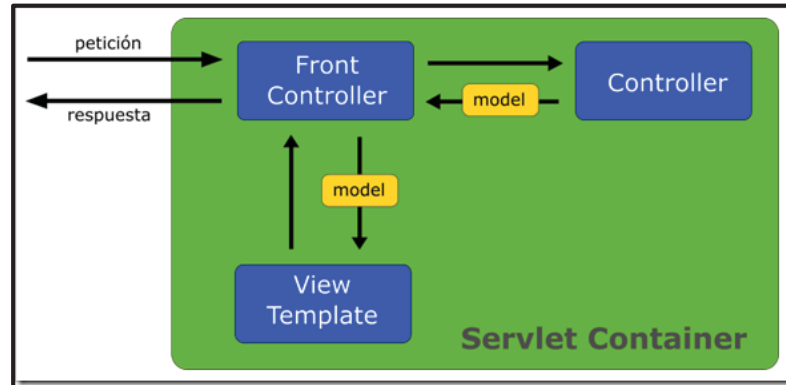


Figura 7. Funcionamiento básico del *DispatcherServlet*.

1. El proceso inicia cuando el usuario envía una petición, normalmente abre el navegador y escribe la *URL* que identifica nuestra aplicación, ejemplo: <http://localhost:8084/tutoriales/spring>, esta petición es manejada por el *Front Controller* quien se encarga analizar la *URL* para determinar cuál es el controlador adecuado para la misma.

2. El *Controller* o controlador usualmente accede a la base de datos y rellena el modelo con los datos requeridos para generar la vista, el modelo es enviado de regreso al *Front Controller* junto con el nombre lógico de la vista.

3. El componente encargado de generar las vistas es el *View Template* este utilizará el modelo y la tecnología de vista que hayamos configurado para generar la vista final, que puede ser *HTML*, *PDF*, *XLS*, etc., cuando la vista esté preparada será devuelta al *Front Controller* quien la pondrá a disposición del usuario.

5.4 JSP y Servlets

Los *servlets Java* y las páginas de servidor *Java (JSP)* son programas *Java* que se ejecutan en un servidor de aplicaciones *Java* y amplían las prestaciones del servidor web [11].

Los *servlets Java* son clases *Java* diseñadas para responder a solicitudes *HTTP* en el contexto de una aplicación web.

Puede ver las *JSP* como una extensión de *HTML* que le ofrece la capacidad de incluir fragmentos de código *Java* dentro de las páginas *HTML*. Estos fragmentos de código *Java* generan contenido dinámico, que está incluido dentro del otro contenido *HTML/XML*. Una *JSP* se convierte a un *servlet Java* y se ejecuta en el servidor. Las sentencias *JSP* incluidas en la *JSP* pasan a formar parte del *servlet* generado a partir de la *JSP*. El *servlet* resultante se ejecuta en el servidor.

El servidor *HTTP* no ejecuta aplicaciones web *Java* directamente. Las solicitudes *HTTP* para aplicaciones *Java* son reenviadas por el servidor *HTTP* a los servidores de aplicaciones *Java*.

Una aplicación Web realiza tareas de procesamiento y presentación:

- Los *Servlets* son adecuados para procesamiento.
- Las páginas *JSP* son adecuadas para presentación.

Una aplicación Web puede combinar *Servlets* y páginas *JSP*:

- Procesado de parámetros de la petición: *Servlets*.
- Acceso a bases de datos: *Servlets*.
- Lógica de la aplicación: *Servlets*.
- Presentación (vistas): *JSP*.

Modelo de funcionamiento, ver Figura 8.

1. El cliente envía la petición *HTTP* a un *servlet*.
2. El *servlet* procesa la petición.
 - Si es necesario, se conecta a la base de datos.
3. El *servlet* redirige la petición a un *JSP*.
 - Si es necesario, añade *beans* como parámetros.
4. El *JSP* lee los parámetros y devuelve la respuesta formateada visualmente al usuario.

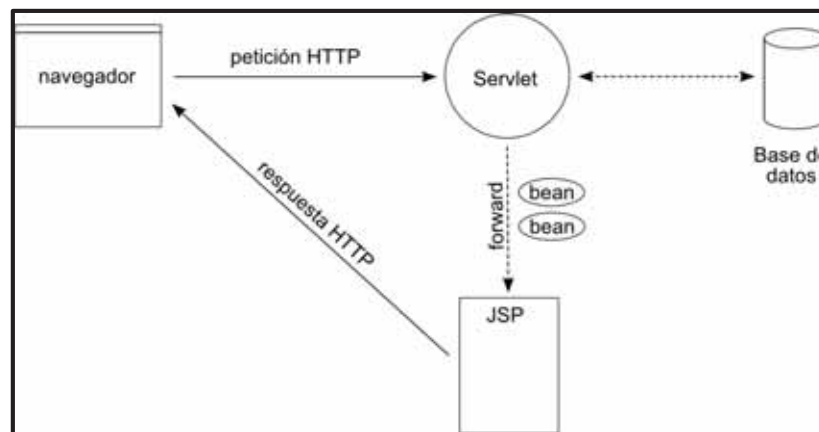


Figura 8. Modelo de funcionamiento, *Servlet* y *JSP*.

5.5 Bootstrap

Bootstrap es un *framework* desarrollado y liberado por *Twitter* que tiene como objetivo facilitar el diseño web. Permite crear de forma sencilla webs de diseño adaptable, es decir, que se ajusten a cualquier dispositivo y tamaño de pantalla y

siempre se vean igual de bien. Es *Open Source* o código abierto, por lo que lo podemos usar de forma gratuita y sin restricciones [12].

El resultado gusta a clientes/as y usuarios/as, ni más ni menos. A la gran mayoría de personas que nos contratan les preocupa que visitar su web sea una experiencia agradable y atractiva, sea desde un computadora o desde un móvil o tablet. Con *Bootstrap* se consigue esto, además de que la carga de la web sea rápida y que la navegación sea fluida e intuitiva. Además, facilita mucho la construcción de una página, que siempre viene bien.

Ventajas de usar *Bootstrap*

La más genérica es que permite simplificar el proceso de maquetación, sirviéndonos de guía para aplicar las buenas prácticas y los diferentes estándares. Aquí van unos cuantos pros más:

Puedes tener una web bien organizada de forma visual rápidamente: la curva de aprendizaje hace que su manejo sea asequible y rápido si ya sabes maquetar.

Permite utilizar muchos elementos web: desde iconos a desplegables, combinando *HTML5*, *CSS* y *Javascript*.

Sea lo que sea que creemos, el diseño será adaptable, no importa el dispositivo, la escala o resolución.

El *grid system*: maquetar por columnas nunca fue tan fácil. Además, son muy configurables.

Se integra muy bien con las principales librerías *Javascript*.

El haber sido creado por *Twitter* nos da ciertas garantías: está muy pensado y hay mucho trabajo ya hecho. Por lo tanto, hay una comunidad muy activa creando, arreglando cosas, ofreciendo plugins y mucho más..

Cuenta con implementaciones externas para *WordPress*, *Drupal*, etc.

Nos permite usar *Less*, para enriquecer aún más los estilos de la web

5.6 CSS y *HTML5*

HTML es el lenguaje de marcas con el cual están hechas la mayoría de las páginas web. Los archivos *CSS* encapsulan algunas características gráficas de las páginas web para evitar la redundancia de estar siempre declarando los atributos visuales [13].

HTML es un lenguaje muy simple que sirve para crear páginas web, de por sí no incluye instrucciones de acceso a base de datos ni control de flujo.

Por otro lado, el *CSS* es indicado para presentar información con formato, es decir aporta estilos a lo que se quiere mostrar.

CSS Es un lenguaje muy simple y está directamente relacionado con *HTML* en el sentido que *HTML* le da la posición a los objetos y *CSS* el estilo.

Hay dos formas de escribir CSS dentro de *HTML*: dentro de la misma etiqueta o tag *HTML* (CSS embebido) y, colocando todo lo referente a estilos dentro de un archivo con extensión .css, para luego incluir este archivo dentro de la página *HTML*.

5.7 Incidentes relacionados a la seguridad

Los tipos de incidentes que se llevaron a cabo en este proyecto son los siguientes:

- Robo a vehículos. Este tipo de incidentes se presenta cuando algún miembro de la comunidad fue víctima de alguna sustracción de cierto objeto que se hallaba dentro o fuera del vehículo estacionado.

- Robo a transeúnte. Hace referencia al incidente en el cual los miembros de la comunidad fueron despojados de algún objeto sin su consentimiento mientras estaban dentro o cerca de la unidad.

- Robo a aulas o instalaciones. Aplica para aquellos incidentes donde miembros de la comunidad fueron víctimas de alguna sustracción de cierto equipo o material de algún inmueble, como laboratorios, auditorios, talleres o cubículos.

- Bloqueos. Miembros de la comunidad o personas externas que impidan el acceso ya sea a algún edificio, pasillo o entradas de la unidad.

- Agresiones. Acontecimientos en los cuales los miembros de la comunidad se encuentran involucrados en un ataque o acto violento que pueda causar daño a los demás miembros.

- Incendios. Este tipo de incidentes se presenta en el caso de que miembros de la comunidad se percatan de un conato de incendio (fuego incipiente o inicial que puede controlarse), deben notificarlo a protección civil y ya controlado o extinguido el fuego, si desean los usuarios pueden emitir alguna alerta en la aplicación propuesta.

- Fuga de gas u otro químico. Este incidente se presenta cuando algunos miembros de la comunidad perciben una eventualidad relacionada con gas, de la misma manera deben notificarlo a protección civil y si desean pueden registrar dicho suceso con la finalidad de extremar las medidas de precaución necesarias para evitar una explosión.

- Fuga de otro químico. Este incidente se presenta cuando algunos miembros de la comunidad perciben una eventualidad relacionada algún químico, de la misma manera deben notificarlo a protección civil y si desean pueden registrar dicho suceso con la finalidad de extremar las medidas de precaución necesarias para evitar una intoxicación colectiva.

6. Desarrollo del proyecto

6.1 Base de datos

La base de datos está construida bajo el modelo entidad-relación como se muestra en la Figura 9, está compuesta de cuatro tablas que representan las entidades de nuestro modelo y cada una de estas entidades contienen atributos:

- **'reporte'** en esta tabla se almacenan los datos de un nuevo incidente registrado, los atributos que contiene son: `id_reporte` (llave primaria), `usuario_reporte`, `fecha`, `hora`, `latitud`, `longitud`, `descripcion`, `incidente_id_incidente` (llave foránea) y `espacio_fisico_id_espacio_fisico` (llave foránea).

- **'incidente'** en esta tabla se insertan los 8 tipos de incidentes que se consideran para este proyecto, los atributos que contiene son: `id_incidente` y `tipo_incidente`.

- **'espacio_fisico'** en esta tabla se insertan los 55 espacios físicos que abarcan la región total del mapa de la UAM Azcapotzalco, los atributos que contiene son: `id_espacio_fisico` (llave primaria) y `lugar`.

- **'opinion'** en esta tabla se almacenan los datos de una nueva opinión agregada, los atributos que contiene son: `id_opinion` (llave primaria), `usuario_opinion`, `comentario`, `valoracion`, `fecha_opinion`, `hora_opinion` y `reporte_id_reporte` (llave foránea).

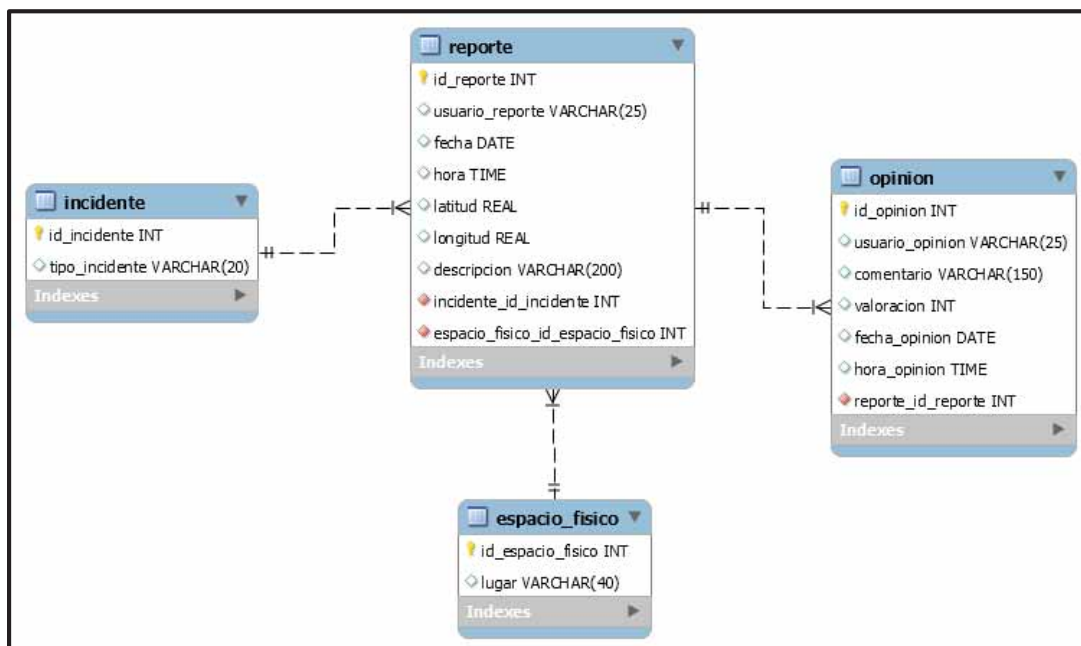


Figura 9. Modelo Entidad-Relación de la base de datos.

La cardinalidad o relación que hay entre la tabla `incidente` con la tabla `reporte` es de uno a muchos (1,n), ya que un incidente se relaciona con muchos reportes y muchos reportes se relaciona con un incidente.

La relación que hay entre la tabla `espacio_fisico` con la tabla `reporte` también es de uno a muchos (1,n), ya que un espacio físico se relaciona con muchos reportes y muchos reportes se relaciona con un espacio físico.

La relación que hay entre la tabla `opinion` con la tabla `reporte` es de muchos a uno (n,1), ya que muchas opiniones se relaciona con un reporte y un reporte se relaciona con muchas opiniones.

El script para la creación de la base de datos, la inserción de los 8 incidentes e inserción de los 55 espacios físicos puede observarse en la sección de Anexo 11.1.

6.2 Mapeo de entidades

Una entidad va a ser una simple clase *Java* que se desee persistir en la base de datos. Para las clases *Java*, para cada propiedad que se quiera persistir debe haber un método `get/set` asociado.

Archivo de mapeo "`Clase.hbm.xml`". **Para cada clase** que se quiera persistir **se creó un archivo XML** con la información que permite mapear la clase a una base de datos relacional. Este archivo está en el mismo paquete que la clase a persistir. En la Figura 10 se muestra, a manera de ejemplo, el mapeo de la entidad "Reporte", el resto se puede encontrar los Anexos 11.12, 11.18 y 11.19.



```
Reporte.hbm.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <hibernate-mapping>
3   <class name="uam.incidentes.objetos.Reporte" table="reporte">
4     <id name="idReporte" column="id_reporte" type="int">
5       <generator class="native"/>
6     </id>
7     <property name="usuarioReporte" column="usuario_reporte" type="string"/>
8     <property name="fecha" column="fecha" type="string"/>
9     <property name="hora" column="hora" type="string"/>
10    <property name="lat" column="latitud" type="double"/>
11    <property name="lng" column="longitud" type="double"/>
12    <property name="descripcion" column="descripcion" type="string"/>
13
14    <property name="idIncidenteFK" column="incidente_id_incidente" type="int"/>
15    <property name="idEspacioFK" column="espacio_fisico_id_espacio_fisico" type="int"/>
16
17  </class>
18 </hibernate-mapping>
```

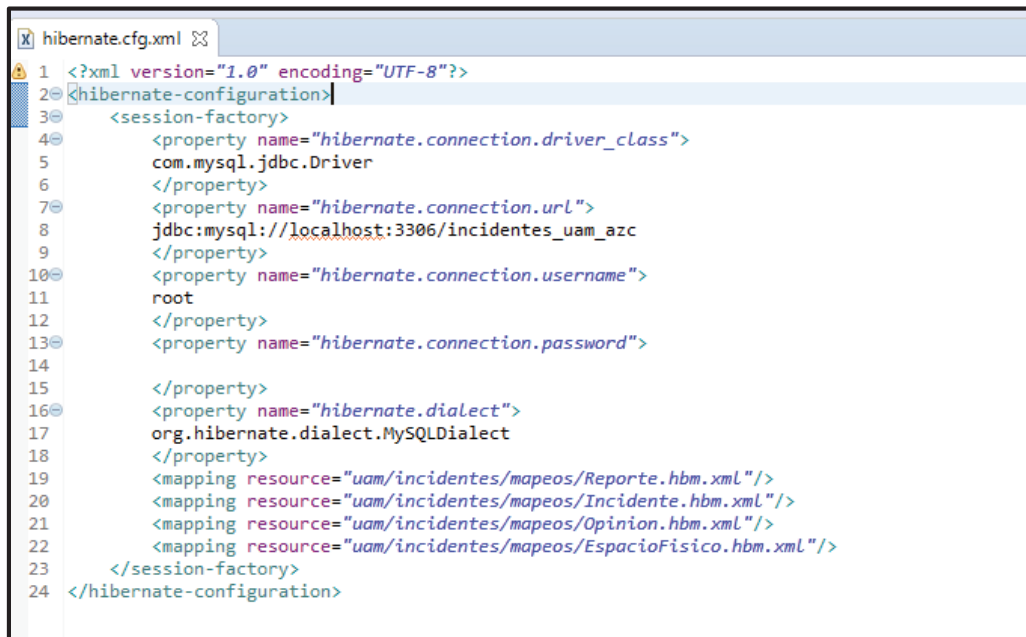
Figura 10. Archivo de mapeo "`Reporte.hbm.xml`".

El tag `<class>` indica que se va a mapear una clase, el atributo `name` contiene el nombre de la clase incluyendo el paquete en el que se encuentra y el atributo `table` contiene el nombre de la tabla en la que se va a mapear la clase. Los tags `<id>` y `<property>` en el atributo `name` contiene el nombre de la propiedad *Java* que se desea mapear a la base de datos, el atributo `column` contiene el nombre de la

columna de la base de datos asociado a la propiedad y el atributo *type* contiene el tipo del mapeo.

6.3 Conexión a la base de datos con *Hibernate*

La forma de configurar *Hibernate* es usando el archivo de configuración *XML* llamado *hibernate.cfg.xml*, ver Figura 11. Este archivo se guarda en el paquete raíz de nuestras clases *Java*, es decir fuera de cualquier paquete.



```
hibernate.cfg.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <hibernate-configuration>
3   <session-factory>
4     <property name="hibernate.connection.driver_class">
5       com.mysql.jdbc.Driver
6     </property>
7     <property name="hibernate.connection.url">
8       jdbc:mysql://localhost:3306/incidentes_uam_azc
9     </property>
10    <property name="hibernate.connection.username">
11      root
12    </property>
13    <property name="hibernate.connection.password">
14
15    </property>
16    <property name="hibernate.dialect">
17      org.hibernate.dialect.MySQLDialect
18    </property>
19    <mapping resource="uam/incidentes/mapeos/Reporte.hbm.xml"/>
20    <mapping resource="uam/incidentes/mapeos/Incidente.hbm.xml"/>
21    <mapping resource="uam/incidentes/mapeos/Opinion.hbm.xml"/>
22    <mapping resource="uam/incidentes/mapeos/EspacioFisico.hbm.xml"/>
23  </session-factory>
24 </hibernate-configuration>
```

Figura 11. Archivo de configuración *hibernate.cfg.xml*.

La información que contiene el archivo *XML* es la siguiente:

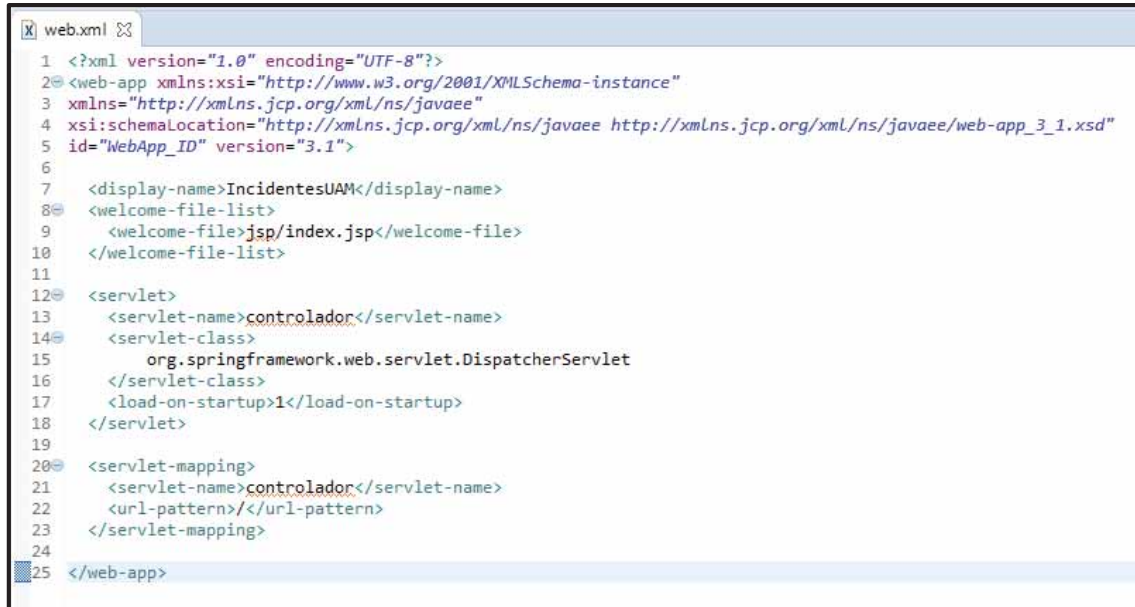
- **Propiedades de configuración.** La primera propiedad contiene el nombre del driver de la base de datos a usar. La segunda propiedad contiene la *URL* de conexión a la base de datos tal y como se usa en *JDBC*. La tercera propiedad contiene el usuario de la base de datos. La cuarta propiedad contiene la contraseña de la base de datos y la quinta propiedad contiene el lenguaje de *SQL* que usa *Hibernate* para la base de datos.
- **Las clases que se quieren mapear.** Se indica el nombre de los archivos de mapeo ".hbm.xml"

Para conectarse a la base de datos, se creó una nueva clase en 'CrearConexion.java', puede observarse en la sección de Anexo 11.2.

6.4 Configurar archivo *web.xml*

Se configura el archivo **web.xml** para que arranque al ejecutar la aplicación en el servidor. De esta forma, solo con indicar el nombre de la aplicación web 'IncidentesUAM' y colocando 'index.jsp' como punto de entrada a la aplicación, cuando se ejecute se levantará en el servidor.

Además, en **web.xml** se indica que hay un *servlet* con el nombre 'controlador', la clase de *Spring* a la que corresponde y el tipo de mapeo que manejará, en este caso al colocar '/' se indica el *servlet* por default, ver Figura 12.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
5 id="WebApp_ID" version="3.1">
6
7 <display-name>IncidentesUAM</display-name>
8 <welcome-file-list>
9 <welcome-file>jsp/index.jsp</welcome-file>
10 </welcome-file-list>
11
12 <servlet>
13 <servlet-name>controlador</servlet-name>
14 <servlet-class>
15 org.springframework.web.servlet.DispatcherServlet
16 </servlet-class>
17 <load-on-startup>1</load-on-startup>
18 </servlet>
19
20 <servlet-mapping>
21 <servlet-name>controlador</servlet-name>
22 <url-pattern>/</url-pattern>
23 </servlet-mapping>
24
25 </web-app>
```

Figura 12. Configuración de *web.xml*.

Se creó un archivo llamado **controlador-servlet.xml** en el directorio **WebContent/WEB-INF**, debido al nombre del *servlet*, ver Figura 13.

Antes de configurar fue necesario crear un paquete en el directorio *src*, donde se colocaron los Controladores, el nombre del paquete es: **uam.incidentes.controladores**. Esta ruta (nombre de paquete) hay que indicar que es aquella en la que se buscan de manera automática los controladores. También se configuró la manera en que se redireccionan las vistas una vez que sean invocadas por el controlador.

En la propiedad **prefix** se indica lo que se coloca antes de la palabra (nombre de la página) cuando se invoque una vista. En **suffix** se coloca lo que esta después. En este caso se indicó que se colocó **/jsp/** (la ruta en donde están las vistas) y que todas son **.jsp**.

Para las imágenes, los scripts externos *JavaScript* y los estilos *CSS*, *Bootstrap*, fue necesario configurar el entorno para indicar la ubicación de estos archivos, para esto al archivo de configuración de Spring, se le agregaron las siguientes líneas:

```
<mvc:resources mapping="/recursos/**" location="/WEB-INF/" />
<mvc:annotation-driven />
```

De esta forma el archivo **controlador-servlet.xml** quedó de la siguiente manera.


```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <beans xmlns="http://www.springframework.org/schema/beans"
4       xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:context="http://www.springframework.org/schema/context"
5       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6       xsi:schemaLocation="
7         http://www.springframework.org/schema/beans
8         http://www.springframework.org/schema/beans/spring-beans.xsd
9         http://www.springframework.org/schema/mvc
10        http://www.springframework.org/schema/mvc/spring-mvc.xsd
11        http://www.springframework.org/schema/context
12        http://www.springframework.org/schema/context/spring-context.xsd">
13
14     <context:component-scan base-package="uam.incidentes.controladores"/>
15
16     <bean id="viewResolver"
17         class="org.springframework.web.servlet.view.UrlBasedViewResolver">
18         <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
19         <property name="prefix" value="/jsp/" />
20         <property name="suffix" value=".jsp" />
21     </bean>
22
23     <mvc:resources mapping="/recursos/**" location="/WEB-INF/" />
24     <mvc:annotation-driven />
25
26
27 </beans>
28

```

Figura 13. Configuración de *controlador-servlet.xml*.

6.5 Módulo para el registro de incidentes

Este módulo permite a cualquier usuario, que sea miembro de la comunidad UAM, proporcionar información de algún incidente por el que haya pasado o sido testigo. Asimismo, es el encargado de almacenar sobre la base de datos, los datos ingresados en el formulario de la interfaz gráfica, entre ellos el tipo de incidente, fecha, hora, latitud, longitud, descripción, espacio físico de la unidad y nombre del usuario que reporta. Aunque el usuario puede omitir este último dato, puede reportar de manera anónima si así lo desea.

6.5.1 Diseño del módulo registrar

Para el diseño de este módulo primero se creó la clase 'Reporte.java' incluyendo sus atributos correspondientes junto con los métodos *getters* y *setters*, ver Figura 14, posteriormente mapear dicha clase con su respectiva tabla de la base de datos en un archivo *XML*, en el archivo de mapeo 'Reporte.hbm.xml', ver Figura 10.

El controlador de registro se creó dentro del paquete *uam.incidentes.controladores* como 'ControladorRegistrar.java', ver Figura 14. La clase *ControladorRegistrar* es la encargada de capturar la invocación a la liga (*registro_incidente*) y realizar el mapeo de un objeto *Reporte* a la forma a través del atributo *reporteFrm*. Para obtener los datos de la forma, se creó un nuevo método que atrapa la invocación al nombre especificado en el atributo *action*, en este caso *registrarReporte*. También se indica el tipo de objeto mapeado que se está recibiendo.

Sobre la misma clase *ControladorRegistrar*, se agrega la línea *@ModelAttribute("reporteFrm")Reporte reporte* a los parámetros del método *registrarReporte*, indicando que se está recuperando cierto atributo *reporteFrm* enviado de la forma y se mapea como un objeto *Reporte*.

Para facilitar la impresión de los datos en consola, se agregaron los métodos toString() a la clase *Reporte*.

Para realizar una inserción se invoca al método *insertarReporte(Reporte)* de la clase 'RegistroReporte.java', ver Figura 14, es necesario comenzar una Transacción (*Transaction*), esto es con el método *beginTransaction()* del objeto *Session*. Una vez creada, se utiliza el método *save()* al cuál se le pasa el objeto a guardar, el cual ya ha sido mapeado en los archivos *XML* correspondientes. Posteriormente se debe realizar un *commit* para asegurar que se reflejan los cambios en la base de datos.

Después de registrar el incidente, se manda a llamar una página genérica, en este caso la página de inicio *"/index"*.

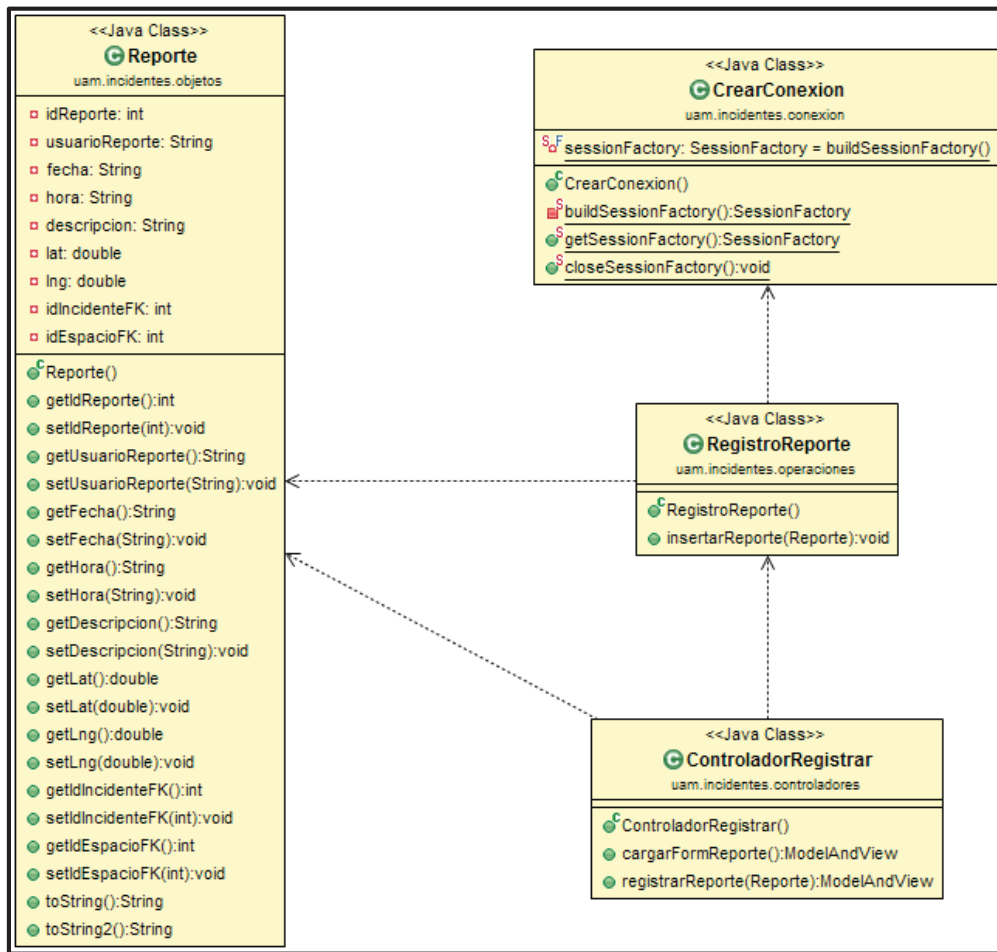


Figura 14. Diagrama de clases para registrar un reporte.

Dentro de la vista *JSP* 'registro_incidente.jsp' se usaron las etiquetas (tags) de *Spring*, agregando la siguiente línea.

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="tag"%>
```

Utilizando estas etiquetas se creó un formulario, dentro hay elementos *input* que contiene la propiedad *path* que representa el nombre de cada uno de los atributos de la clase Reporte, en el navegador se visualiza como la Figura 15.

El primer campo es el tipo de incidente y se elige mediante una lista de opciones, la fecha y hora se generan automáticamente al día actual utilizando otra función en *JavaScript* 'obtenerFechaHora.js', la latitud y longitud se obtienen al mover el marcador del mapa, la descripción es una caja de texto y el usuario solo debe de agregar los detalles de lo sucedido y en el último campo el usuario puede colocar su nombre.

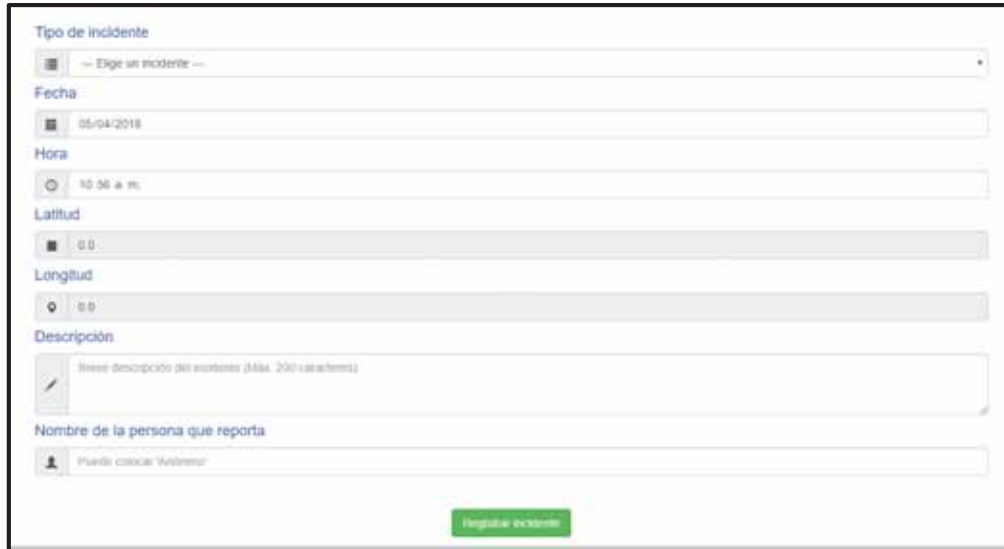
The image shows a web form titled 'Formulario para registrar reporte'. It contains several input fields: a dropdown menu for 'Tipo de incidente' with the placeholder text 'Elige un incidente...'; a date field for 'Fecha' showing '05/04/2018'; a time field for 'Hora' showing '10:56 a.m.'; latitude and longitude fields, both showing '0.0'; a text area for 'Descripción' with a placeholder 'Escribe descripción del incidente (Máx. 200 caracteres)'; and a text field for 'Nombre de la persona que reporta' with a placeholder 'Puede colocar "Anónimo"'. At the bottom center of the form is a green button labeled 'Registrar incidente'.

Figura 15. Formulario para registrar reporte.

Utilizando la *API* de *Google Maps*, dentro del código *JavaScript* 'registrarMapa.js' se declaró la función *initMap()*, para que sea inicializada y muestre el mapa una vez cargada la página, en esta función se añadió lo siguiente.

- Límites del mapa. Para que el mapa solo sea desplazado cerca de la región de la UAM Azcapotzalco. Si el usuario intenta moverse a una región distinta a la escuela, solo puede hacerlo a calles que se encuentren cercanas.
- Marcadores (iconos). Para saber dónde están ubicados la sección de vigilancia, protección civil, servicio médico, edificios y accesos de la universidad.
- Límites del marcador. Para que el marcador no sea arrastrado fuera del área de la UAM Azcapotzalco. Si el usuario intenta arrastrar al marcador en un área distinta a la escuela, no puede hacerlo, ya que solo se registran sucesos que ocurran dentro de la universidad.
- Polígonos. Para determinar los distintos espacios que conforman a la UAM Azcapotzalco. El polígono principal es la región de la universidad, está marcado con un contorno rojo sobre el mapa, los polígonos de los edificios y entradas son rellenados de cierto color sobre el mapa, el resto de espacios (estacionamiento, áreas verdes, plazas, canchas y kioskos) no son rellenados de algún color, pero se contemplan para el registro.

- Evento para obtener coordenadas. Para llenar los campos de latitud y longitud del formulario.
- Ventanas de información. Para mostrar las coordenadas y lugar donde este posicionado el marcador cuando es arrastrado.

6.5.2 Implementación del módulo registrar

Parte de los códigos para el registro de incidentes, puede observarse en los Anexos 11.3 - 11.7. En la Figura 16, se tiene la vista para reportar un incidente, primero se debe de posicionar o arrastrar el marcador en aquel punto donde ocurrió el incidente, esto servirá para obtener los campos latitud y longitud del formulario. Para registrar un reporte válido, el usuario debe elegir un tipo de incidente, colocar una breve descripción del hecho y su nombre (si lo prefiere). Por medio de validaciones, si el usuario deja algún campo vacío, no podrá registrar su incidente hasta llenar el campo faltante.











Figura 16. Vista para el registro de incidente.




6.6 Módulo para la consulta de incidentes

La principal función de este módulo consiste en mostrar al usuario aquellos incidentes que ya ocurrieron y fueron registrados por parte de los miembros de la comunidad. Estos incidentes son visualizados de distintas formas y se observan

sobre el mapa de la UAM Azcapotzalco añadiendo una tabla que incluye todos los datos consultados. El usuario puede consultar incidentes de 3 formas, por tipo de incidente, espacio físico y fecha, estos son visualizados por un marcador de cierto color, conforme a la siguiente lista:

-  Robo a vehículo. Marcador de color verde.
-  Robo a transeúnte. Marcador de color azul oscuro.
-  Robo a instalación. Marcador de color púrpura.
-  Agresión Marcador de color marrón.
-  Bloqueo. Marcador de color rojo.
-  Incendio. Marcador de color naranja.
-  Fuga de gas. Marcador de color amarillo.
-  Fuga de otro químico. Marcador de color azul cielo.

Además, sobre el mapa se visualizan 3 lugares importantes, por si el usuario desconoce la ubicación de estos:

-  Sección de Vigilancia, ubicado en el edificio E, Planta Baja.
-  Protección Civil, ubicada en el edificio C, Primer Piso.
-  Servicio Médico, ubicado en el edificio E, Planta Baja.

6.6.1 Diseño del módulo consultar (incidente, espacio y fecha)

Para el diseño de este módulo se creó la clase 'ReporteIncidenteEspacio.java' con sus respectivos atributos y sus métodos *getters* y *setters*, ver Figura 17, posteriormente se mapeó la clase 'Incidente.java', 'EspacioFisico.java' y 'Opinion.java' con sus respectivas tablas de la base de datos en varios archivos XML, en los archivos de mapeo 'Incidente.hbm.xml', 'EspacioFisico.hbm.xml' y 'Opinion.hbm.xml'

El controlador de consultas se creó dentro del paquete uam.incidentes.controladores como 'ControladorConsutar.java', ver Figura 17. La clase *ControladorConsutar* es la encargada de capturar la invocación a las ligas (*consulta_incidente*, *consulta_espacio* y *consulta_fecha*) y realizar el mapeo de un objeto *ReporteIncidenteEspacio* a las formas a través de los atributos *IncidenteFrm*, *EspacioFrm* y *FechaFrm*.

Para obtener los datos de las formas, se crearon nuevos métodos que atrapan la invocación a los nombres especificados en el atributo *action*, en este caso *consultarIncidente*, *consultarEspacio* y *consultarFecha*. También se indica el tipo de objeto mapeado que se está recibiendo.

Sobre la misma clase *ControladorConsultar*, se agregan las líneas *@ModelAttribute("IncidenteFrm")ReporteIncidenteEspacio rep*,

@ModelAttribute("EspacioFrm")ReporteIncidenteEspacio repEspacio y

@ModelAttribute("FechaFrm")ReporteIncidenteEspacio repFecha.

A los parámetros de los métodos *mostrarConsulta*, *mostrarConsultaEspacio* y *mostrarConsultaFecha*, se indica que se está recuperando ciertos atributos *IncidenteFrm*, *EspacioFrm* y *FechaFrm* enviado de las formas y se mapean como un objeto *ReporteIncidenteEspacio*.

Para facilitar la impresión de los datos en consola, se agregaron los métodos *toString()* a la clase *ReporteIncidenteEspacio*.

La clase *ReporteIncidenteEspacio* contiene los elementos a obtener de cada consulta. El principal problema es que no se tiene una tabla en la base de datos que contenga todos los elementos, por lo que no se cuenta con un mapeo. En este caso se debe llenar la clase a utilizar a partir de los resultados regresados por cada sentencia.

Para realizar una consulta se invoca al método correspondiente *reporteIncid(ReporteIncidenteEspacio)*, *reporteEspac(ReporteIncidenteEspacio)*, *reporteFecha(ReporteIncidenteEspacio)* estos métodos son del tipo *List<ReporteIncidenteEspacio>* de la clase 'ConsultaReporte.java', ver Figura 17.

Para cada consulta se creó una sentencia.

Ejemplo, para construir la sentencia, se debe tener:

```
String sentencia = "SELECT ALIAS1.atributo1, ALIAS1.atributo2,
ALIASN.atributoN FROM Clase1 as ALIAS1, ClaseN as ALIASN";
```

Donde:

- Clase1 y ClaseN son clases mapeadas a su tabla correspondiente.
- ALIAS1 y ALIASN son nombres que facilitan el manejo de las clases.
- atributo1, atributo2 y atributoN son atributos de las clases java que están mapeadas a una tabla.

Usando la *sesion* se llama al método *createQuery(sentencia)* y retorna un objeto *Query*. Después, sobre el objeto *Query* se llama al método *list()* que regresa una lista de los objetos que ha retornado.

```
Query query = sesion.createQuery(sentencia);
```

```
lista = query.list();
```

Aquí se obtiene una lista de arreglos de objetos (*Object []*), cada elemento de la lista corresponde a la información que se tendría en la tupla obtenida por la base de datos y cada elemento del objeto corresponde a una columna (en el orden especificado por los nombres en la sentencia).

Dentro de un ciclo *for*, los datos recuperados se guardan en un arreglo de objetos, creando una instancia de un objeto a la clase *ReporteIncidenteEspacio*, retornando una lista del tipo *LinkedList <>*, en la que se obtiene la consulta correspondiente.

Después por cada consulta, los datos resultantes se muestran en la misma página, en este caso la página consulta por incidente “/consulta_incidente_vista”, la página consulta por espacio físico “/consulta_espacio_vista” y la página consulta por fecha “/consulta_fecha_vista”.

El método *reporteIncEspac(ReporteIncidenteEspacio)* de la clase ‘ConsultaReporte.java’, obtiene el total de los reportes registrados, pero este se utiliza en el siguiente módulo para la vista de la aplicación web “**Reportes Registrados - Agregar Opinión**”.

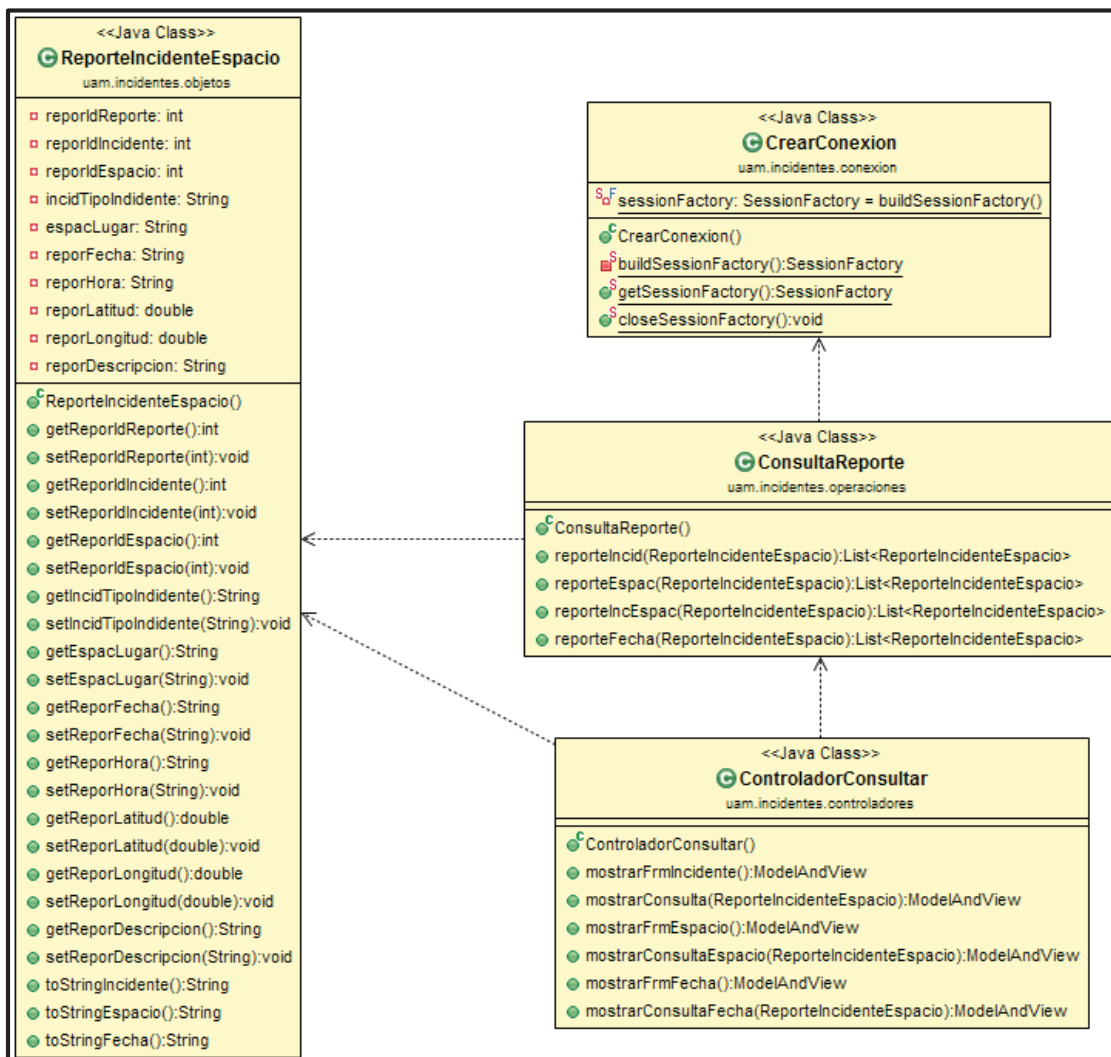


Figura 17. Diagrama de clases para realizar consultas.

6.6.2 Implementación del módulo consultar (incidente, espacio y fecha)

Parte de los códigos para las consultas, puede observarse en la sección de Anexos 11.8 - 11.10. En la Figura 18, Figura 19 y Figura 20 se muestra la vista para consultar por alguno de los 8 incidentes considerados, para consultar por alguno de los 55 espacios considerados y para consultar por fecha por medio de un calendario, al realizar cualquier consulta se muestran los incidentes sobre el mapa y además en la parte inferior se incluye una tabla que contiene los datos de la consulta realizada. En caso de que la consulta no exista o no haya datos, se le notifica al usuario.



Figura 18. Vista para consultar por incidente.



Figura 19. Vista para consultar por espacio físico.



Figura 20. Vista para consultar por fecha.

6.7 Módulo para la opinión de incidentes

Este módulo permite al usuario añadir su opinión sobre el incidente que ya está registrado y se encuentre visualizado en el mapa. Al contar con opiniones, es más precisa la información que el usuario proporcione, extendiendo la información de lo sucedido. Además, el usuario puede añadir una clasificación del incidente a través de la escala basada en estrellas, las cuales serán consideradas para determinar la relevancia del incidente. Esto significa que el usuario volara el incidente, por lo que una estrella significa “nada relevante” y cinco estrellas es sinónimo de “muy relevante”.

Así que este módulo hace de la aplicación, un ambiente colaborativo para la comunidad universitaria, donde el usuario aporta sus opiniones y valoraciones sobre los incidentes. Estas valoraciones serán de utilidad para el resto de la comunidad (usuarios de este sistema).

6.7.1 Diseño del módulo opiniones

En este módulo se usan las mismas instrucciones que se usaron en el registro y consulta de incidentes, pero adaptándolo a la clase opinión.

Agregar opinión.

Se creó la clase ‘Opinion.java’ incluyendo sus atributos correspondientes junto con los métodos *get/set*, ver Figura 21, posteriormente mapear dicha clase con su respectiva tabla de la B.D. en un archivo XML, en el archivo de mapeo ‘Opinion.hbm.xml’.

El controlador de opiniones se creó como ‘ControladorOpinion.java’, ver Figura 21. La clase *ControladorOpinion* es la encargada de capturar la invocación a la liga (*total_reportes*) y realizar el mapeo de un objeto *Opinion* a la forma a través del atributo *OpinionFrm*. Para obtener los datos de la forma, se creó un nuevo método que atrapa la invocación al nombre especificado en el atributo *action*, en este caso *registrarOpinion*.

Sobre la misma clase *ControladorOpinion*, se agrega la línea `@ModelAttribute("OpinionFrm")Opinion op` a los parámetros del método *registrarOpinion*, indicando que se está recuperando cierto atributo *OpinionFrm* enviado de la forma y se mapea como un objeto *Opinion*.

Para facilitar la impresión de los datos en consola, se agregaron los métodos `toString()` a la clase *Opinion*.

Para realizar una inserción se invoca al método *insertarOpinion(Opinion)* de la clase 'OpinionReporte.java', ver Figura 21, es necesario comenzar una Transacción (*Transaction*), esto es con el método *beginTransaction()* del objeto *Session*. Una vez creada, se utiliza el método *save()* al cuál se le pasa el objeto a guardar, el cual ya ha sido mapeado en los archivos *XML* correspondientes. Posteriormente se debe realizar un *commit* para asegurar que se reflejan los cambios en la base de datos.

Después de insertar la opinión, se manda a otra página, en este caso la página de inicio `/index`.

Mostrar opiniones.

Nuevamente la clase *ControladorOpinion* es la encargada de capturar la invocación a la liga (*opiniones*) y realizar el mapeo de un objeto *Opinion* a la forma a través del atributo *mostrarOpinionFrm*.

Para obtener los datos de la forma, se creó un método que atrapa la invocación al nombre especificado en el atributo *action*, en este caso *mostrarOpinion*. También se indica el tipo de objeto mapeado que se está recibiendo.

Sobre la misma clase *ControladorOpinion*, se agrega la línea `@ModelAttribute("mostrarOpinionFrm")Opinion op`, a los parámetros del método *visualizarOpiniones*, indicando que se está recuperando cierto atributo *mostrarOpinionFrm* enviado de la forma y se mapea como un objeto *Opinion*.

Para facilitar la impresión de los datos en consola, se agregaron los métodos `toString()` a la clase *Opinion*.

Para realizar la consulta se invoca al método correspondiente *opinionValoracion(Opinion)* este método es del tipo `List<Opinion>` de la clase 'ObtenerOpiniones.java', ver Figura 21.

Para esta consulta también se creó una sentencia. Usando la *sesion* se llama al método *createQuery(sentencia)* y retorna un objeto *Query*. Después, sobre el objeto *Query* se llama al método *list()* que regresa una lista de los objetos que ha retornado.

```
Query query = sesion.createQuery(sentencia);  
lista = query.list();
```

Aquí se obtiene una lista de arreglos de objetos (`Object []`), cada elemento de la lista corresponde a la información que se tendría en la tupla obtenida por la

base de datos y cada elemento del objeto corresponde a una columna (en el orden especificado por los nombres en la sentencia).

Dentro de un ciclo *for*, los datos recuperados se guardan en un arreglo de objetos, creando una instancia de un objeto a la clase *Opinion*, retornando una lista del tipo *LinkedList* <>, en la que se obtiene la consulta.

Después de la consulta, el resultado se muestra en la misma página, en este caso la página mostrar opiniones es */opiniones_agregadas*”.

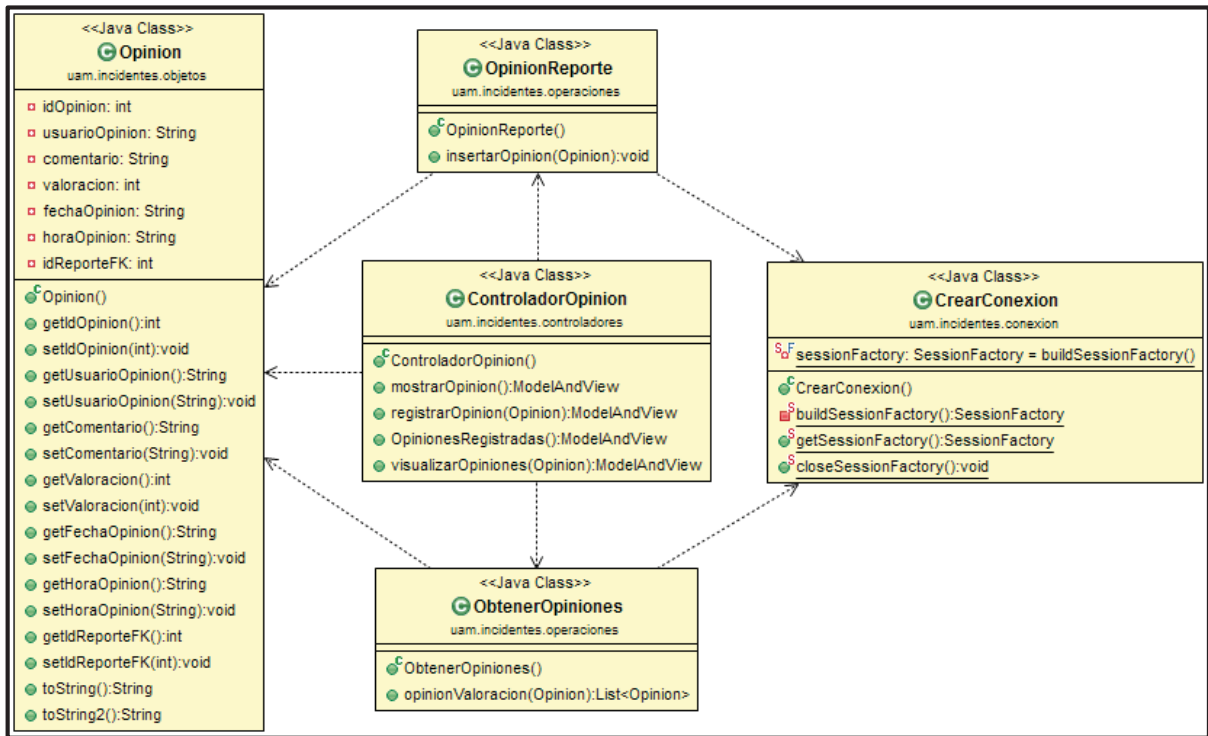


Figura 21. Diagrama de clases para agregar y mostrar opiniones.

Para navegar entre las páginas de la aplicación web, se agregan estas ligas en la clase *vistasReferencias.java*. Estas tienen referencia a la página de inicio, página acerca del proyecto, página proyecto tecnológico y página contacto.

El nombre de la liga (*home, about, project, contact*) son las que va a atrapar el método *vistasReferencias* y los métodos *clickHome()*, *clickAcerca()*, *clickProyecto()*, *clickContacto()* son los que atrapan la petición de la vista *JSP*, ver Figura 22.

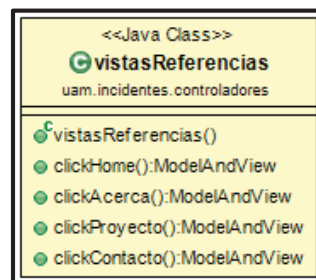


Figura 22. Diagrama de clase para navegar entre páginas *JSP*.

6.7.2 Implementación del módulo opiniones

Parte de los códigos para agregar y mostrar opiniones, puede observarse en la sección de Anexos 11.11 - 11.17. En la Figura 23 se tiene la visualización del total de reportes registrados, en este caso como no se ha generado algún reporte de prueba, indica que no hay reportes. Debajo del mapa se tienen dos botones, el primero sirve para agregar una opinión sobre algún incidente existente. Al presionar el primer botón se muestra un formulario como el de la Figura 24. En este formulario basta con seleccionar algún incidente existente, agregar nombre (si lo prefiere), algún comentario y una clasificación para indicar la relevancia.



Figura 23. Vista para ver el total de reportes registrados.

The image shows a form for adding an opinion. At the top, there are two buttons: a green one with a speech bubble icon and a yellow one with a speech bubble icon. Below them, a text prompt says 'Solo debe colocar nombre (opcional), comentario y clasificación'. The form has the following fields: 'No. Reporte' (with a pencil icon and a value of 0), 'Nombre' (with a placeholder 'Puede omitir el nombre'), 'Comentario' (with a placeholder 'Escriba comentario (máx. 100 caracteres)'), 'Clasificación' (with a star rating of 5 stars), 'Fecha' (with a value of 29/04/2016), and 'Hora' (with a value of 12:31 a.m.). At the bottom of the form is a yellow button labeled 'Guardar opinión'. Below the form, there is a red text prompt: 'Para más información del total de los incidentes registrados, consulte la siguiente tabla.' and a dropdown menu labeled 'Mostrar/Ocultar tabla de reportes'.

Figura 24. Vista para agregar una opinión.

Al presionar el segundo botón, se redirecciona a la vista mostrar opiniones, que es la vista de la Figura 25. Para ver las opiniones de algún marcador, solo basta con seleccionar un incidente del mapa, en automático se obtiene el no. de reporte en campo del panel y solo hay que presionar el botón 'Mostrar opiniones', con esto la página se recarga y agrega las opiniones encontradas del incidente, ver la Figura 26.

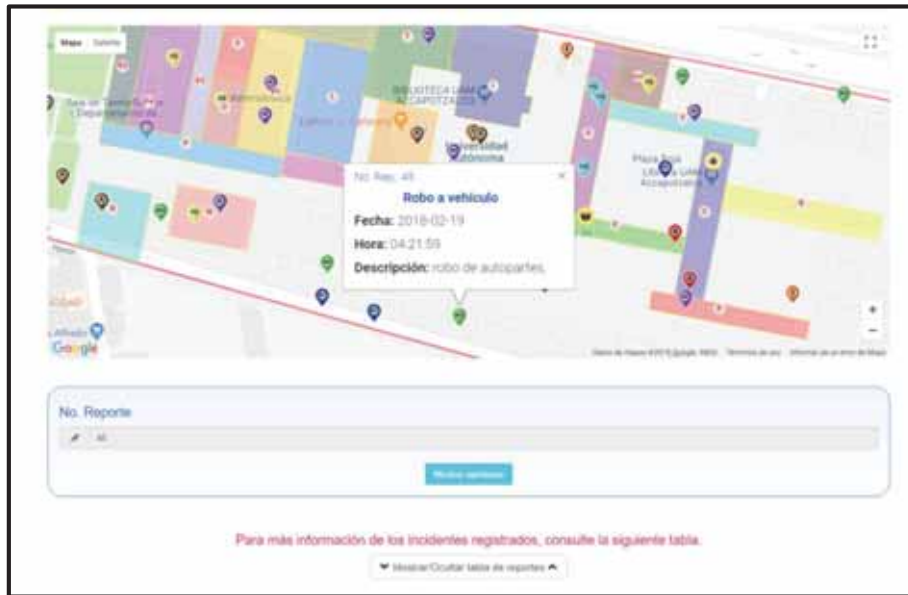


Figura 25. Vista para mostrar opiniones de algún incidente.

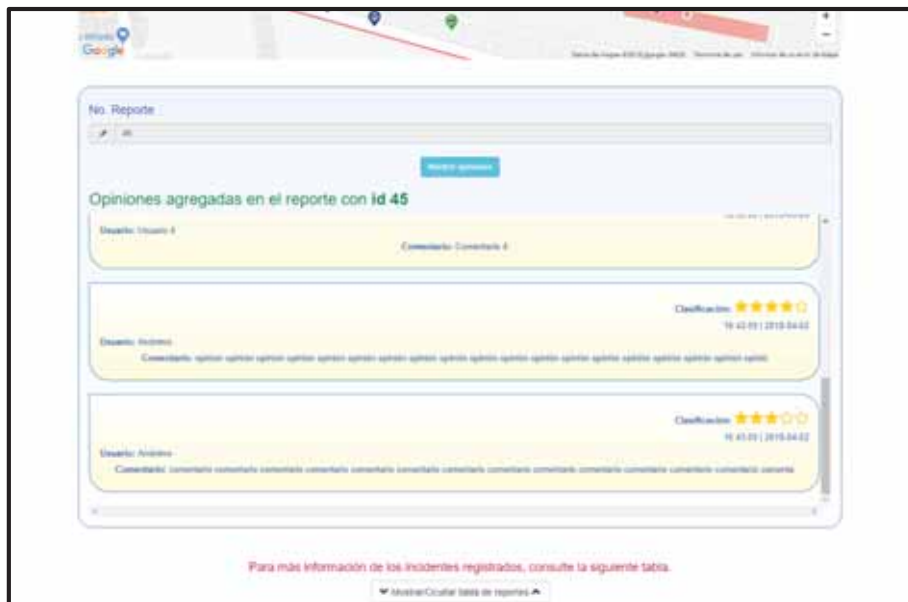


Figura 26. Vista de opiniones agregadas en un incidente.

6.8 Interfaz gráfica de usuario

Para que las vistas *JSP* se muestren correctamente y el zoom funcione bien en los dispositivos móviles, se añadió parte de la siguiente etiqueta dentro de la cabecera `<head>` de las vistas *JSP*:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Utilizando *Bootstrap* se ajustó el tamaño de las rejillas (filas, columnas) para tener una interfaz gráfica adecuada y se ajustó de acuerdo al tamaño de la pantalla el encabezado (*header*), cuerpo de contenido, menú y pie de página (*footer*). Por lo tanto, la aplicación web puede ser visualizada en móviles, tablets, laptops o máquinas de escritorio, ver Figura 27.

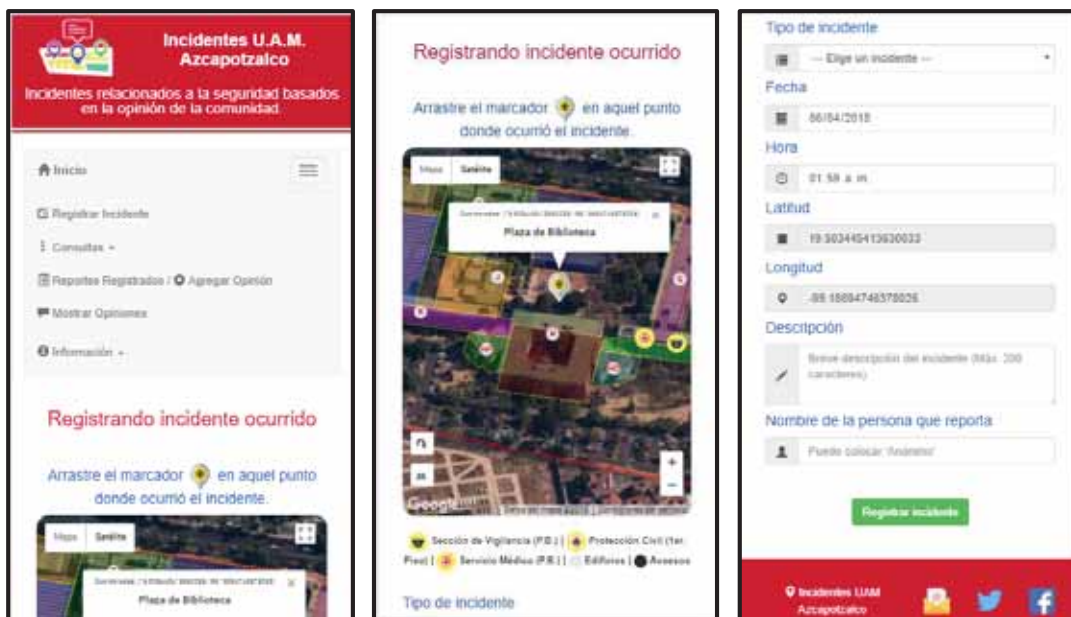


Figura 27. Vista versión móvil para el registro del incidente.

El encabezado o **header** tiene un fondo de color rojizo, que es el color representativo de la UAM Azcapotzalco, contiene un logo que representa a la aplicación, un título y un breve párrafo.

En el cuerpo de contenido, contiene un **menú** con los siguientes enlaces:

- *Inicio*: Muestra la página principal de la aplicación, en el viene colocado un párrafo de bienvenida, un carrusel de fotos haciendo referencia a los tipos de incidentes relacionados y una liga que incluye los teléfonos de emergencia de la universidad, ver Figura 28.

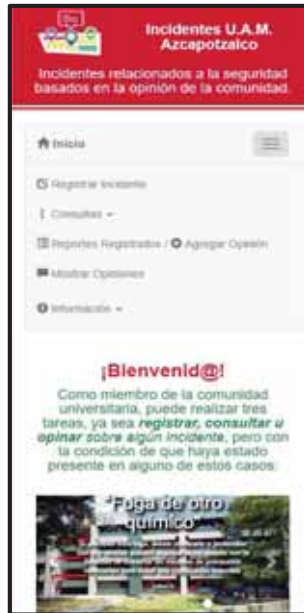


Figura 28. Vista versión móvil de la página de inicio.

- *Registrar Incidente*: Muestra el mapa de la unidad para agregar la ubicación del hecho, una sección de iconos que se visualizan en el mapa y el formulario para dar de alta un reporte, ver Figura 29.



Figura 29. Vista versión escritorio para el registro de incidente.

- *Consultas /*
 - *Consultar por Incidente*: Muestra un campo de lista seleccionable, para elegir el tipo de incidente y mostrar los incidentes sobre el mapa, al hacer click sobre un marcador del mapa se muestra una ventana de información que está asociada a una tabla que está en la parte inferior e incluye los detalles de cada incidente, ver Figura 30.



Figura 30. Vista versión móvil para la consulta por incidente.

- *Consultar por Espacio Físico:* Muestra un campo de lista seleccionable, para elegir el espacio físico y visualizar los incidentes sobre el mapa, al hacer click sobre un marcador del mapa se muestra una ventana de información que también está relacionada a una tabla, ver Figura 31.



Figura 31. Vista versión móvil para la consulta por espacio físico.

- *Consultar por Fecha:* Muestra un campo del tipo fecha, para elegir el día por medio de un calendario y mostrar los incidentes sobre el mapa, al hacer click sobre un marcador del mapa se muestra una ventana de información que también está relacionada a una tabla, ver Figura 32.



Figura 32. Vista versión móvil para la consulta por fecha.

- *Reportes Registrados – Agregar Opinión*: Muestra sobre el mapa la totalidad de los incidentes dados de alta, además cuando se elija un marcador del mapa, se despliega una ventana de información que incluye algunos detalles del incidente también se relaciona a una tabla. Además, se tiene la opción de agregar una opinión del incidente, en la opinión basta con agregar nombre (opcional), comentario y clasificación, ver Figuras 33 y 34.



Figura 33. Vista versión móvil de los reportes registrados.



Figura 34. Vista versión móvil para agregar una opinión.

- **Mostrar Opiniones:** Muestra las opiniones que se han agregado en los incidentes, solo basta con elegir un marcador y hacer click en el botón para visualizarlos, ver Figura 35.



Figura 35. Vista versión móvil para ver opiniones agregadas.

- **Información /**

- **Acerca del Proyecto:** Muestra un panel de información de la aplicación web. La información se describe por medio de preguntas ¿Qué es?, ¿Para qué sirve? y ¿Cómo funciona?, ver Figura 36.

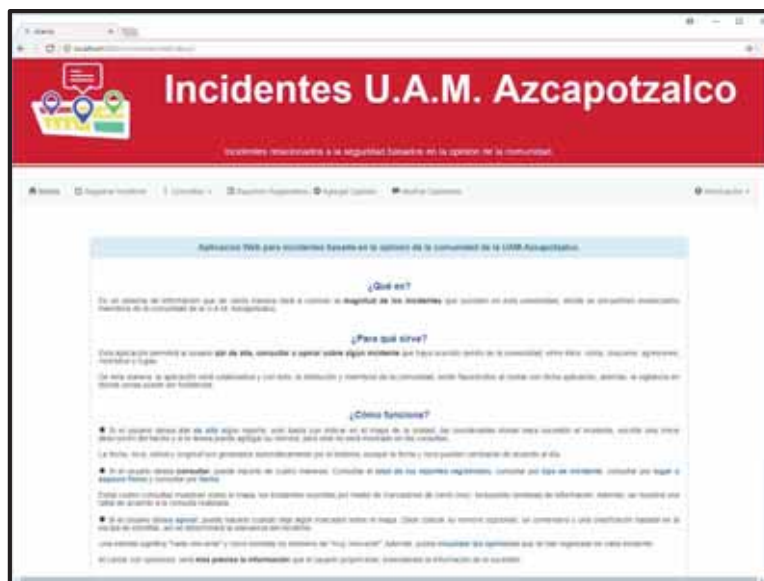


Figura 36. Vista versión escritorio de la página acerca del proyecto.

- *Proyecto Tecnológico*: Muestra datos del proyecto, alumno, asesor y redes sociales del Área de Investigación en Sistemas de Información Inteligentes (AISII), ver Figura 37.



Figura 37. Vista versión escritorio de la página proyecto tecnológico.

- *Contacto*: Muestra el nombre, correo electrónico y foto del alumno, ver Figura 38.



Figura 38. Vista versión escritorio de la página contacto.

El pie de página o **footer** tiene el mismo color de fondo que el *header*, contiene una liga al 'Acerca del Proyecto' y tres iconos que direccionan a las redes sociales del alumno.

7. Resultados

7.1 Resultados de registro

Para comprobar que los resultados del primer módulo sean correctos, se agregaron *50 reportes de prueba*, con esto al menos se consideran más de uno por cada tipo de incidente, el número de reportes registrados se indican en la siguiente lista:

- Robo a vehículo: 8 reportes.
- Robo a transeúnte: 3 reportes.
- Robo a instalación: 9 reportes.
- Agresión: 9 reportes.
- Bloqueo: 11 reportes.
- Incendio: 3 reportes.
- Fuga de gas: 4 reportes.
- Fuga de otro químico: 3 reportes.

Obteniendo el porcentaje de cada tipo de incidente, se obtiene una gráfica circular, ver Figura 39.

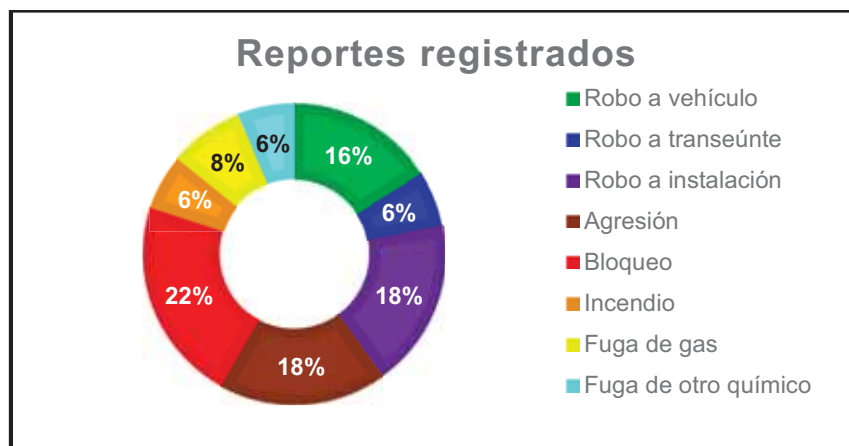


Figura 39. Porcentaje de los reportes de prueba registrados.

7.2 Resultados de consulta

Para comprobar que los resultados del segundo módulo sean correctos, se consultó *3 pruebas de consulta*. Primero por tipo de incidente se eligió 'Bloqueo', ver Figura 40, obteniendo los *11 incidentes* esperados sobre el mapa y en la tabla los datos completos, ver Figura 41. Después, por espacio físico se eligió 'Edificio G', ver Figura 42, obteniendo *3 incidentes* sobre el mapa y la tabla, ver Figura 43. Y por último, por fecha se eligió '18/02/2018', ver Figura 44, obteniendo *19 incidentes* sobre el mapa y la tabla, ver Figura 45.



Figura 40. Consulta sobre el mapa del incidente Bloqueo.

Consultó el tipo de incidente **Bloqueo** y hay **11** incidente(s) registrado(s). Para más información, consulte la siguiente tabla.

Mostrar/Ocultar tabla consultada

NO. REP.	INCIDENTE	LUGAR	FECHA	HORA	LATITUD	LONGITUD	DESCRIPCIÓN
7	Bloqueo	Entrada 4	2018-01-15	10:29:58	19.50278464778562	-99.18363221048422	bloqueo en la entrada 4
17	Bloqueo	Edificio E	2018-02-18	07:39:59	19.50287091057436	-99.18553121444677	bloqueo pasillo del E, primer piso
19	Bloqueo	Entrada 6	2018-02-18	09:40:57	19.504782009964954	-99.18907709476446	bloqueo entrada 6
25	Bloqueo	Entrada 7	2017-11-16	18:50:51	19.502996986578304	-99.1901660716245	bloqueo entrada 7
36	Bloqueo	Entrada 2	2018-02-18	22:49:09	19.50185068150811	-99.18394871112798	bloqueo entrada 2
40	Bloqueo	Entrada 2	2018-02-19	16:25:28	19.50182207968774	-99.18388970252971	bloqueo entrada 2
41	Bloqueo	Entrada 5	2018-01-19	18:27:34	19.503323822019907	-99.18353565003969	bloqueo entrada 5
43	Bloqueo	Edificio C	2018-02-19	17:53:22	19.502551498027326	-99.18542392608617	bloqueo edificio D
46	Bloqueo	Entrada 6	2018-02-21	12:11:27	19.50478979984214	-99.18907709476446	bloqueo entrada 6
47	Bloqueo	Entrada 5	2018-02-21	12:32:56	19.50331667178034	-99.18254637977575	bloqueo entrada 5
48	Bloqueo	Entrada 4	2018-02-21	12:34:38	19.5028017889050	-99.1836643969724	bloqueo entrada 4

Figura 41. Consulta sobre la tabla del incidente Bloqueo.

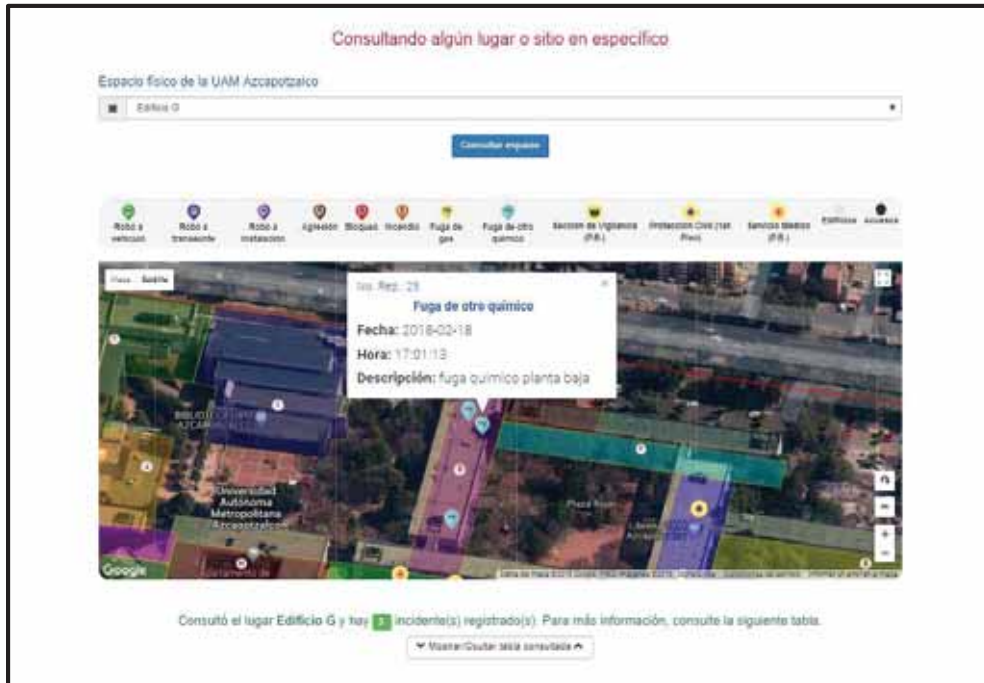


Figura 42. Consulta sobre el mapa del espacio físico Edificio G.

Consultó el lugar **Edificio G** y hay **3** incidente(s) registrado(s). Para más información, consulte la siguiente tabla.

Mostrar/Ocultar tabla consultada

NO. REP.	INCIDENTE	LUGAR	FECHA	HORA	LATITUD	LONGITUD	DESCRIPCIÓN
2	Fuga de otro químico	Edificio G	2018-02-18	18:23:19	19.503315593691713	-99.18619103786443	breve descripción fuga químico planta baja
13	Fuga de otro químico	Edificio G	2018-02-05	15:36:47	19.50386170766283	-99.18612696464808	fuga químico
29	Fuga de otro químico	Edificio G	2018-02-18	17:01:13	19.503790915215	-99.18607302066778	fuga químico planta baja

Figura 43. Consulta sobre la tabla del espacio físico Edificio G.



Figura 44. Consulta sobre el mapa de la fecha 28/02/2018.

Consultó la fecha 2018-02-18 y hay 18 incidente(s) registrado(s). Para más información, consulte la siguiente tabla.

Mostrar/Ocultar tabla consultada

#	descripcion	ubicacion	fecha hora	hora	latitud	longitud	telefono	descripcion alternativa sobre el incidente
18	Bloqueo	Entrada 6	2018-02-18	06:40:57	19.504782006664954	-99.18907709476448		bloqueo entrada 6
21	Robo a transeúnte	Plaza Roja	2018-02-18	12:46:33	19.50331053707225	-99.18560095188116		robo
22	Incendio	Áreas Verdes G1	2018-02-18	12:47:32	19.504119584174952	-99.18631441847912		incendio leve y fue controlado
24	Fuga de gas	Edificio W	2018-02-18	18:49:47	19.50300208298643	-99.18901808616513		fuga de gas 3er piso
29	Fuga de otro químico	Edificio G	2018-02-18	17:01:13	19.503790915215	-99.18607302066778		fuga químico planta baja
32	Robo a instalación	Plaza de Biblioteca	2018-02-18	22:37:47	19.503431089974406	-99.1871405398557		robo
33	Robo a instalación	Edificio P1	2018-02-18	21:09:32	19.50367422836807	-99.1885084664533		robo p1 primer piso
36	Bloqueo	Entrada 2	2018-02-18	22:49:06	19.50185068150811	-99.18394871112798		bloqueo entrada 2
39	Robo a vehículo	Estacionamiento de Alumnos SurEste	2018-02-18	23:37:33	19.502351265046917	-99.18381996509527		robo auto estacionado
42	Agresión	Plaza de Egresados	2018-02-18	17:52:51	19.502530044505463	-99.18647535202001		agresion plaza egresados
44	Fuga de gas	Edificio O	2018-02-18	16:04:47	19.503781495190275	-99.18881980269903		fuga gas en el edificio O

Figura 45. Consulta sobre la tabla de la fecha 28/02/2018.

7.3 Resultados de opiniones

Para comprobar que los resultados del tercer módulo sean correctos, se agregaron 20 opiniones de prueba sobre 4 de los reportes de prueba ya registrados, con esto

al menos se consideran más de una opinión en cada uno de los 4 reportes, el número de opiniones agregados se indican en la siguiente lista:

- No. Reporte 1: 8 opiniones.
- No. Reporte 27: 2 opiniones.
- No. Reporte 38: 4 opiniones.
- No. Reporte 42: 6 opiniones.

Así que se eligen los No. reportes 1, 27, 38 y 42 del mapa, ver Figura 46, y en estos se agregan opiniones de prueba.



Figura 46. Mapa con el total de reportes de prueba registrados.

Una vez agregadas las opiniones, presionamos el segundo botón para mostrar las opiniones agregadas, obteniendo como resultado las imágenes de la figura 47.



Figura 47. Opiniones mostradas en los reportes 1, 27, 38 y 42.

8. Análisis y Discusión de resultados

Para el primer módulo el total de pruebas realizadas fueron *50 reportes registrados*. Para el segundo módulo el total de pruebas realizadas fueron *3 incidentes consultados*. Para el tercer módulo el total de pruebas realizadas fueron *20 opiniones agregadas* y estas mismas *opiniones* fueron *mostradas*.

De las cuales todas las pruebas resultaron precisas.

Módulos	Correctos	Erróneos
Registrar reporte	50	0
Consultar reporte	3	0
Agregar opinión	20	0
Mostrar opiniones	20	0

Tabla 2. Precisión de los módulos.

El motivo por el que las pruebas resultaron 100% efectivas, se debe a que se usaron validaciones para todos los campos de cada formulario. De esta forma el usuario no puede registrar un reporte, con algún campo vacío, consultar cierto criterio o agregar una opinión si no se llenan o seleccionan los campos solicitados. Además, para los resultados, los datos no se obtuvieron de alguna fuente externa, sino que se generaron mediante la misma aplicación, esto sirve de mucho ya que se tiene el tipo de dato o formato adecuado para cada campo de los formularios.

Por ejemplo para la fecha, se tiene el formato especificado por el tipo de dato 'date', para la hora, se tiene el formato por el tipo de dato 'time'. Para estos dos campos ya no se tiene problema en el tipo de dato ya que en estos no se puede alterar el campo y escribir alguna cadena, sino en el formato correspondiente. Para los campos latitud y longitud, estos son del tipo lectura, por lo que el usuario no puede alterar estos números y solo se obtienen al mover el marcador sobre el mapa cuando se está registrando algún reporte. Anteriormente en cuanto a los acentos, se tenía problemas ya que se generaban símbolos extraños, pero se solucionó al cambiar el tipo de codificación de la vista a ISO-8859-1.

Con estas validaciones, tipo de dato o formato adecuado, de cierta manera hizo que los datos no fueran erróneos y siempre fueran correctos a la hora de registrar o consultar datos.

9. Conclusiones

En este proyecto se logró desarrollar una aplicación web que de cierta manera beneficia a la comunidad universitaria, ya que el simple hecho de saber preguntas sencillas como, ¿Cuándo ocurrió?, ¿Dónde ocurrió? y ¿Cómo ocurrió? cierto tipo de incidente que está relacionado a la seguridad de la UAM Azcapotzalco. Con esto se puede mantener informada a toda la comunidad sobre lo que ocurre dentro de la escuela con ayuda de esta aplicación.

Si bien la universidad cuenta con cámaras de seguridad, pero en caso de robo, existen puntos ciegos en el cual no puede ser captado o grabado el incidente

y es ahí donde las personas que cometen este tipo de incidente se aprovechan de la situación para llevar a cabo el acto.

Por lo visto hay incidentes que podrían o han perjudicado a algunos miembros de la comunidad, con esta aplicación lo que se espera es evitar espacios o zonas de la UAM Azcapotzalco, donde suelen ocurrir incidentes que suelen ser frecuentes y puedan afectar de alguna manera a la sociedad. Así la comunidad estará alerta para no pasar por algún incidente que pueda afectarle de cierta manera.

La aplicación web se adapta a cualquier dispositivo sea móvil, tablet, laptop o máquina de escritorio, tiene la capacidad de permitir al usuario registrar uno o varios reportes, consultar incidentes de tres formas, por tipo, por espacio físico y por fecha. Además, permite al usuario emitir una opinión sobre algún reporte registrado, siempre y cuando lo haya sido testigo o vivido dentro de la UAM Azcapotzalco.

Tomando en cuenta el análisis de los resultados, al tener el 100% de efectividad en el registro y obtención de datos de cada uno de los módulos, esto admite confirmar que la aplicación web cumple con el propósito para la que fue desarrollada, es decir, que se consiguió crear un proyecto tecnológico que da a conocer los tipos de incidentes que ocurren y el lugar del suceso.

Uno de los trabajos a futuro que podría hacerse respecto a este proyecto, es implementar las mismas funcionalidades, que realiza este proyecto, pero llevándose a cabo en cada una de las 5 universidades U.A.M.

Una de las mejoras que puede hacerse al sistema, es que el módulo de opiniones, se lleve a cabo dentro del mapa de *Google*, me refiero a que se pueda agregar una opinión dentro de la misma ventana de información que aparece al dar click sobre algún marcador del mapa. Y que además, sobre la misma ventana de información del marcador, se muestren las opiniones agregadas.

Otra de las posibles mejoras que podría hacerse al proyecto, es que los usuarios reciban algún tipo de notificación ya sea a su correo, sobre quienes han opinado al respecto sobre el reporte que dio de alta. Pero supongo que para esto pueden crearse algunas clases para que dentro de la aplicación, se cree pueda crear una nueva cuenta. Y si se quiere recibir notificaciones se necesite ingresar a la aplicación con un *login* y *password*, para así mantener como que la sesión activa.

10. Referencias

[1] Departamento de comunicación social, (2017, Oct). “Encuesta Nacional de Seguridad Pública Urbana (ENSU)”. INEGI. México. [En línea]. Disponible: http://www.inegi.org.mx/saladeprensa/boletines/2017/ensu/ensu2017_10.pdf, [Accedido el: 20- Oct - 2017].

[2] R. A. Ruvalcaba Flores, “Sistema web para gestionar incidentes delictivos”, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2017

[3] E. F. Rosas Coronado, “Sistema de información para el análisis de la seguridad social o pública a través de periódicos”, proyecto terminal, División de

Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2014.

[4] A. López Arteaga, “Sistema web para la geolocalización de incidentes delictivos a partir de publicaciones en *Twitter*”, propuesta de proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2017.

[5] J. A. E Aguilar Rodríguez and V. H. Tapia Cordero, “Red social para la detección de zonas de riesgo y participación ciudadana para dispositivos móviles y computadoras personales”, tesis de ingeniería, Instituto Politécnico Nacional, Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas, México, 2015.

[6] “Semáforo Delictivo”, aplicación móvil. [En línea]. Disponible: <https://play.google.com/store/apps>, [Accedido el: 01-Nov-2017].

[7] “CincoD - Vecino Vigilante”, aplicación móvil. [En línea]. Disponible: <https://play.google.com/store/apps>, [Accedido el: 02-Nov-2017].

[8] Google Developers, (2017, Ago). “Agregar un mapa de Google con un marcador a tu sitio web”. Google Developers. [En línea]. Disponible: <https://developers.google.com/maps/documentation/javascript/adding-a-google-map>, [Accedido el: 05 - Feb - 2018].

[9] Programa En Línea, (2015, Jun). “¿Que es *Java Hibernate*?”. [En línea]. Disponible: <https://programaenlinea.net/que-es-java-hibernate>, [Accedido el: 02 - Abr - 2018].

[10] Tutor de Programación, (2017, Mar). “*Spring MVC* Creación de aplicaciones web”. [En línea]. Disponible: <https://acodigo.blogspot.mx/2017/03/spring-mvc-creacion-de-aplicaciones-web.html>, [Accedido el: 02 - Abr - 2018].

[11] *IBM Knowledge Center*, (2018). “Programación de *JSP* y *servlets Java*”, [En línea]. Disponible: https://www.ibm.com/support/knowledgecenter/es/ssw_ibm_i_73/rzahg/rzahg3aau1.htm, [Accedido el: 02 - Abr - 2018].

[12] Punto Abierto, (2016, Ago). “Qué es *Bootstrap* y cuáles son sus ventajas”, [En línea]. Disponible: <https://puntoabierto.net/blog/que-es-bootstrap-y-cuales-son-sus-ventajas>, [Accedido el: 02 - Abr - 2018].

[13] Facilcloud, (2016, Abr). “*CSS* y *HTML* ¿cuál es la diferencia?”, [En línea]. Disponible: <https://www.facilcloud.com/noticias/css-y-html-cual-es-la-diferencia/>, [Accedido el: 02 - Abr - 2018].

11. Anexos

11.1 Script para la base de datos e inserción de registros

```
-- Crea la base de datos
```

```

CREATE DATABASE IF NOT EXISTS incidentes_uam_azc CHARACTER SET utf8 COLLATE
utf8_general_ci;

-- Usar la base de datos
USE incidentes_uam_azc;

-- Creación de las tablas
CREATE TABLE incidente (
    id_incidente INT AUTO_INCREMENT PRIMARY KEY,
    tipo_incidente VARCHAR(20)
) ENGINE=InnoDB;

CREATE TABLE espacio_fisico (
    id_espacio_fisico INT AUTO_INCREMENT PRIMARY KEY,
    lugar VARCHAR(40)
) ENGINE=InnoDB;

CREATE TABLE reporte (
    id_reporte INT AUTO_INCREMENT PRIMARY KEY,
    usuario_reporte VARCHAR(25),
    fecha DATE,
    hora TIME,
    latitud REAL,
    longitud REAL,
    descripcion VARCHAR(200),

    incidente_id_incidente INT NOT NULL, -- FK
    espacio_fisico_id_espacio_fisico INT NOT NULL, -- FK

    CONSTRAINT reporte_incidente
        FOREIGN KEY (incidente_id_incidente)
        REFERENCES incidente(id_incidente)
        ON DELETE CASCADE ON UPDATE CASCADE,

    CONSTRAINT reporte_espacio
        FOREIGN KEY (espacio_fisico_id_espacio_fisico)
        REFERENCES espacio_fisico(id_espacio_fisico)
        ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;

CREATE TABLE opinion (
    id_opinion INT AUTO_INCREMENT PRIMARY KEY,
    usuario_opinion VARCHAR(25),
    comentario VARCHAR(150),
    valoracion INT,
    fecha_opinion DATE,
    hora_opinion TIME,
    reporte_id_reporte INT NOT NULL, -- FK

    CONSTRAINT opinion_reporte
        FOREIGN KEY (reporte_id_reporte)
        REFERENCES reporte(id_reporte)
        ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;

-- Insertar los tipos de incidentes
INSERT INTO incidentes_uam_azc.incidente (tipo_incidente) VALUES

```

```
('Robo a vehículo'),
('Robo a transeúnte'),
('Robo a instalación'),
('Agresión'),
('Bloqueo'),
('Incendio'),
('Fuga de gas'),
('Fuga de otro químico');
```

```
-- Insertar los espacios físicos de la UAM Azc
```

```
INSERT INTO incidentes_uam_azc.espacio_fisico (lugar) VALUES
```

```
('Edificio B'),
('Edificio C'),
('Edificio D'),
('Edificio E'),
('Edificio F'),
('Edificio G'),
('Edificio G-Bis'),
('Edificio H'),
('Edificio HO'),
('Edificio HP'),
('Edificio I'),
('Edificio J'),
('Edificio K'),
('Edificio L'),
('Edificio M'),
('Edificio O'),
('Edificio P'),
('Edificio P1'),
('Edificio P2'),
('Edificio P3'),
('Edificio P4'),
('Edificio Q'),
('Edificio R'),
('Edificio S'),
('Edificio T'),
('Edificio W'),
('Plaza de Egresados'),
('Plaza Roja'),
('Plaza de Biblioteca'),
('Área Deportiva'),
('Canchas de Basquetball'),
('Cancha de Fútbol Rápido'),
('Estacionamiento de Profesores NorEste'),
('Estacionamiento de Profesores Sur'),
('Estacionamiento de Alumnos SurOeste'),
('Estacionamiento de Alumnos SurEste'),
('Entrada 2'),
('Entrada 4'),
('Entrada 5'),
('Entrada 6'),
('Entrada 7'),
('Kiosko B'),
('Kiosko Canchas'),
('Áreas Verdes B'),
('Áreas Verdes C/D'),
('Áreas Verdes D'),
```

```

('Áreas Verdes E'),
('Áreas Verdes G/I'),
('Áreas Verdes H/HP'),
('Áreas Verdes I/T'),
('Áreas Verdes J/L'),
('Áreas Verdes K'),
('Áreas Verdes P/W'),
('Áreas Verdes P/R'),
('Áreas Verdes R/Canchas Basquet');

```

11.2 Clase *java* para crear conexión a la base de datos

```

public class CrearConexion {

    private static final SessionFactory sessionFactory =
buildSessionFactory();
    private static SessionFactory buildSessionFactory(){
        try {
            // Crea una sesion basado en hibernate.cfg.xml
            return new
Configuration().configure().buildSessionFactory();
        }
        catch (Throwable ex) {
            System.err.println("Error al crear sessionFactory" + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public static void closeSessionFactory(){
        sessionFactory.close();
    }
}

```

11.3 Clase *reporte.java*

```

public class Reporte {

    private int idReporte;
    private String usuarioReporte;
    private String fecha;
    private String hora;
    private String descripcion;
    private double lat;
    private double lng;
    private int idIncidenteFK; //FK
    private int idEspacioFK; //FK

    public int getIdReporte() {
        return idReporte;
    }
    public void setIdReporte(int idReporte) {
        this.idReporte = idReporte;
    }
}

```

```

public String getUsuarioReporte() {
    return usuarioReporte;
}
public void setUsuarioReporte(String usuarioReporte) {
    this.usuarioReporte = usuarioReporte;
}
public String getFecha() {
    return fecha;
}
public void setFecha(String fecha) {
    this.fecha = fecha;
}
public String getHora() {
    return hora;
}
public void setHora(String hora) {
    this.hora = hora;
}
public String getDescripcion() {
    return descripcion;
}
public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
}
public double getLat() {
    return lat;
}
public void setLat(double lat) {
    this.lat = lat;
}
public double getLng() {
    return lng;
}
public void setLng(double lng) {
    this.lng = lng;
}
public int getIdIncidenteFK() {
    return idIncidenteFK;
}
public void setIdIncidenteFK(int idIncidenteFK) {
    this.idIncidenteFK = idIncidenteFK;
}
public int getIdEspacioFK() {
    return idEspacioFK;
}
public void setIdEspacioFK(int idEspacioFK) {
    this.idEspacioFK = idEspacioFK;
}

public String toString(){
    String mensaje = "Id Incidente:" + idIncidenteFK + ",
Usuario:" + usuarioReporte + ", Fecha:" + fecha
+ descripcion + ", Latitud:" + lat + ", Longitud:"
+ lng + ", idEspacio:" + idEspacioFK;
    return mensaje;
}

```

```

        public String toString2(){
            String mensaje = "Id Reporte:" + idReporte + ",
idIncidente:" + idIncidenteFK + ", Usuario:" + usuarioReporte
                                + ", Fecha:" + fecha + ", Hora:" +
hora + ", Descripción:" + descripcion + ", Latitud:"
                                + lat + ", Longitud:" + lng + ",
idEspacio:" + idEspacioFK;
            return mensaje;
        }
    }
}

```

11.4 Mapeo clase reporte

```

<?xml version="1.0" encoding="UTF-8"?>
<hibernate-mapping>
    <class name="uam.incidentes.objetos.Reporte" table="reporte">
        <id name="idReporte" column="id_reporte" type="int">
            <generator class="native"/>
        </id>
        <property name="usuarioReporte" column="usuario_reporte"
type="string"/>
        <property name="fecha" column="fecha" type="string"/>
        <property name="hora" column="hora" type="string"/>
        <property name="lat" column="latitud" type="double"/>
        <property name="lng" column="longitud" type="double"/>
        <property name="descripcion" column="descripcion"
type="string"/>

        <property name="idIncidenteFK"
column="incidente_id_incidente" type="int"/>
        <property name="idEspacioFK"
column="espacio_fisico_id_espacio_fisico" type="int"/>

    </class>
</hibernate-mapping>

```

11.5 Formulario para registrar reporte (simplificado)

```

<tag:form action="registrarReporte" method="POST"
modelAttribute="reporteFrm">
    <tag:hidden path="idReporte" id="idReporte"/>
    <tag:hidden path="idEspacioFK" id="idEspacioFK"/>

    <h4>Tipo de incidente</h4>
    <tag:select          path="idIncidenteFK"          id="idIncidenteFK"
class="form-control">
        <tag:option value="0" >--- Elige un incidente ---</tag:option>
        <tag:option value="1" >Robo a vehículo</tag:option>

```



```

        <tag:option value="2">Robo a transeúnte</tag:option>
        <tag:option value="3">Robo a instalación</tag:option>
        <tag:option value="4">Agresión</tag:option>
        <tag:option value="5">Bloqueo</tag:option>
        <tag:option value="6">Incendio</tag:option>
        <tag:option value="7">Fuga de gas</tag:option>
        <tag:option value="8">Fuga de otro químico</tag:option>
    </tag:select>

    <h4>Fecha</h4>
    <tag:input type="date" path="fecha" id="fecha" required="true"/>

    <h4>Hora</h4>
    <tag:input type="time" path="hora" id="hora" required="true"/>

    <h4>Latitud</h4>
    <tag:input path="lat" id="lat" readonly="true"/>

    <h4>Longitud</h4>
    <tag:input path="lng" id="lng" readonly="true"/>

    <h4>Descripción</h4>
    <tag:textarea path="descripcion" id="descripcion" maxlength="200"
    rows="3" placeholder="Breve descripción del incidente (Máx. 200
    caracteres)"/>

    <h4>Nombre de la persona que reporta</h4>
    <tag:input path="usuarioReporte" id="usuarioReporte"
    maxlength="25" placeholder="Puede colocar 'Anónimo'"/>
    <br><br>
    <center><tag:button>Registrar incidente</tag:button></center>
</tag:form>

```

11.6 Controlador registrar reporte

@Controller

```

public class ControladorRegistrar {
    @RequestMapping("registro_incidente")
    public ModelAndView cargarFormReporte(){
        //System.out.println("Mapeando datos de reporte en la
        forma");
    }
}

```

```

        ModelAndView modelo = new
ModelAndView("/registro_incidente_vista");
        Reporte reporte = new Reporte();
        modelo.addObject("reporteFrm", reporte);
        return modelo;
    }

    @RequestMapping(value="registrarReporte",
method=RequestMethod.POST)
    public ModelAndView
registrarReporte(@ModelAttribute("reporteFrm")Reporte reporte){

        System.out.println("DATOS COLOCADOS: " + reporte.toString());

        RegistroReporte reg = new RegistroReporte();
        reg.insertarReporte(reporte);
        return new ModelAndView("/index");
    }
}

```

11.7 Insertar reporte

```

public class RegistroReporte {

    public void insertarReporte(Reporte reporte){
        SessionFactory sf = CrearConexion.getSessionFactory();
        Session sesion=sf.openSession();
        try{

            sesion.beginTransaction();
            sesion.save(reporte);
            sesion.getTransaction().commit();

            System.out.println("REPORTE REGISTRADO!!");
            System.out.println("DATOS REGISTRADOS: " +
reporte.toString2()+"\n");

        } catch(Throwable ex){
            System.out.println("No se pudo abrir la conexion");
            throw new ExceptionInInitializerError(ex);
        }finally{
            sesion.close();
        }
    }
}

```

11.8 Clase ReporteIncidenteEspacio.java

```

public class ReporteIncidenteEspacio {

    private int reporIdReporte;
    private int reporIdIncidente;
    private int reporIdEspacio;
    private String incidTipoIndidente;
    private String espacLugar;
    private String reporFecha;
}

```

```

private String reporHora;
private double reporLatitud;
private double reporLongitud;
private String reporDescripcion;

public int getReporIdReporte() {
    return reporIdReporte;
}
public void setReporIdReporte(int reporIdReporte) {
    this.reporIdReporte = reporIdReporte;
}
public int getReporIdIncidente() {
    return reporIdIncidente;
}
public void setReporIdIncidente(int reporIdIncidente) {
    this.reporIdIncidente = reporIdIncidente;
}
public int getReporIdEspacio() {
    return reporIdEspacio;
}
public void setReporIdEspacio(int reporIdEspacio) {
    this.reporIdEspacio = reporIdEspacio;
}
public String getIncidTipoIndidente() {
    return incidTipoIndidente;
}
public void setIncidTipoIndidente(String incidTipoIndidente) {
    this.incidTipoIndidente = incidTipoIndidente;
}
public String getEspacLugar() {
    return espacLugar;
}
public void setEspacLugar(String espacLugar) {
    this.espacLugar = espacLugar;
}
public String getReporFecha() {
    return reporFecha;
}
public void setReporFecha(String reporFecha) {
    this.reporFecha = reporFecha;
}
public String getReporHora() {
    return reporHora;
}
public void setReporHora(String reporHora) {
    this.reporHora = reporHora;
}
public double getReporLatitud() {
    return reporLatitud;
}
public void setReporLatitud(double reporLatitud) {
    this.reporLatitud = reporLatitud;
}
public double getReporLongitud() {
    return reporLongitud;
}
public void setReporLongitud(double reporLongitud) {

```

```

        this.reporLongitud = reporLongitud;
    }
    public String getReporDescripcion() {
        return reporDescripcion;
    }
    public void setReporDescripcion(String reporDescripcion) {
        this.reporDescripcion = reporDescripcion;
    }

    public String toStringIncidente(){
        String mensaje = "Id Reporte:" + reporIdReporte + ", Tipo
incidente:" + incidTipoIndidente + ", Espacio físico:" + espacLugar
        + ", Fecha:" + reporFecha + ", Hora:"
+ reporHora + ", Latitud:" + reporLatitud + ", Longitud:" + reporLongitud
+ ", Descripción:" + reporDescripcion;
        return mensaje;
    }

    public String toStringEspacio(){
        String mensaje = "Id Reporte:" + reporIdReporte + ", Espacio
físico:" + espacLugar + ", Tipo incidente:" + incidTipoIndidente
        + ", Fecha:" + reporFecha + ", Hora:"
+ reporHora + ", Latitud:" + reporLatitud + ", Longitud:" + reporLongitud
+ ", Descripción:" + reporDescripcion;
        return mensaje;
    }

    public String toStringFecha(){
        String mensaje = "Id Reporte:" + reporIdReporte + ", Fecha:"
+ reporFecha + ", Tipo incidente:" + incidTipoIndidente + ", Espacio
físico:" + espacLugar
        + ", Hora:" + reporHora + ",
Latitud:" + reporLatitud + ", Longitud:" + reporLongitud + ",
Descripción:" + reporDescripcion;
        return mensaje;
    }
}

```

11.9 Controlador consultar incidentes

```

@Controller
public class ControladorConsultar {

    @RequestMapping("consulta_incidente")
    public ModelAndView mostrarFrmIncidente(){
        ModelAndView modelo = new
ModelAndView("/consulta_incidente_vista");
        ReporteIncidenteEspacio rep = new ReporteIncidenteEspacio();
        modelo.addObject("IncidenteFrm", rep);
        return modelo;
    }

    @RequestMapping(value="consultarIncidente",
method=RequestMethod.POST)
    public ModelAndView
mostrarConsulta (@ModelAttribute("IncidenteFrm")ReporteIncidenteEspacio
rep){

```

```

        System.out.println("Tipo de incidente seleccionado:
"+rep.getReporIdIncidente());

        ConsultaReporte sel = new ConsultaReporte();
        List<ReporteIncidenteEspacio>lista=sel.reporteIncid(rep); //

        if(lista.size()!=0){
            System.out.println("CONSULTA POR INCIDENTE: ");
            for (int i = 0; i < lista.size(); i++) {
                ReporteIncidenteEspacio repoInci = lista.get(i);
                System.out.println(repoInci.toStringIncidente());

            }
            System.out.println("\n");
        } else {
            System.out.println("No hay datos del id incidente:
"+rep.getReporIdIncidente());
        }

        return new ModelAndView("/consulta_incidente_vista");
    }

    @RequestMapping("consulta_espacio")
    public ModelAndView mostrarFrmEspacio(){
        ModelAndView modeloEspacio = new
        ModelAndView("/consulta_espacio_vista");
        ReporteIncidenteEspacio repEspacio = new
        ReporteIncidenteEspacio();
        modeloEspacio.addObject("EspacioFrm", repEspacio);
        return modeloEspacio;
    }

    @RequestMapping(value="consultarEspacio",
method=RequestMethod.POST)
    public ModelAndView
mostrarConsultaEspacio(@ModelAttribute("EspacioFrm")ReporteIncidenteEspacio
io repEspacio){
        System.out.println("Tipo de espacio seleccionado:
"+repEspacio.getReporIdEspacio());

        ConsultaReporte selEspacio = new ConsultaReporte();
        List<ReporteIncidenteEspacio>listaEspacio =
selEspacio.reporteEspac(repEspacio); //
        if(listaEspacio.size()!=0){
            System.out.println("CONSULTA POR ESPACIO FÍSICO: ");
            for (int i = 0; i < listaEspacio.size(); i++) {
                ReporteIncidenteEspacio repoEspac =
listaEspacio.get(i);
                System.out.println(repoEspac.toStringEspacio());
            }
            System.out.println("\n");
        } else {
            System.out.println("No hay datos del id espacio:
"+repEspacio.getReporIdEspacio());
        }
        return new ModelAndView("/consulta_espacio_vista");
    }
}

```

```

    @RequestMapping("consulta_fecha")
    public ModelAndView mostrarFrmFecha(){
        ModelAndView modeloFecha = new
ModelAndView("/consulta_fecha_vista");
        ReporteIncidenteEspacio repFecha = new
ReporteIncidenteEspacio();
        modeloFecha.addObject("FechaFrm", repFecha);
        return modeloFecha;
    }
    @RequestMapping(value="consultarFecha", method=RequestMethod.POST)
    public ModelAndView
mostrarConsultaFecha(@ModelAttribute("FechaFrm")ReporteIncidenteEspacio
repFecha){
        System.out.println("Fecha seleccionada:
"+repFecha.getReporFecha());

        ConsultaReporte selFecha = new ConsultaReporte();
        List<ReporteIncidenteEspacio>listaFecha =
selFecha.reporteFecha(repFecha); //
        if(listaFecha.size()!=0){
            System.out.println("CONSULTA POR FECHA: ");
            for (int i = 0; i < listaFecha.size(); i++) {
                ReporteIncidenteEspacio repoFecha =
listaFecha.get(i);
                System.out.println(repoFecha.toStringFecha());
            }
            System.out.println("\n");
        } else {
            System.out.println("No hay datos de la fecha:
"+repFecha.getReporFecha());
        }
        return new ModelAndView("/consulta_fecha_vista");
    }
}

```

11.10 Consultar reporte

```

public class ConsultaReporte {

//SELECCIONAR ID REPORTE, LUGAR, FECHA, HORA, LATITUD, LONGITUD Y
DESCRIPCION DE ALGUN TIPO DE INCIDENTE
public List<ReporteIncidenteEspacio> reporteIncid(ReporteIncidenteEspacio
rep){

        List<Object[]>lista;
        List<ReporteIncidenteEspacio>listaReporteInc = new
LinkedList<>();
        Session sesion = null;

        try{
            sesion=CrearConexion.getSessionFactory().openSession();
        } catch (ExceptionInInitializerError ex) {
            System.err.println("No se pudo crear la session");
            throw new ExceptionInInitializerError(ex);
        }
    }
}

```

```

    }
    /*Consulta en MySQL
    SELECT reporte.id_reporte, incidente.tipo_incidente,
espacio_fisico.lugar, reporte.fecha, reporte.hora, reporte.latitud,
reporte.longitud, reporte.descripcion
    FROM reporte, incidente, espacio_fisico
    WHERE reporte.incidente_id_incidente = incidente.id_incidente
    AND reporte.espacio_fisico_id_espacio_fisico =
espacio_fisico.id_espacio_fisico
    AND reporte.incidente_id_incidente='1-8' ORDER BY
reporte.id_reporte;*/

    String sentencia = "SELECT REP.idReporte, INC.tipoIncidente,
ESP.lugar, REP.fecha, REP.hora, REP.lat, REP.lng, REP.descripcion "
        + "FROM Reporte as REP, Incidente as INC,
EspacioFisico as ESP "
        + "WHERE REP.idIncidenteFK = INC.idIncidente "
        + "AND REP.idEspacioFK = ESP.idEspacioFisico "
        + "AND REP.idIncidenteFK =
'" + rep.getReporIdIncidente() + "' ORDER BY REP.idReporte";
    Query query = sesion.createQuery(sentencia);
    lista = query.list();

    for(int i=0; i<lista.size();i++){
        Object[] datosRecuperados = lista.get(i);
        ReporteIncidenteEspacio reporInc = new
ReporteIncidenteEspacio();
        reporInc.setReporIdReporte((int)datosRecuperados[0]);

        reporInc.setIncidTipoIndidente((String)datosRecuperados[1]);
        reporInc.setEspacLugar((String)datosRecuperados[2]);
        reporInc.setReporFecha((String)datosRecuperados[3]);
        reporInc.setReporHora((String)datosRecuperados[4]);
        reporInc.setReporLatitud((double)datosRecuperados[5]);
        reporInc.setReporLongitud((double)datosRecuperados[6]);

        reporInc.setReporDescripcion((String)datosRecuperados[7]);
        listaReporteInc.add(reporInc);
    }
    return listaReporteInc;
}

//SELECCIONAR ID REPORTE, TIPO DE INCIDENTE, FECHA, HORA, LATITUD,
LONGITUD Y DESCRIPCION DE ALGUN ESPACIO FISICO
public List<ReporteIncidenteEspacio> reporteEspac(ReporteIncidenteEspacio
repEspacio){

    List<Object[]>listaEspacio;
    List<ReporteIncidenteEspacio>listaReporteEsp = new
LinkedList<>();
    Session sesion = null;

    try{
        sesion=CrearConexion.getSessionFactory().openSession();
    } catch (ExceptionInInitializerError ex) {
        System.err.println("No se pudo crear la session");
        throw new ExceptionInInitializerError(ex);
    }
}

```

```

    }
    /*Consulta en MySQL
    SELECT reporte.id_reporte, incidente.tipo_incidente,
espacio_fisico.lugar, reporte.fecha, reporte.hora, reporte.latitud,
reporte.longitud, reporte.descripcion
    FROM reporte, incidente, espacio_fisico
    WHERE reporte.incidente_id_incidente = incidente.id_incidente
    AND reporte.espacio_fisico_id_espacio_fisico =
espacio_fisico.id_espacio_fisico
    AND reporte.espacio_fisico_id_espacio_fisico='1-55' ORDER BY
reporte.id_reporte;*/

```

```

    String sentencia = "SELECT REP.idReporte, INC.tipoIncidente,
ESP.lugar, REP.fecha, REP.hora, REP.lat, REP.lng, REP.descripcion "
        + "FROM Reporte as REP, Incidente as INC,
EspacioFisico as ESP "
        + "WHERE REP.idIncidenteFK = INC.idIncidente "
        + "AND REP.idEspacioFK = ESP.idEspacioFisico "
        + "AND REP.idEspacioFK =

```

```

    '"+repEspacio.getReporIdEspacio()+"' ORDER BY REP.idReporte";

```

```

    Query query = sesion.createQuery(sentencia);

```

```

    listaEspacio = query.list();

```

```

    for(int i=0; i<listaEspacio.size();i++){

```

```

        Object[] datosRecuperados = listaEspacio.get(i);

```

```

        ReporteIncidenteEspacio reporEsp = new

```

```

ReporteIncidenteEspacio();

```

```

        reporEsp.setReporIdReporte((int)datosRecuperados[0]);

```

```

        reporEsp.setIncidTipoIndidente((String)datosRecuperados[1]);

```

```

        reporEsp.setEspacLugar((String)datosRecuperados[2]);

```

```

        reporEsp.setReporFecha((String)datosRecuperados[3]);

```

```

        reporEsp.setReporHora((String)datosRecuperados[4]);

```

```

        reporEsp.setReporLatitud((double)datosRecuperados[5]);

```

```

        reporEsp.setReporLongitud((double)datosRecuperados[6]);

```

```

        reporEsp.setReporDescripcion((String)datosRecuperados[7]);

```

```

        listaReporteEsp.add(reporEsp);

```

```

    }

```

```

    return listaReporteEsp;

```

```

}

```

```

//SELECCIONAR TODOS LOS REPORTES REGISTRADOS, MENOS EL USUARIO QUE
REPORTÓ

```

```

public List<ReporteIncidenteEspacio>

```

```

reporteIncEspac(ReporteIncidenteEspacio repIncEspacio){

```

```

    List<Object[]>listaIncEspacio;

```

```

    List<ReporteIncidenteEspacio>listaReporteIncEsp = new

```

```

LinkedList<>();

```

```

    Session sesion = null;

```

```

    try{

```

```

        sesion=CrearConexion.getSessionFactory().openSession();

```

```

    } catch (ExceptionInInitializerError ex) {

```

```

        System.err.println("No se pudo crear la session");

```

```

        throw new ExceptionInInitializerError(ex);
    }

```



```

    }
    /*Consulta en MySQL
    SELECT reporte.id_reporte, incidente.tipo_incidente,
espacio_fisico.lugar, reporte.fecha, reporte.hora, reporte.latitud,
reporte.longitud, reporte.descripcion
    FROM reporte, incidente, espacio_fisico
    WHERE reporte.incidente_id_incidente = incidente.id_incidente
    AND reporte.espacio_fisico_id_espacio_fisico =
espacio_fisico.id_espacio_fisico
    ORDER BY reporte.id_reporte;*/

    String sentencia = "SELECT REP.idReporte, INC.tipoIncidente,
ESP.lugar, REP.fecha, REP.hora, REP.lat, REP.lng, REP.descripcion "
        + "FROM Reporte as REP, Incidente as INC, EspacioFisico
as ESP "
        + "WHERE REP.idIncidenteFK = INC.idIncidente "
        + "AND REP.idEspacioFK = ESP.idEspacioFisico "
        + "ORDER BY REP.idReporte";
    Query query = sesion.createQuery(sentencia);
    listaIncEspacio = query.list();

    for(int i=0; i<listaIncEspacio.size();i++){
        Object[] datosRecuperados = listaIncEspacio.get(i);
        ReporteIncidenteEspacio reporIncEsp = new
ReporteIncidenteEspacio();
        reporIncEsp.setReporIdReporte((int)datosRecuperados[0]);

        reporIncEsp.setIncidTipoIndidente((String)datosRecuperados[1]);
        reporIncEsp.setEspacLugar((String)datosRecuperados[2]);
        reporIncEsp.setReporFecha((String)datosRecuperados[3]);
        reporIncEsp.setReporHora((String)datosRecuperados[4]);
        reporIncEsp.setReporLatitud((double)datosRecuperados[5]);
        reporIncEsp.setReporLongitud((double)datosRecuperados[6]);
        reporIncEsp.setReporDescripcion((String)datosRecuperados[7]);
        listaReporteIncEsp.add(reporIncEsp);
    }
    return listaReporteIncEsp;
}

//SELECCIONAR ID REPORTE, TIPO DE INCIDENTE, LUGAR, HORA, LATITUD,
LONGITUD Y DESCRIPCION DE CIERTA FECHA
public List<ReporteIncidenteEspacio> reporteFecha(ReporteIncidenteEspacio
repFecha){

    List<Object[]>listaFecha;
    List<ReporteIncidenteEspacio>listaReporteFecha = new
LinkedList<>();
    Session sesion = null;

    try{
        sesion=CrearConexion.getSessionFactory().openSession();
    } catch (ExceptionInInitializerError ex) {
        System.err.println("No se pudo crear la session");
        throw new ExceptionInInitializerError(ex);
    }
    /*Consulta en MySQL

```

```

        SELECT reporte.id_reporte, incidente.tipo_incidente,
espacio_fisico.lugar, reporte.fecha, reporte.hora, reporte.latitud,
reporte.longitud, reporte.descripcion
        FROM reporte, incidente, espacio_fisico
        WHERE reporte.incidente_id_incidente = incidente.id_incidente
        AND reporte.espacio_fisico_id_espacio_fisico =
espacio_fisico.id_espacio_fisico
        AND reporte.fecha = 'AAAA-MM-DD' ORDER BY reporte.id_reporte;*/

String sentencia = "SELECT REP.idReporte, INC.tipoIncidente,
ESP.lugar, REP.fecha, REP.hora, REP.lat, REP.lng, REP.descripcion "
        + "FROM Reporte as REP, Incidente as INC, EspacioFisico
as ESP "
        + "WHERE REP.idIncidenteFK = INC.idIncidente "
        + "AND REP.idEspacioFK = ESP.idEspacioFisico "
        + "AND REP.fecha = '"+repFecha.getReporFecha()+"' ORDER
BY REP.idReporte";
Query query = sesion.createQuery(sentencia);
listaFecha = query.list();

for(int i=0; i<listaFecha.size();i++){
    Object[] datosRecuperados = listaFecha.get(i);
    ReporteIncidenteEspacio reporFech = new
ReporteIncidenteEspacio();
    reporFech.setReporIdReporte((int)datosRecuperados[0]);
    reporFech.setIncidTipoIndidente((String)datosRecuperados[1]);
    reporFech.setEspacLugar((String)datosRecuperados[2]);
    reporFech.setReporFecha((String)datosRecuperados[3]);
    reporFech.setReporHora((String)datosRecuperados[4]);
    reporFech.setReporLatitud((double)datosRecuperados[5]);
    reporFech.setReporLongitud((double)datosRecuperados[6]);
    reporFech.setReporDescripcion((String)datosRecuperados[7]);
    listaReporteFecha.add(reporFech);
}
return listaReporteFecha;
}
}
}

```

11.11 Clase opinion.java

```

public class Opinion {

    private int idOpinion;
    private String usuarioOpinion;
    private String comentario;
    private int valoracion;
    private String fechaOpinion;
    private String horaOpinion;

    private int idReporteFK;

    public int getIdOpinion() {
        return idOpinion;
    }
    public void setIdOpinion(int idOpinion) {
        this.idOpinion = idOpinion;
    }
}

```

```

public String getUsuarioOpinion() {
    return usuarioOpinion;
}
public void setUsuarioOpinion(String usuarioOpinion) {
    this.usuarioOpinion = usuarioOpinion;
}
public String getComentario() {
    return comentario;
}
public void setComentario(String comentario) {
    this.comentario = comentario;
}
public int getValoracion() {
    return valoracion;
}
public void setValoracion(int valoracion) {
    this.valoracion = valoracion;
}

public String getFechaOpinion() {
    return fechaOpinion;
}
public void setFechaOpinion(String fechaOpinion) {
    this.fechaOpinion = fechaOpinion;
}
public String getHoraOpinion() {
    return horaOpinion;
}
public void setHoraOpinion(String horaOpinion) {
    this.horaOpinion = horaOpinion;
}
public int getIdReporteFK() {
    return idReporteFK;
}
public void setIdReporteFK(int idReporteFK) {
    this.idReporteFK = idReporteFK;
}

public String toString(){
    String mensaje = "Id Reporte:" + idReporteFK + ", Usuario:" +
usuarioOpinion + ", Comentario:" + comentario + ", Valoración:" +
valoracion + ", Fecha Opinión:" + fechaOpinion + ", Hora Opinión:" +
horaOpinion;
    return mensaje;
}
public String toString2(){
    String mensaje = "Usuario:" + usuarioOpinion + ",
Comentario:" + comentario + ", Valoración:" + valoracion + ", Fecha
Opinión:" + fechaOpinion + ", Hora Opinión:" + horaOpinion;
    return mensaje;
}
}
}

```

11.12 Mapeo clase opinion

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<hibernate-mapping>
  <class name="uam.incidentes.objetos.Opinion" table="opinion">
    <id name="idOpinion" column="id_opinion" type="int">
      <generator class="native"/>
    </id>
    <property name="usuarioOpinion" column="usuario_opinion"
type="string"/>
    <property name="comentario" column="comentario"
type="string"/>
    <property name="valoracion" column="valoracion" type="int"/>
    <property name="fechaOpinion" column="fecha_opinion"
type="string"/>
    <property name="horaOpinion" column="hora_opinion"
type="string"/>

    <property name="idReporteFK" column="reporte_id_reporte"
type="int"/>
  </class>
</hibernate-mapping>

```

11.13 Formulario para agregar opinión (simplificado)

```

<tag:form action="registrarOpinion" method="POST" modelAttribute =
"OpinionFrm">
  <tag:hidden path="idOpinion" id="idOpinion"/>
  <h4>No. Reporte </h4>
  <tag:input path="idReporteFK" id="idReporteFK" readonly="true"/>

  <h4>Nombre</h4>
  <tag:input path="usuarioOpinion" id="usuarioOpinion" maxlength="25"
placeholder="Puede colocar 'Anónimo'" required="true"/>

  <h4>Comentario</h4>
  <tag:textarea path="comentario" id="comentario" maxlength="150"
rows="2" placeholder="Breve comentario (Máx. 150 caracteres)"
required="true"/>

  <h4>Clasificación</h4>
  <div class="starrrr"></div>
  <div>
    <span class="nada" style="display: none;">
    <span class="choice1"></span>: Nada relevante. </span>

    <span class="poco" style="display: none;">
    <span class="choice2"></span>: Poco relevante. </span>

    <span class="algo" style="display: none;">
    <span class="choice3"></span>: Algo relevante. </span>

    <span class="bastante" style="display: none;">
    <span class="choice4"></span>: Bastante relevante. </span>

    <span class="muy" style="display: none;">
    <span class="choice5"></span>: Muy relevante. </span>
  </div>

```

```

<tag:hidden path="valoracion" id="valoracion"/>
  <script>
    $(".starrrr").starrrr({
      rating: 0,
      change: function(e, valor){
        if (valor==1) {
          $(".nada").show();
          $(".poco").hide();
          $(".algo").hide();
          $(".bastante").hide();
          $(".muy").hide();
          $('.choice1').text(valor);
        }
        else if (valor==2) {
          $(".nada").hide();
          $(".poco").show();
          $(".algo").hide();
          $(".bastante").hide();
          $(".muy").hide();
          $('.choice2').text(valor);
        }
        else if (valor==3) {
          $(".nada").hide();
          $(".poco").hide();
          $(".algo").show();
          $(".bastante").hide();
          $(".muy").hide();
          $('.choice3').text(valor);
        }
        else if (valor==4) {
          $(".nada").hide();
          $(".poco").hide();
          $(".algo").hide();
          $(".bastante").show();
          $(".muy").hide();
          $('.choice4').text(valor);
        }
        else if (valor==5) {
          $(".nada").hide();
          $(".poco").hide();
          $(".algo").hide();
          $(".bastante").hide();
          $(".muy").show();
          $('.choice5').text(valor);
        }
        else {
          valor = 0;
          $(".nada").hide();
          $(".poco").hide();
          $(".algo").hide();
          $(".bastante").hide();
          $(".muy").hide();
        }
        $("#valoracion").attr("value",valor);
      }
    });
  </script>

```

```

    <h4>Fecha</h4>
    <tag:input type="date" path="fechaOpinion" id="fechaOpinion"
readonly="true"/>

    <h4>Hora</h4>
    <tag:input type="time" path="horaOpinion" id="horaOpinion"
readonly="true"/>
    <br>
    <center><tag:button>Agregar opinión</tag:button></center>
</tag:form>

```

11.14 Controlador agregar/mostrar opinión

@Controller

```

public class ControladorOpinion {
    //Total de reportes
    @RequestMapping("total_reportes")
    public ModelAndView mostrarOpinion(){
        ModelAndView modeloOpinion= new
ModelAndView("/consulta_inc_esp_vista");
        Opinion op = new Opinion();
        modeloOpinion.addObject("OpinionFrm", op);
        return modeloOpinion;
    }
    //Registrar opinión
    @RequestMapping(value="registrarOpinion",
method=RequestMethod.POST)
    public ModelAndView
registrarOpinion(@ModelAttribute("OpinionFrm")Opinion op){

        System.out.println("OPINIONES COLOCADAS: " + op.toString());

        OpinionReporte regOp = new OpinionReporte();
        regOp.insertarOpinion(op);
        return new ModelAndView("/index");
    }

    //Mostrar opiniones
    @RequestMapping("opiniones")
    public ModelAndView OpinionesRegistradas(){
        ModelAndView modeloOpinion= new
ModelAndView("/opiniones_agregadas");
        Opinion op = new Opinion();
        modeloOpinion.addObject("mostrarOpinionFrm", op);
        return modeloOpinion;
    }
    @RequestMapping(value="mostrarOpinion", method=RequestMethod.POST)
    public ModelAndView
visualizarOpiniones(@ModelAttribute("mostrarOpinionFrm")Opinion op){

        System.out.println("Id Reporte seleccionado:
"+op.getIdReporteFK());

        ObtenerOpiniones selOpinion = new ObtenerOpiniones();

```

```

        List<Opinion>listaOpinion =
selOpinion.opinionValoracion(op);

        if(listaOpinion.size()!=0){
            System.out.println("MOSTRAR OPINIONES: ");
            for (int i = 0; i < listaOpinion.size(); i++) {
                Opinion opin = listaOpinion.get(i);
                System.out.println(opin.toString2());
            }
            System.out.println("\n");
        } else {
            System.out.println("No hay opiniones del id:
"+op.getIdReporteFK());
        }

        return new ModelAndView("/opiniones_agregadas");
    }
}

```

11.15 Insertar opinión

```

public class OpinionReporte {
    public void insertarOpinion(Opinion opinion){
        SessionFactory sf = CrearConexion.getSessionFactory();
        Session sesion=sf.openSession();
        try{
            sesion.beginTransaction();
            sesion.save(opinion);
            sesion.getTransaction().commit();

            System.out.println("OPINIONES REGISTRADAS!");
            System.out.println("OPINIONES: " +
opinion.toString()+"\n");
        } catch(Throwable ex){
            System.out.println("No se pudo abrir la conexion");
            throw new ExceptionInInitializerError(ex);
        }finally{
            sesion.close();
        }
    }
}

```

11.16 Obtener opiniones

```

public class ObtenerOpiniones {
    public List<Opinion> opinionValoracion(Opinion op){

        List<Object[]>listaOpinion;
        List<Opinion>listaReporteOpinion = new LinkedList<>();
        Session sesion = null;

        try{
            sesion=CrearConexion.getSessionFactory().openSession();
        } catch (ExceptionInInitializerError ex) {

```

```

        System.err.println("No se pudo crear la session");
        throw new ExceptionInInitializerError(ex);
    }
    /*-- SELECCIONAR USUARIO, COMENTARIO, VALORACION, FECHA, HORA
DE LAS OPINIONES AGREGADAS DE CIERTO ID_REPORTE
    SELECT opinion.usuario_opinion, opinion.comentario,
opinion.valoracion, opinion.fecha_opinion, opinion.hora_opinion
    FROM opinion WHERE opinion.reporte_id_reporte = 'n' ORDER BY
opinion.id_opinion;*/

    String sentencia = "SELECT OP.usuarioOpinion, OP.comentario,
OP.valoracion, OP.fechaOpinion, OP.horaOpinion "
        + "FROM Opinion as OP WHERE OP.idReporteFK =
'" + op.getIdReporteFK() + "' "
        + "ORDER BY OP.idOpinion";
    Query query = sesion.createQuery(sentencia);
    listaOpinion = query.list();

    for(int i=0; i<listaOpinion.size();i++){
        Object[] datosRecuperados = listaOpinion.get(i);
        Opinion reporOp = new Opinion();
        reporOp.setUsuarioOpinion((String)datosRecuperados[0]);
        reporOp.setComentario((String)datosRecuperados[1]);
        reporOp.setValoracion((int)datosRecuperados[2]);
        reporOp.setFechaOpinion((String)datosRecuperados[3]);
        reporOp.setHoraOpinion((String)datosRecuperados[4]);

        listaReporteOpinion.add(reporOp);
    }
    return listaReporteOpinion;
}
}
}

```

11.17 Clase vistasReferencias.java

```

@Controller

public class vistasReferencias {
    @RequestMapping("home") //Nombre de la liga que va a atrapar este metodo
    //Crear metodo q atrape la peticion
    public ModelAndView clickHome(){ //ModelAndView es un objeto de spring
        para la vista
        return new ModelAndView("/index"); //retornara /jsp/index.jsp
    }

    @RequestMapping("about")
    public ModelAndView clickAcerca(){
        return new ModelAndView("/acerca");
    }

    @RequestMapping("project")
    public ModelAndView clickProyecto(){
        return new ModelAndView("/proyecto_terminal");
    }

    @RequestMapping("contact")
    public ModelAndView clickContacto(){

```



```

        return new ModelAndView("/contacto");
    }
}

```

11.18 Mapeo clase incidente

```

<?xml version="1.0" encoding="UTF-8"?>
<hibernate-mapping>
    <class name="uam.incidentes.objetos.Incidente" table="incidente">
        <id name="idIncidente" column="id_incidente" type="int">
            <generator class="native"/>
        </id>
        <property name="tipoIncidente" column="tipo_incidente"
type="string"/>
    </class>
</hibernate-mapping>

```

11.19 Mapeo clase espacio físico

```

<?xml version="1.0" encoding="UTF-8"?>
<hibernate-mapping>
    <class name="uam.incidentes.objetos.EspacioFisico"
table="espacio_fisico">
        <id name="idEspacioFisico" column="id_espacio_fisico"
type="int">
            <generator class="native"/>
        </id>
        <property name="lugar" column="lugar" type="string"/>
    </class>
</hibernate-mapping>

```

11.20 Clase incidente.java

```

public class Incidente {

    private String idIncidente;
    private String tipoIncidente;

    public String getIdIncidente() {
        return idIncidente;
    }
    public void setIdIncidente(String idIncidente) {
        this.idIncidente = idIncidente;
    }
    public String getTipoIncidente() {
        return tipoIncidente;
    }
    public void setTipoIncidente(String tipoIncidente) {
        this.tipoIncidente = tipoIncidente;
    }

    public String toString(){
        String mensaje = idIncidente +", "+tipoIncidente;
    }
}

```

```
        return mensaje;
    }
}
```

11.21 Clase espaciofisico.java

```
public class EspacioFisico {
    private int idEspacioFisico;
    private String lugar;

    public int getIdEspacioFisico() {
        return idEspacioFisico;
    }
    public void setIdEspacioFisico(int idEspacioFisico) {
        this.idEspacioFisico = idEspacioFisico;
    }
    public String getLugar() {
        return lugar;
    }
    public void setLugar(String lugar) {
        this.lugar = lugar;
    }
}
```