Instituto Politécnico Nacional

Escuela Superior de Cómputo

Web App Development.

PRACTICA 1: EVENTOS SERVLET

Profesor: M. en C. José Asunción Enríquez Zárate

Alumno: Atziri Pérez García

atziripg.99@gmail.com

3CM4

November 9, 2020

Contents

1 Introducción **2**

2 Conceptos **3**
2.1 Apache Tomcat . . . 3
2.2 JDBC . . . 3
2.2.1 Driver JDBC . . . 3
2.2.2 Paquete . . . 3
2.3 Servlets . . . 4
2.3.1 Ciclo de vida . . . 4
2.4 Bootstrap . . . 4

3 Desarrollo **5**
3.0.1 listaDeEventos . . . 10
3.0.2 agregarEvento . . . 10
3.0.3 eliminarEvento . . . 10
3.0.4 almacenarEvento . . . 10
3.0.5 actualizat y mostrar Evento . . . 11

4 Resultados **12**

5 Conclusión **15**

6 Referencias Bibliográficas **16**

List of Figures

1 Tomcat funcionando 5
2 Lista De Eventos 12
3 Agregar Evento 13
4 Actualizar evento 13
5 Ver evento 14
6 Lista de Eventos con modificaciones 14

1 Introducción

La presente práctica tiene como objetivo realizar un proyecto que donde se hagan las acciónes básicas CRUD (Create, Retrieve, Update y Delete). Se pondrá en práctica el uso de muchas herramientas para la realización de páginas web como el servidor de aplicación Apache Tomcat, el API JDBC para crear conexión a base de datos, uso se servlets para manejar la funcionalidad de la página web, HTML para el diseño, así como frameworks para mejorar la presentación de la Web GUI como lo es Bootstrap (Front End) y validaciones en los forms. El principal uso del proyecto es administrar los eventos realizados en la ESCOM, en este proyecto se realizan las funciones básicas:

- Mostar eventos
- Agregar un evento nuevo
- Eliminar evento
- Actulizar evento
- Ver datos de un evento en específico

Cada acción tiene una interfaz diferente para un mejor y más fácil manejo del usuario.

2 Conceptos

Como se mencióno, se hizo uso de varias tecnologías para realizar el proyecto, las cuales explicaremos más a fondo

2.1 Apache Tomcat

El software Apache Tomcat es una implementación de código abierto de las tecnologías Java Servlet, JavaServer Pages, Java Expression Language y Java WebSocket. Las especificaciones de Java Servlet, JavaServer Pages, Java Expression Language y Java WebSocket se desarrollan bajo el Proceso de la comunidad Java. Funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Oracle Corporation. El software Apache Tomcat impulsa numerosas aplicaciones web de misión crítica a gran escala en una amplia gama de industrias y organizaciones. Algunos de estos usuarios y sus historias se enumeran en la página wiki de PoweredBy.

Apache Tomcat lo usamos como servidor de aplicación, en donde estará alojado nuestro proyecto.

2.2 JDBC

Java Database Connectivity (JDBC) es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la biblioteca de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión; para ello provee el localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar cualquier tipo de tarea con la base de datos a la que tenga permiso: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

2.2.1 Driver JDBC

Los drivers JDBC son adaptadores del lado del cliente (instalados en la máquina cliente, no en el servidor) que convierten la petición proveniente del programa JAVA a un protocolo que el SGBD pueda entender.

- Driver JDBC Tipo 1 (también llamado Puente JDBC-ODBC) convierte el método JDBC a una llamada a una función ODBC. Utiliza los drivers ODBC para conectar con la base de datos.
- Driver JDBC Tipo 2 (también llamado driver API-Nativo) convierte el método JDBC a llamadas nativas de la API de la base de datos. Es más rápido que el puente JDBC-ODBC pero se necesita instalar la librería cliente de la base de datos en la máquina cliente y el driver es dependiente de la plataforma.
- Driver JDBC Tipo 3. Hace uso de un Middleware entre el JDBC y el SGBD.
- Driver JDBC Tipo 4 (también llamado Driver Java Puro directo a la base de datos). Es independiente a la plataforma.

2.2.2 Paquete

JDBC ofrece java.sql, en donde se encuentran las clases para manejar bases de datos:

- DriverManager: Para cargar un driver
- Connection: Para establecer conexiones con las bases de datos
- Statement: Para ejecutar sentencias SQL y enviarlas a las BBDD
- PreparedStatement: La ruta de ejecución está predeterminada en el servidor de base de datos que le permite ser ejecutado varias veces
- ResultSet: Para almacenar el resultado de la consulta

2.3 Servlets

Un servlet es una clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor. Aunque los servlets pueden responder a cualquier tipo de solicitudes, estos son utilizados comúnmente para extender las aplicaciones alojadas por servidores web, de tal manera que pueden ser vistos como applets de Java que se ejecutan en servidores en vez de navegadores web. Este tipo de servlets son la contraparte Java de otras tecnologías de contenido dinámico Web, como PHP y ASP.NET.

2.3.1 Ciclo de vida

1. Inicializar el servlet: Cuando un servidor carga un servlet, ejecuta el método init del servlet. El proceso de inicialización debe completarse antes de poder manejar peticiones de los clientes, y antes de que el servlet sea destruido.

2. Interactuar con los clientes: Después de la inicialización, el servlet puede dar servicio a las peticiones de los clientes. Estas peticiones serán atendidas por la misma instancia del servlet, por lo que hay que tener cuidado al acceder a variables compartidas, ya que podrían darse problemas de sincronización entre requerimientos simultáneos.

3. Destruir el servlet: Los servlets se ejecutan hasta que el servidor los destruye, por cierre del servidor o bien a petición del administrador del sistema. Cuando un servidor destruye un servlet, ejecuta el método destroy del propio servlet. Este método sólo se ejecuta una vez y puede ser llamado cuando aún queden respuestas en proceso, por lo que hay que tener la atención de esperarlas. El servidor no ejecutará de nuevo el servlet hasta haberlo cargado e inicializado de nuevo.

2.4 Bootstrap

Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end.

3 Desarrollo

A continuacipon se explica el proceso que se siguió para realizar este proyecto paso a paso:

Uso del Apache Tomcat: Como sabemos, usamos el IDE Apache NetBeans para desarrollar el proyecto, por lo que es necesario conectar Apache Tomat al IDE que estemos ocupando. En la Tarea 1 se explicó la instalación y la conexión con el IDE. En la figura podemos ver que el servidor esta funcionando con normalidad.

Figure 1: Tomcat funcionando

Evento Después de crear empaquetados para tener más organizado el proyecto, continuamos haciendo la primera clase de nuestro proyecto, la cual es Evento. En esta clase encontramos las funciones basicas de una clase como el constructor , los getter y los setter. Aquí mismo podemos visualizar cuales son los datos que manipularemos de un evento, los cuales son:

- idEvento
- nombreEvento
- sede
- duracion
- fechaInicio
- fechaTermino

```
1
2               /**
3                *
4                * @author Atziri Perez
5                */
6               public class Evento implements Serializable{
7                   private int idEvento;
8                   private String nombreEvento;
9                   private String sede;
10                  private double duracion;
11                  private Date fechaInicio;
12                  private Date fechaTermino;
13
14                  public Evento(){}
15
16                  public int getIdEvento() {
17                      return idEvento;
18                  }
19
20                  public void setIdEvento(int idEvento) {
21                      this.idEvento = idEvento;
22                  }
23
24                  public String getNombreEvento() {
25                      return nombreEvento;
26                  }
27
28                  public void setNombreEvento(String nombreEvento) {
29                      this.nombreEvento = nombreEvento;
30                  }
31
32                  public String getSede() {
33                      return sede;
```

```
34                  }
35
36                  public void setSede(String sede) {
37                      this.sede = sede;
38                  }
39
40                  public double getDuracion() {
41                      return duracion;
42                  }
43
44                  public void setDuracion(double duracion) {
45                      this.duracion = duracion;
46                  }
47
48                  public Date getFechaInicio() {
49                      return fechaInicio;
50                  }
51
52                  public void setFechaInicio(Date fechaInicio) {
53                      this.fechaInicio = fechaInicio;
54                  }
55
56                  public Date getFechaTermino() {
57                      return fechaTermino;
58                  }
59
60                  public void setFechaTermino(Date fechaTermino) {
61                      this.fechaTermino = fechaTermino;
62                  }
63
64
65                  @Override
66                  public String toString() {
67                      StringBuilder sb = new StringBuilder();
68                      sb.append("Evento{idEvento=").append(idEvento).append("\n");
69                      sb.append(", nombreEvento=").append(nombreEvento).append("\n");
70                      sb.append(", sede=").append(sede).append("\n");
71                      sb.append(", duracion=").append(duracion).append("\n");
72                      sb.append(", fechaInicio=").append(fechaInicio).append("\n");
73                      sb.append(", fechaTermino=").append(fechaTermino).append("\n");
74                      sb.append('}');
75                      return sb.toString();
76                  }
77
78              }
79
```

EventoDAO Después de crear la clase Evento, debemos hacer los métodos para lograr el CRUD que necesitamos, así como la conexión a la base de datos. Estos métodos de conexión y los métodos de CRUD se incluyeron en otra clase llamada EventoDAO. Cabe mencionar que se realizó la separación solo para abstraer la clase Evento, y mantener separados los métodos y la conexión de BD.

```
1               /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package com.ipn.mx.modelo.dao;
7
8  import com.ipn.mx.modelo.dto.Evento;
9  import java.sql.Connection;
10 import java.sql.DriverManager;
11 import java.sql.PreparedStatement;
12 import java.sql.ResultSet;
13 import java.sql.Date;
14 import java.sql.SQLException;
15 import java.util.List;
```

```java
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 *
 * @author Atziri Perez
 */
public class EventoDAO {
    public static final String SQL_INSERT ="insert into evento(nombreEvento, sede, duracion,fecha
    public static final String SQL_UPDATE ="update evento set nombreEvento = ?, sede = ?, duracion
    public static final String SQL_DELETE ="delete from evento where idEvento = ?";
    public static final String SQL_SELECT_ALL ="select * from evento";
    public static final String SQL_SELECT ="select * from evento where idEvento = ?";

    private Connection conexion = null;
    private void obtenerConexion(){
        String usr = "root";
        String pwd = "password";
        String driver = "com.mysql.cj.jdbc.Driver";
        String url ="jdbc:mysql://localhost:3306/ProyectoBase3CM4?serverTimezone=America/Mexico_C
        try {
            Class.forName(driver);
            conexion = DriverManager.getConnection(url, usr, pwd);
        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(Evento.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void create(Evento e)throws SQLException{
        obtenerConexion();
        PreparedStatement ps = null;
        try {
            ps = conexion.prepareStatement(SQL_INSERT);
            ps.setString(1, e.getNombreEvento());
            ps.setString(2, e.getSede());
            ps.setDouble(3, e.getDuracion());
            ps.setDate(4, e.getFechaInicio());
            ps.setDate(5, e.getFechaTermino());
            ps.executeUpdate();
        } finally{
            if (ps!=null)
                ps.close();
            if(conexion!=null)
                conexion.close();
        }
    }
    public void update(Evento e) throws SQLException{
        obtenerConexion();
        PreparedStatement ps = null;
        try {
            ps = conexion.prepareStatement(SQL_UPDATE);
            ps.setString(1, e.getNombreEvento());
            ps.setString(2, e.getSede());
            ps.setDouble(3, e.getDuracion());
            ps.setDate(4, e.getFechaInicio());
            ps.setDate(5, e.getFechaTermino());
            ps.setInt(6, e.getIdEvento());
            ps.executeUpdate();
        } finally{
            if (ps!=null)
                ps.close();
            if(conexion!=null)
                conexion.close();
        }
    }

```

```
82      public void delete(Evento e) throws SQLException{
83          obtenerConexion();
84          PreparedStatement ps = null;
85          try {
86              ps = conexion.prepareStatement(SQL_DELETE);
87              ps.setInt(1, e.getIdEvento());
88              ps.executeUpdate();
89          } finally{
90              if (ps!=null)
91                  ps.close();
92              if(conexion!=null)
93                  conexion.close();
94          }
95      }
96
97      public List readAll()throws SQLException{
98          obtenerConexion();
99          PreparedStatement ps = null;
100         ResultSet rs = null;
101         try{
102             ps = conexion.prepareStatement(SQL_SELECT_ALL);
103             rs = ps.executeQuery();
104             List resultados = obtenerResultados(rs);
105             if(resultados.size() > 0)
106                 return resultados;
107             else
108                 return null;
109         }finally{
110             if(rs!=null)
111                 rs.close();
112             if(ps!=null)
113                 ps.close();
114             if(conexion!= null)
115                 conexion.close();
116         }
117     }
118
119     public Evento read(Evento e) throws SQLException{
120         obtenerConexion();
121         PreparedStatement ps = null;
122         ResultSet rs = null;
123         try{
124             ps = conexion.prepareStatement(SQL_SELECT);
125             ps.setInt(1, e.getIdEvento());
126             rs = ps.executeQuery();
127             List resultados = obtenerResultados(rs);
128             if(resultados.size()>0)
129                 return (Evento)resultados.get(0);
130             else
131                 return null;
132         }finally{
133             if(rs!=null)
134                 rs.close();
135             if(ps!=null)
136                 ps.close();
137             if(conexion!= null)
138                 conexion.close();
139         }
140     }
141
142     private List obtenerResultados(ResultSet rs) throws SQLException{
143         List resultados = new ArrayList();
144         while(rs.next()){
145             Evento e = new Evento();
146             e.setIdEvento(rs.getInt("idEvento"));
147             e.setNombreEvento(rs.getString("nombreEvento"));
```

```
148             e.setSede(rs.getString("sede"));
149             e.setDuracion(rs.getDouble("duracion"));
150             e.setFechaInicio(rs.getDate("fechaInicio"));
151             e.setFechaTermino(rs.getDate("fechaTermino"));
152             resultados.add(e);
153         }
154         return resultados;
155     }
156 
157     public static void main(String[] args) {
158         Evento e = new Evento();
159         e.setIdEvento(1);
160         e.setNombreEvento("Hackaton");
161         e.setSede("Auditorio ESCOM");
162         e.setDuracion(5);
163         e.setFechaInicio(Date.valueOf("2020-10-16"));
164         e.setFechaTermino(Date.valueOf("2020-10-24"));
165 
166         EventoDAO dao = new EventoDAO();
167         try {
168             dao.create(e);
169         } catch (SQLException ex) {
170             Logger.getLogger(Evento.class.getName()).log(Level.SEVERE, null, ex);
171         }
172     }
173 }
174 
```

Como se puede ver, en EventoDAO.java es en donde se hace uso del API JDBC y se hace la conexión con la base de datos que ya existe en el usuario que se determinó en el código.

Podemos ver también los metodos create(), delete(),update(), read(), readAll() en donde conectamos con la base de datos y ejecutamos la consulta correspondiente. Es importante darse cuenta que aun no trabajamos con ningun aspecto WEB, solo se está desarrollando la parte funcional de nuestro proyecto, por consiguiente es recomendable probar la funcionalidad del proyecto hasta este punto paraa verificar que las acciones CRUD se hacen corretamente en la base de datos.

Servlet Una vez que estamos seguros que el proyecto funciona bien, podemos empezar a trabajar con servlets para ir formando la estructura de nuestra página web. Creamos un servlet en donde determinamos las acciones principales de este:

```
1         String accion = request.getParameter("accion");
2         if (accion.equals("listaDeEventos")) {
3             listaDeEventos(request, response);
4         } else {
5             if (accion.equals("nuevo")) {
6                 agregarEvento(request, response);
7             } else {
8                 if (accion.equals("eliminar")) {
9                     eliminarEvento(request, response);
10                } else {
11                    if (accion.equals("actualizar")) {
12                        actualizarEvento(request, response);
13                    }
14                    if (accion.equals("guardar")) {
15                        almacenarEvento(request, response);
16                    }
17                    if (accion.equals("ver")) {
18                        mostrarEvento(request, response);
19                    }
20                }
21            }
22        }
23 
```

Cada uno de los métodos mostrados en el código anterior, están determinados en este mismo servlet, en cada uno incluimos código html para formar la estructura de la página web respecto al Front-End.

3.0.1 listaDeEventos

listaDeEventos será el home de nuestra página, es donde se mostrará el listado de los eventos que se encuentran en la base de datos, así como la opción de realizar acciones a cada uno (Ver, Eliminar o Editar). Esta es nuestra vista principal.

```
    private void listaDeEventos(HttpServletRequest request, HttpServletResponse response) throws
        ...
        int idEvento;
            String nombreEvento;
            String sede;
            double duracion;
            Date fechaInicio;
            Date fechaTermino;

            EventoDAO dao = new EventoDAO();
            List lista = dao.readAll();
        for (int i = 0; i < lista.size(); i++) {
            Evento evento = (Evento) lista.get(i);
            idEvento = evento.getIdEvento();
            nombreEvento = evento.getNombreEvento();
            sede = evento.getSede();
            duracion = evento.getDuracion();
            fechaInicio = evento.getFechaInicio();
            fechaTermino = evento.getFechaTermino();
    ..}

```

Dejando de lado el código html, podemos ver que se hace uso del método readAll, el cual es una consulta SELECT a la base de datos, la cual la desarrollamos en EventDAO.java

3.0.2 agregarEvento

El método agregarEvento, solo nos redirige a otro form html para crear un nuevo evento:

```
    private void agregarEvento(HttpServletRequest request, HttpServletResponse response) {
        try {
            request.getRequestDispatcher("eventoForm.html").forward(request, response);
        } catch (ServletException | IOException ex) {
            Logger.getLogger(Servlet.class.getName()).log(Level.SEVERE, null, ex);
        }

```

3.0.3 eliminarEvento

El método eliminatEvento, eliminará un evento tomando como parámetro el id del evento a eliminar:

```
        EventoDAO dao = new EventoDAO();
        Evento e = new Evento();
        try {
            e.setIdEvento(Integer.parseInt(request.getParameter("id")));
            e = dao.read(e);
            dao.delete(e);
            listaDeEventos(request, response);
        } catch (SQLException | IOException ex) {
            Logger.getLogger(Servlet.class.getName()).log(Level.SEVERE, null, ex);
        }

```

3.0.4 almacenarEvento

El método almacenarEvento hace uso de los metodos creat() y update() para almacenar un evento nuevo en la base de datos, o bien si ya existe, actualizar sus datos a los nuevo que el usuario escribió. Esta validación se realizó verificando si se envió como parámetro en la URL la id del evento (En caso de actualizar datos)usando el método POST:

```
    private void almacenarEvento(HttpServletRequest request, HttpServletResponse response) {
        EventoDAO dao = new EventoDAO();
        Evento e = new Evento();
        if (request.getParameter("id") == null || request.getParameter("id").isEmpty()) {
            e.setNombreEvento(request.getParameter("nombreEvento"));
            e.setSede(request.getParameter("sedeEvento"));
            e.setDuracion(Double.parseDouble(request.getParameter("duracion")));
            e.setFechaInicio(Date.valueOf(request.getParameter("fechaInicio")));
            e.setFechaTermino(Date.valueOf(request.getParameter("fechaTermino")));
            try {
                dao.create(e);
                response.sendRedirect("Servlet?accion=listaDeEventos");
            } catch (SQLException | IOException ex) {
                Logger.getLogger(Servlet.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else {
            e.setIdEvento(Integer.parseInt(request.getParameter("id")));
            e.setNombreEvento(request.getParameter("nombreEvento"));
            e.setSede(request.getParameter("sedeEvento"));
            e.setDuracion(Double.parseDouble(request.getParameter("duracion")));
            e.setFechaInicio(Date.valueOf(request.getParameter("fechaInicio")));
            e.setFechaTermino(Date.valueOf(request.getParameter("fechaTermino")));
            try {
                dao.update(e);
                response.sendRedirect("Servlet?accion=listaDeEventos");
            } catch (SQLException | IOException ex) {
                Logger.getLogger(Servlet.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

```

3.0.5 actualizat y mostrar Evento

Los metoddos actualizarEvento y mostrarEvento hacen uso de update() y read() respectivamente, instanciando el objeto Evento de la misma manera que se realizó en los métodos anteriores. *Para ver código, ir al código fuente

```

```

4 Resultados

Finalmente, mostramos el proyecto funcionando:

- Primero mostramos la página incial Lista de eventos donde vemos los eventos que se encuentran actualmente en la base de datos:

Figure 2: Lista De Eventos

- Ahora agregamos un evento

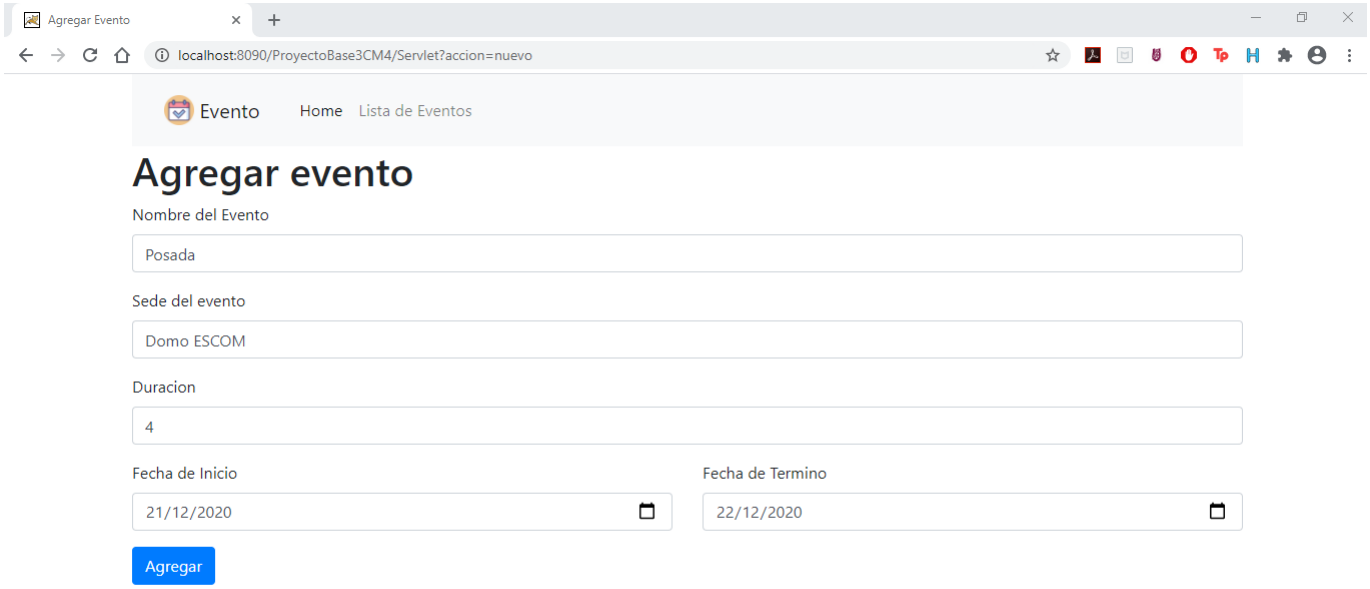

Figure 3: Agregar Evento

- Ahora probemos la funcion de actualizar evento, actualizaremos el evento 7

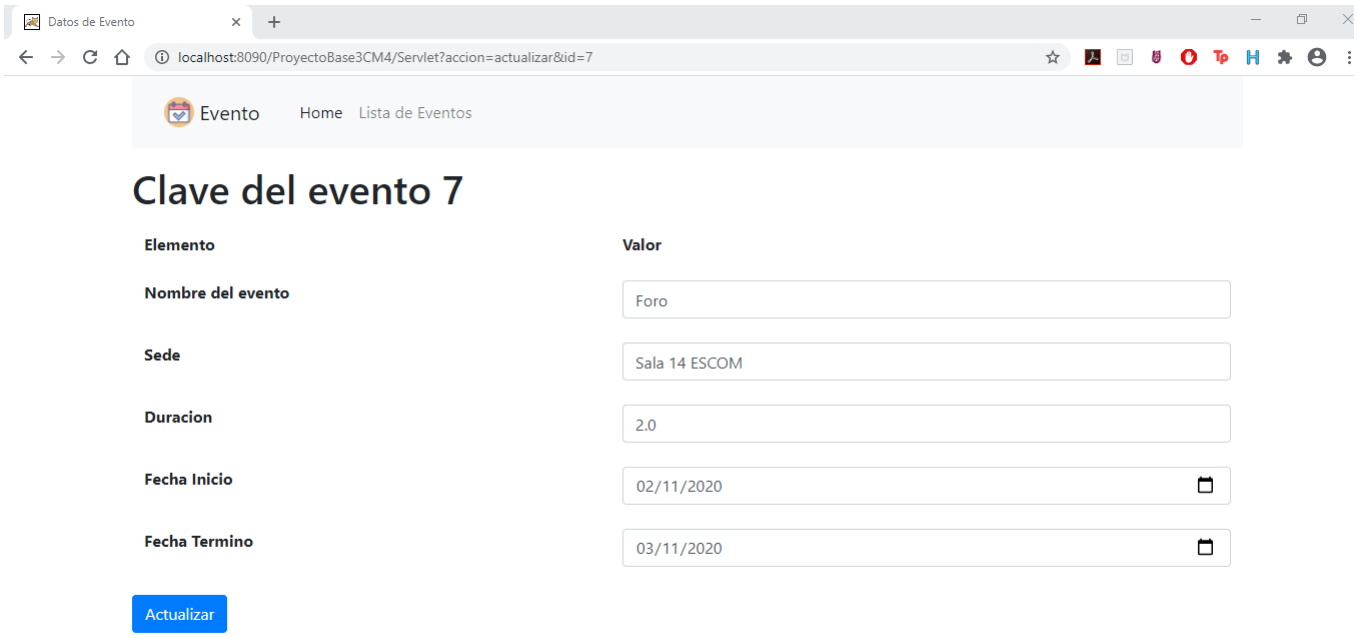

Figure 4: Actualizar evento

- Continuamos mostrando la información completa de un evento, en este caso el numero 10

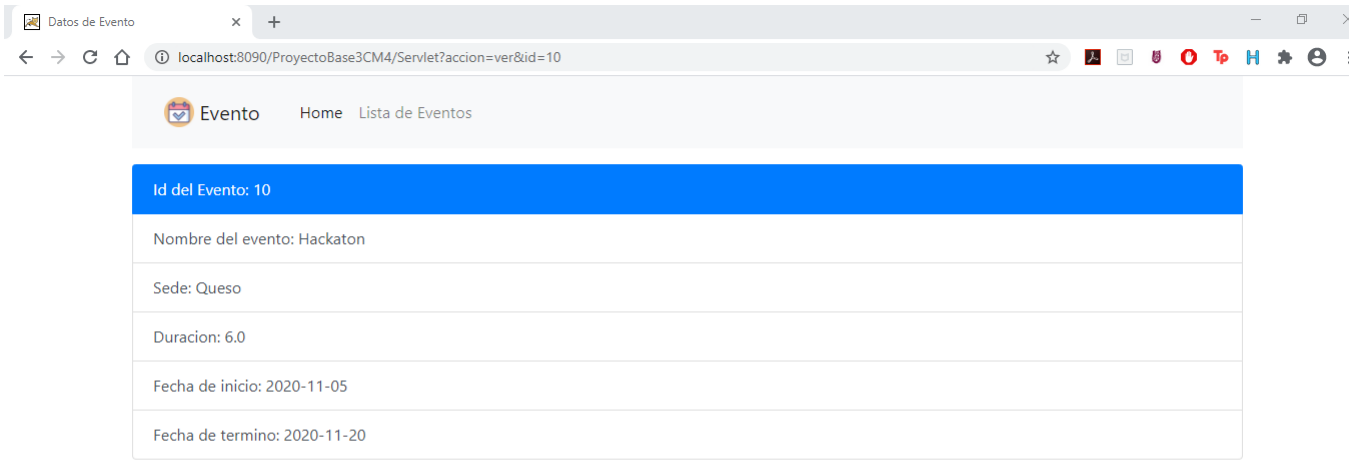

Figure 5: Ver evento

- Eliminaremos ese mismo evento, y lista de Eventos queda de la siguiente manera:

Figure 6: Lista de Eventos con modificaciones

5 Conclusión

El proyceto resultó ser muy completo e integral para entender todas las tecnologías que se necesitan durante la implementación de un proyecto web. En mi experiencia, solo estaba familiarizada con el usp de Bootstrap para la parte del front-end, sin embargo no fue dificil adaptarme a JDBC, ni a los servlets ya qe siguen la línea de las herramientas web que he utilizado (Flask para python, ASP C, etc.)

Atziri Pérez García

6 Referencias Bibliográficas

References

[Tomcat, 2019] Apache Tomcat *Tomcat 9* Apache Tomcat Disponible en: *https://tomcat.apache.org/*

[Bootstrap, 2020] *Bootstrap* Disponible en: *https://getbootstrap.com*

[JDBC, 2020] *¿Que es JDBC?* Disponible en: *http://profesores.fi-b.unam.mx/sun/Downloads/Java/jdbc.pdf*