Implementación de una aplicación Web mediante SpringBoot: Una base para un sistema electrónico de votaciones para una población reducida a una unidad habitacional.

Flores-Silva P.[1,2,3] , Pedro Flores Silva[1,2,3] and Arturo Flores Silva[1,2,3]

[1] Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, UNAM Cd.Mx. 04510, MX
[2] Instituto de Física, UNAM Cd.Mx. 04510, MX
[3] Facultad de Ciencias,UNAM Cd.Mx. 04510, MX

Resumen Se implementó una aplicación web mediante SpringBoot, Maven, Tomcat, MySQL y java para la creación de un sistema de votación totalmente electrónico para una población reducida. Se logró implementar el esqueleto o una base para dicho sistema de votaciones. La implementación cuenta con un sistema de seguridad que, da acceso solo a usuarios registrados o dados de alta, además cuenta con las funcionalidades de consulta, modificación, adición y eliminación de los datos de una tabla para una base de datos propuesta para uno o más personas con un orden jerárquico determinado. Estas funcionalidades cuentan con validaciones de datos de tal forma que si una persona, que tiene el acceso a dichas funcionalidades, ingresa valores diferentes a los definidos en la tabla se mostrará una pagina de error. Finalmente se implementó una página en la cual se pueden publicar avisos relevantes y es accesible por cualquier usuario registrado.

Keywords: Sistema de Votación · SpringBoot · Maven · Tomcat · Java.

1. Introducción

Toda persona tiene preferencias, pero no todas las personas tienen las mismas preferencias. Como sociedad hay ocasiones en las cuales es necesario decidir acerca de un tema relevante, ya sea la elección de algún presidente, decidir si se legaliza la marihuana o cualquier otro tema donde las preferencias de cada una de las personas sea un factor importante en la toma de decisiones. Un ejemplo muy claro de la importancia de estas preferencias se encuentra en el año 2006 en México. Los candidatos preferidos a tomar la presidencia de dicho país fueron Felipe Calderón H. y Andrés López O. El ganador de la contienda electoral fue Felipe C. con el 36.69 % de los votos válidos, por otro lado Andrés O. se posicionó en segundo lugar con el 36.11 % de los votos válidos [1]. En aquella época el presidenciable que ocupó el segundo lugar no aceptó el resultado pues la contienda, como puede verse, fue muy cerrada. Gran parte de la población, según López O., lo apoyaba. Debido a esta situación se generaron múltiples movilizaciones a favor y en contra de la decisión tomada por el pueblo.
En la actualidad el sistema de votación de este país no es del todo electrónico. Aún las personas emiten sus votos mediante papel, para posteriormente ser contabilizados por el órgano regulador llamado INE. Esta falta de "modernizacion" ha producido muchos casos de escandalo, ya que se ha dicho es altamente corruptible el voto mediante el papel.
Conociendo esta problemática se optó por fabricar una aplicación web que simule un proceso de toma de decisiones totalmente electrónico. Sin embargo, dado que fabricar una aplicación web útil

para todo un país es extremadamente complicado se optó a reducir el problema a una población más pequeña como la de una unidad habitacional en donde también existe la necesidad de realizar un proceso de toma de decisiones de cualquier índole ya sea elegir algún administrador, decidir en que se debe invertir el dinero entre otros aspectos relevantes, también es necesario mostrar avisos a la comunidad ya que sin ellos no es posible tener una comunidad organizada.
El presente documento pretende mostrar la documentación de la aplicación web creada para este fin (implementada con SpringBoot y Java). Cabe mencionar que el proyecto no ofrece las funcionalidades mencionadas, sin embargo, se considera este la base para crear un proyecto de está índole. Más adelante se darán detalles de esto último.

2. Metodología

En la comunidad propuesta es necesario que exista un orden jerárquico entre habitantes. Uno o más usuarios tendrán la posibilidad de administrar la pagina de avisos al usuario regular, dar de alta nuevos usuarios, actualizar los datos de los usuarios existentes y tener la posibilidad de eliminarlos. A estos usuarios los denominaremos administradores.
Un usuario regular únicamente puede acceder a los avisos publicados por el o los administradores y realizar una votación. A estos usuarios los llamaremos Habitantes o usuarios.
Estas acciones u operaciones requieren de una base de datos en la cual se almacenará la información correspondiente a cada rubro. Esta base de datos es esencial para la aplicación web ya que sin ella no es posible, de una forma natural, realizar ninguna de las acciones previamente expuestas. La estructura de la base de datos general se construye mediante un diagrama entidad relación (figura 1) en el programa MySQL Workbench 8.0.
La base de datos de la figura 1 consta de cinco tablas. A continuación se describe cada una de ellas, así como cada elemento de las mismas.

- Tabla habitante:
  - idhabitante: Llave primaria de esta tabla, definida como un valor entero no nulo.
  - enabled: definido como un bit. Servirá para reconocer si un habitante aún vive en la unidad habitacional. Más adelante se restringirá a los valores true o false, donde el primero denotará que el habitante es miembro de la comunidad.
  - password: Será la contraseña designada al habitante en cuestión. Sin ella el habitante no podrá acceder al sistema. Se define como un varchar.
  - username: nombre de usuario con el que podrá, junto con su contraseña, acceder al sistema. Se define como un varchar de índice único, es decir, dos personas no podrán poseer un mismo username.
  - nombre: Nombre completo del habitante servirá para identificar al usuario. Definido como un varchar.
- Tabla rol:
  - idrol: Llave primaria de esta tabla definida como un valor entero no nulo. Únicamente tendrá dos valores 1 y 2.
  - rol: Se define el orden jerarquico antes mencionado. ROLE_ADMIN y ROLE_USER
- Tabla rolHabitante: esta tabla es útil para definir una relación muchos a muchos entre las tablas rol y habitante, recordemos que un habitante puede tener dos roles: administrador y usuario. Para esta tabla se definen dos llaves foráneas: habitante_idhabitante se relaciona con la llave primaria idhabitante de la tabla habitante, mientras que la llave foránea rol_idrol se relaciona con la llave primaria idrol de la tabla rol.

- Tabla categoría: Tabla que complementa la tabla habitante ya que en la misma se pueden agregar información relevante acerca de un habitante.

  - idcategoria: llave primaria de esta tabla. Se define como un numero entero.
  - estado: Estado de la república de origen del habitante. Definido como un varchar.
  - evaluación: evaluación en escala del 0 al 10 del habitante en términos escolares o de reputación, este campo puede interpretarse de a cuerdo a la situación del habitante.
  - fechaEvaluacion: fecha para la cual se realizo la evaluación al habitante.
  - habitante_idhabitante: llave foránea que relaciona esta tabla con la tabla de habitante. La relación que se sigue es de muchos a uno ya que un habitante puede tener más de una evaluación.
  - institución: institución para la cual el habitante trabaja o estudia. Se define como un varchar.

- Tabla propuesta

  - idpropuesta: llave primaria definida como un entero
  - nombre: nombre de la propuesta. Definido como un varchar
  - fecha: fecha a la cual se planteó la propuesta.
  - asambleaTipo: Tipo de asamblea. Se define como varchar.
  - votos: cantidad de votos para esta propuesta. Definido como un entero.

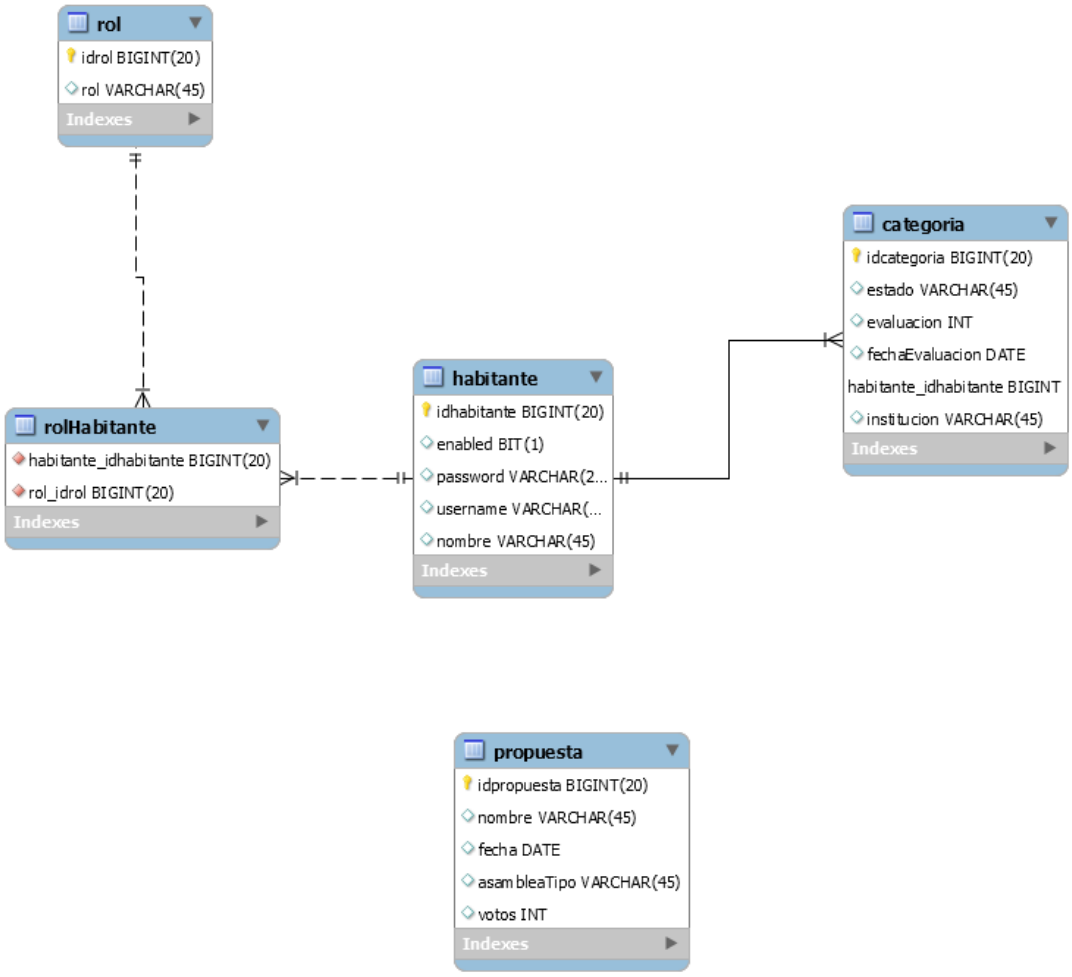

Figura 1. Diagrama Entidad-Relación para el sistema de votaciones.

Dado que la aplicación web requiere conectarse a la base de datos previamente definida es necesario definir las propiedades de la base de datos en la aplicacion web. Esto se hace copiando las siguientes lineas al archivo application.properties.

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/aesmdbf?
useUnicode=true&amp&useJDBCCompliantTimezoneShift=true&amp&
useLegacyDatetimeCode=false&amp&serverTimezone=UTC

spring.datasource.username=root
spring.datasource.password=root
```

```
9
10 spring.jpa.show-sql=true
```

Listing 1.1. En este caso en la linea 3 aesmdbf corresponde al nombre de la base de datos en MySQL. Las lineas 7 y 8 corresponden a el usuario y contraseña definidos para la base de datos en MySQL

Mientras que en el pom.xml se agrega la dependencia para MySQL.

```
1 <dependency>
2       <groupId>mysql</groupId>
3       <artifactId>mysql-connector-java</artifactId>
4       <scope>runtime</scope>
5     </dependency>
```

Listing 1.2. dependencia en el pom para MySQL

Una vez configurada la base de datos se procede a realizar los POJOS correspondientes a las entidades de las tablas en la base de datos. Estos Pojos son como siguen:

```
1  @Entity
2  public class Habitante  {
3
4    @Id
5    @Column(name = "idhabitante")
6    private Long idhabitante;
7
8    @Column(name = "nombre")
9    private String nombre;
10
11   @Column(name = "password")
12   private String password;
13
14   @Column(name = "username")
15   private String username;
16
17   @Column(name = "enabled")
18   private boolean enabled;
19
20   @ManyToMany(fetch = FetchType.EAGER, cascade = {CascadeType.MERGE}) //aqui
       se hace la relacion many to many con la tabla  rol
21   @JoinTable(name="rolhabitante",
22   joinColumns=@JoinColumn(name="habitante_idhabitante"),
23   inverseJoinColumns=@JoinColumn(name="rol_idrol"))
24   private Set<Rol> rol = new HashSet<Rol>(0);
25
26   //Getters y Setters
```

Listing 1.3. POJO correspondiente a la entidad Habitante.

Notese que la linea 20 a 24 se hace la relación muchos a muchos con la tabla Rol, esto mediante la tabla rolhabitante. En las lineas 22 y 23 se unen las columnas de la tabla rol habitante. En la linea 24 se define la tabla Rol como un HashSet para evitar repeticiones y permitir objetos nulos.

```
1 @Id
2   @GeneratedValue(strategy = GenerationType.AUTO)
```

```
3   @Column(name = "idrol")
4   private Long idrol;
5
6   @Column(name = "rol")
7   private String rol;
8
9   @ManyToMany(mappedBy = "rol") //definicion de la relacion maby to maby con
      la tabla habitantes
10    private Set<Habitante> habitantes = new HashSet<>();
11    //Getters y Setters
```

Listing 1.4. POJO correspondiente a la entidad Rol.

Análogo al POJO de habitante, se hace la relación many to many en las lineas 9 a 10.

Dado que cualquier habitante que requiera acceder a la pagina de avisos o a la pagina de administrador, si éste tiene los privilegios, debe autenticarse con su usuario y contraseña se define la clase WebSecurity para este fin. El codigo de esta clase es:

```
1  @Configuration
2  @EnableWebSecurity
3  public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
4
5    // Necesario para evitar que la seguridad se aplique a los resources
6    // Como los css, imagenes y javascripts
7    String[] resources = new String[] { "/include/**", "/css/**", "/icons/**",
       "/img/**", "/js/**", "/layer/**" };
8
9    @Override
10   protected void configure(HttpSecurity http) throws Exception {
11     http.authorizeRequests()
12       .antMatchers(resources).permitAll().antMatchers("/", "/index").
     permitAll() //permite a cualquier persona acceder a los resources
     definido aqui arriba
13       .antMatchers("/admin*").access("hasRole('ADMIN')") //permite
     unicamente al admin acceder a la pagina admin
14       .antMatchers("/listahabitantes*").access("hasRole('ADMIN')") //permite
      unicamente al admin acceder a la pagina lista habitantes
15       .antMatchers("/editHabitante*").access("hasRole('ADMIN')") //permite
     unicamente al admin acceder a la pagina edit habitante
16       .antMatchers("/addHabitante*").access("hasRole('ADMIN')") //permite
     unicamente al admin acceder a la pagina addhabitante
17       .antMatchers("/deleteHabitante*").access("hasRole('ADMIN')") //permite
      unicamente al admin eliminar un uisuario
18       .antMatchers("/user*").access("hasRole('USER') or hasRole('ADMIN')")
     //permite admin y usuario acceder a la pagina usuario
19       .anyRequest().authenticated().and().formLogin() //cualquier solicitud
     debe ser de un usuario autenticado
20       .loginPage("/login").permitAll().defaultSuccessUrl("/menu").failureUrl
     ("/login?error=true") //permite a cualquier persona ver el login, si el
     login fue exitoso redirige al menu por default
21       .usernameParameter("username").passwordParameter("password").and().
     logout().permitAll() //define los parametros username y password para
     porder loguearse
```

```
22        .logoutSuccessUrl("/login?logout"); //cierra sesion y muestra el
      cartel de has cerrado sesion
23    }
24
25    BCryptPasswordEncoder bCryptPasswordEncoder; //Para encriptar las
      contrasenas que se almacenan en la db y solicite a la db que las
      contrasenas tengan ese formato
26
27    // Crea el encriptador de contrasenas
28    @Bean
29    public BCryptPasswordEncoder passwordEncoder() {
30      bCryptPasswordEncoder = new BCryptPasswordEncoder(4);
31  //El numero 4 representa que tan fuerte es la encriptacion.
32  //Se puede en un rango entre 4 y 31.
33  //Si no se pone un numero el programa utilizara uno aleatoriamente cada vez
34  //que inicie la aplicacion, por lo cual las contrasenas encriptadas no
      funcionaran bien
35      return bCryptPasswordEncoder;
36    }
37
38    @Autowired
39    HabitanteDetailsServiceImpl habitanteDetailsService;
40
41    // Registra el service para usuarios y el encriptador de contrasena
42    @Autowired
43    public void configureGlobal(AuthenticationManagerBuilder auth) throws
      Exception {
44
45      // Setting Service to find User in the database.
46      // And Setting PassswordEncoder
47      auth.userDetailsService(habitanteDetailsService).passwordEncoder(
      passwordEncoder());
48    }
49  }
```

Listing 1.5. Clase WebSecurity para el acceso mediante usuario y contraseña.

La anotacion @EnableWebSecurity en la linea 2 permite habilitar el soporte para Spring web Security y provee la integración Modelo vista controlador. Se extiende a WebSecurityConfigurerAdaprter quien permite emplear algunos de sus métodos específicos de la configuración de seguridad web.
El método configure(HttpSecurity) define que URL's deben ser protegidas y cuales no. En este caso los resources definidos en la linea 7, las URL / , /index , /login y /logout no requieren autenticación, esto es posible con el método permitAll().
Por otro lado el acceso a ciertas paginas se realiza con el método .acces(tipo de Rol). Por ejemplo un usuario con Rol ADMIN tiene acceso a las URL /addHabitante, /deleteHabitante, /editHabitante y /listahabitantes mientras que un usuario con rol USER no podrá acceder a ellas.
Los métodos usernameParameter y passwordParameter definen los parámetros de username y contraseña con los cuales se puede acceder al sistema.
Finalmente en las lineas 42 a 49 se configura el POJO de HabitanteDetailsServiceImpl y el encriptador de contraseñas definido como bCryptPasswordEncoder de tal manera que pueda encontrarse un usuario en la base de datos previamente definida, este usuario deberá tener en la misma tabla

una contraseña encriptada.

El controlador de la aplicación web es el encargado de habilitar las vistas en la pagina web y administrar el tipo de mapeo que se solicita en una determinada vista. Para este caso se tiene el siguiente código

```
@Controller
public class AppController {

  private HabitanteService habitanteService;

  public AppController () {

  }

  @Autowired
  public AppController(HabitanteService habitanteService) {
    this.habitanteService = habitanteService;
  }


  @GetMapping({"/","/login"}) //solicita acceso o mapea al login
  public String index() {
    return "index";
  }

  @GetMapping("/menu") //solicita acceso o mapea al menu
  public String menu() {
    return "menu";
  }

  @GetMapping("/user") //solicita acceso o mapea a la pagina user
  public String user() {
    return "user";
  }

  @GetMapping("/admin") //solicita acceso o mapea a la pagina admin
  public String admin() {
    return "admin";
  }

  @SuppressWarnings("rawtypes")
  @GetMapping("/listahabitantes") //funcion para mapear a la pagina lista
    habitantes y obtenga de la db todos los habitantes disponibles en ella
    misma
  public ModelAndView displayAllHabitante() {
    System.out.println("Pagina de Habitante solicitada : Todos los
    Habitantes"); //print en consola no en pagina
    ModelAndView mv = new ModelAndView();
    List habitanteList = habitanteService.getAllHabitantes();
    mv.addObject("habitanteList", habitanteList);
    mv.setViewName("listahabitantes");
```

```
44      return mv;
45    }
46
47    @GetMapping("/addHabitante") //funcion para mapear a la pagina agregar
        habitante, para ello debe instanciarse un nuevo habitante
48    public ModelAndView displayNewHabitanteForm() {
49      ModelAndView mv = new ModelAndView("addHabitante");
50      mv.addObject("headerMessage", "Add Habitante Details");
51      mv.addObject("habitante", new Habitante());
52      return mv;
53    }
54
55    @PostMapping(value = "/addHabitante") //funcion para verificar si se
        guarda correctamente en la db el habitante si es correcto redirecciona
        al menu
56    public ModelAndView saveNewHabitante(@ModelAttribute Habitante habitante,
        BindingResult result) {
57      ModelAndView mv = new ModelAndView("redirect:/menu");
58
59      if (result.hasErrors()) {
60        return new ModelAndView("redirect:/error");
61      }
62      boolean isAdded = habitanteService.guardarHabitante(habitante);
63      if (isAdded) {
64        mv.addObject("message", "Se ha agregado un nuevo Habitante");
65      } else {
66        return new ModelAndView("redirect:/error");
67      }
68
69      return mv;
70    }
71
72    @GetMapping("/editHabitante/{idhabitante}") //funcion que mapea a la
        pagina para editar un habitante con un idHabitante especifico
73    public ModelAndView displayEditHabitanteFrom(@PathVariable Long
        idhabitante) {
74      ModelAndView mv = new ModelAndView("/editHabitante");
75      Habitante habitante = habitanteService.getHabitantePorId(idhabitante);
76      mv.addObject("header message", "Edit Habitante Details");
77      mv.addObject("habitante", habitante);
78      return mv;
79    }
80
81    @PostMapping(value = "/editHabitante/{idhabitante}") //funcion que
        verifica si los nuevos datos son correctos y fue posible editar la info
        del habitante, si es exitosa regresa al menu
82    public ModelAndView saveEditedHabitante(@ModelAttribute Habitante
        habitante, BindingResult result) {
83      ModelAndView mv = new ModelAndView("redirect:/menu");
84
85      if (result.hasErrors()) {
86        System.out.println(result.toString());
```

```
87        return new ModelAndView("redirect:/error");
88      }
89      boolean isSaved = habitanteService.guardarHabitante(habitante);
90      if (!isSaved) {
91        return new ModelAndView("redirect:/error");
92      }
93
94      return mv;
95    }
96
97    @GetMapping("/deleteHabitante/{idhabitante}") //funcion para eliminar un
        habitante de la db redirecciona al menu independientemente si fue
        exitoso o no
98    public ModelAndView deleteHabitanteById(@PathVariable Long idhabitante) {
99      boolean isDeleted = habitanteService.eliminarHabitanteporId(idhabitante)
        ;
100     System.out.println("Respuesta de eliminacion de Habitante" + isDeleted);
         //print en consola
101     ModelAndView mv = new ModelAndView("redirect:/menu");
102     return mv;
103   }
104   //Todas las funciones requieren de funciones definidas en la interfaz
        habitante service
105 }
```

Listing 1.6. Controlador de la aplicación Web: Mapeos.

Para esta clase son necesarios los metodos definidos en la interfaz HabitanteService que será explicada más adelante. Los mapeos de las lineas 16 a 35 no requieren de funciones especiales. Por otro lado los mapeos de las lineas 35 a 105 requieren de funciones especiales ya que estos consisten en la manipulación de los datos almacenados en la base de datos.
El primer mapeo a considerar es el de la linea 36 a 45 consiste en la consulta de la información de todos los habitantes definida en la tabla habitante. En este método se crea una lista llamada habitanteList definida con el método getAllHabitantes() de la Interfaz HabitanteService, esta función consulta la base de datos y guarda todos los elementos disponibles de la tabla habitante y los retorna en forma de lista. Esta lista es agregada como un objeto al controlador ModelAndView y finalmente lo retorna.
El mapeo addHabitante consiste en agregar un nuevo habitante a la base de datos, esto con sus respectivos campos definidos en la tabla habitante. El mapeo de tipo Get es el encargado de mapear a la URL /addhabitante, donde se mostrarán, en cuadros de texto definidos en la pagina HTML, los parámetros de la entidad Habitante, instanciando un habitante nuevo. Por otro lado el mapeo tipo Post se encargara de guardar la instancia previamente realizada, esto gracias al método guardarHabitante(habitante) definido en la Interfaz HabitanteService. En Este mapeo se definen los validadores de datos, lineas 59,60 y 66. Si se pasa un parámetro erróneo a algún campo de la instancia la base de datos enviará un error y el controlador, a través de la instancia result de la clase BindingResult, detectara que hay un error. Si Existe un error se direccionará a la página de error. Como ejemplo tenemos que si se trata de guardar un usuario con una llave primaria diferente a un entero, entonces la instancia result rediccionará a la URL /error y los datos proporcionados no serán guardados en la base de datos.
En caso de no encontrar errores los datos ingresados se almacenarán en la base de datos y redirecciona a la URL /menu.

El tercer mapeo de interés es el mapeo /editHabitante/idhabitante, éste da la posibilidad de editar un habitante de acuerdo al identificador del mismo. Nuevamente el método get mapea a la pagina y solicita a la entidad Habitante la instancia con dicho identificador, esto mediante el método getHabitantePorId(idhabitante) de la Interfaz HabitanteService, en la pagina HTML se muestran todos los campos para esta instancia. El método Post tiene la habilidad de guardar la instancia previamente adquirida. Aquí también se cuenta con los validadores de datos y funcionan de la misma forma que en el caso anterior.
Finalmente el mapeo /deleteHabitante/idhabitante se emplea para eliminar un habitante con un determinado identificador para ello se usa el método eliminarHabitanteporId(idhabitante) de la clase HabitanteService. Para este mapeo no se implementó ningún validador de datos ya que el único problema que podría presentarse es la imposibilidad de eliminar un habitante debido a una mala configuración de la conexión entre la base de datos y la aplicación o una mala implementación, sin embargo este problema se debería al programador y no al usuario final.

La interfaz HabitanteService se define como sigue:

```
public interface HabitanteService {

  @SuppressWarnings("rawtypes")
  public List getAllHabitantes();
  public Habitante getHabitantePorId(Long idhabitante);
  public boolean guardarHabitante(Habitante habitante);
  public boolean eliminarHabitanteporId(Long idhabitante);
}
```

Listing 1.7. Interfaz HabitanteService.

cuya clase que la extiende se define como HabitanteDetailsServiceImpl y su código es el siguiente.

```
@Service
public class HabitanteDetailsServiceImpl implements UserDetailsService ,
    HabitanteService {

  @Autowired
  HabitanteRepository habitanteRepository;

  private HabitanteRepository repository;

  public HabitanteDetailsServiceImpl () {

  }

  @Autowired
  public HabitanteDetailsServiceImpl(HabitanteRepository repository) {
    super();
    this.repository = repository;
  }

  @SuppressWarnings({ "rawtypes", "unchecked" })
  @Override
  public UserDetails loadUserByUsername(String username) throws
    UsernameNotFoundException{
```

```
22      //Buscar el usuario con el repositorio y si no existe lanzar una
        exepcion
23      mx.unam.iimas.proyecto.entity.Habitante appHabitante =
24          habitanteRepository.findByUsername(username).orElseThrow(() -> new
        UsernameNotFoundException("No existe Habitante"));
25      //Mapear nuestra lista de Rango con la de spring security
26      List grantList = new ArrayList();
27      for (Rol rol: appHabitante.getRol()) {
28        // ROLE_USER, ROLE_ADMIN,..
29        GrantedAuthority grantedAuthority = new SimpleGrantedAuthority(rol.
        getRol());
30        grantList.add(grantedAuthority);
31      }
32
33      //Crear El objeto UserDetails que va a ir en sesion y retornarlo.
34          UserDetails habitante = (UserDetails) new User(appHabitante.
        getUsername(), appHabitante.getPassword(), grantList);
35          return habitante;
36    }
37
38
39
40    @SuppressWarnings({ "rawtypes", "unchecked" })
41    @Override
42    public List getAllHabitantes() { //funcion que retorna una lista con todos
        los habitantes
43      List list = new ArrayList();
44      repository.findAll().forEach(e -> list.add(e));
45      return list;
46    }
47
48    @Override
49    public Habitante getHabitantePorId(Long idhabitante) { //funcion que
        retorna un habitante de acuerdo al id que se le pasa a la funcion
50      Habitante habitante = repository.findById(idhabitante).get();
51      return habitante;
52    }
53
54    @Override
55    public boolean guardarHabitante(Habitante habitante) { //funcion que
        guarda un habitante de acuerdo al id que se le pasa a la funcion
56      try {
57        repository.save(habitante);
58        return true;
59      } catch (Exception ex) {
60        return false;
61      }
62    }
63
64    @Override
65    public boolean eliminarHabitanteporId(Long idhabitante) { ////funcion que
        elimina un habitante de acuerdo al id que se le pasa a la funcion
```

```
66      try {
67        repository.deleteById(idhabitante);
68        return true;
69      } catch(Exception ex) {
70        return false;
71      }
72    }
73  }
```

Listing 1.8. HabitanteDetailsServiceImpl.

Esta clase requiere de la interfaz HabitanteRepository, quién provee métodos para operaciones CRUD con la base de datos.
El método de la linea 19 a 36 permite realizar cargar un usuario o habitante mediante la operacion CRUD definida con el método loadUserByUsername a través de su username y verificar los roles que posee así como cargarlos en memoria. Si no se encuentra el habitante con el username de referencia este método lanza una excepción. Este método permite el login con usuario y contraseña, este método se emplea en la clase WebSecurityConfig.
El método de la linea 40 a 46 también hace una operación CRUD y consiste en recuperar o consultar todos los datos guardados en la base de datos y guardar esta información en una lista para así retornarla.
El método getHabitantePorId de las lineas 48 a 52 es similar al primer método, consulta en la base de datos un habitante cuyo identificador corresponda a el idhabitante que le fue suministrado y retorna dicho habitante con todos sus atributos. A este método se le debió implementar una excepción del mismo tipo que el primer método.
Los últimos dos métodos también son operaciones CRUD de guardar en base de datos y eliminar algún habitante de la base de datos. En ambos se uso la sentencia try-catch con el fin de realizar parte de las validaciones de datos implementadas en el controlador de la aplicación web, ya que si ocurre un problema en el guardado o eliminación estas funciones retornaran false, este valor es enviado a los métodos post del controlador y éstos finalmente redireccionarán a la URL /error.

Finalmente las interfaces que proveen los métodos para las operaciones CRUD son las siguientes:

```
1  @Repository
2  public interface HabitanteRepository extends CrudRepository<Habitante, Long>
       {
3    public Optional<Habitante> findByUsername(String username);
4
5  }
```

Listing 1.9. HabitanteRepository para operaciones CRUD entre la aplicación Web y la tabla habitante.

```
1  @Repository
2  public interface RolRepository extends CrudRepository<Rol,Long>{
3
4    List<Rol> findByRol(String rol);
5  }
6
7  }
```

Listing 1.10. HabitanteRepository para operaciones CRUD entre la aplicación Web y la tabla rol.

Los métodos para las operaciones CRUD se adquieren al extender la interfaz CrudRepository. La conexión o mapeo con las entidades se definen en los corchetes angulares <, >

Como utilitario se programó una aplicación java para encriptar una cadena de caracteres tal y como se requiere en la clase WebSecurityConfig, cabe mencionar que este programa es independiente de la aplicación Web. El código es el siguiente

```
1 public class Passgenerator {
2
3   public static void main(String [] args) {
4     BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder
      (4);
5
6     //El String que mandamos al metodo encode es el password que queremos
      encriptar
7     System.out.println(bCryptPasswordEncoder.encode("1234"));
8   }
9 }
```

Listing 1.11. Encriptador de contraseñas.

Las implementaciones de las vistas no se presentan en este documento ya que no aportan información útil ya que únicamente se hacen acciones en el lenguaje de thymeleaf, que refieren a la clase correspondiente al controlador de la aplicación que se definió en lineas anteriores. Además si se muestran, este documento tendría una extensión mucho mayor.

3. Resultados

Las operaciones de consulta, alta, baja y modificación de valores, de la tabla habitante así como la asignación de roles a cada usuario son operaciones funcionales y además se implementó correctamente las validaciones para cada campo que lo requirió. Finalmente se implementaron correctamente las vistas mediante archivos HTML.
El mapa de navegación de nuestra aplicación web se muestra en la figura 2

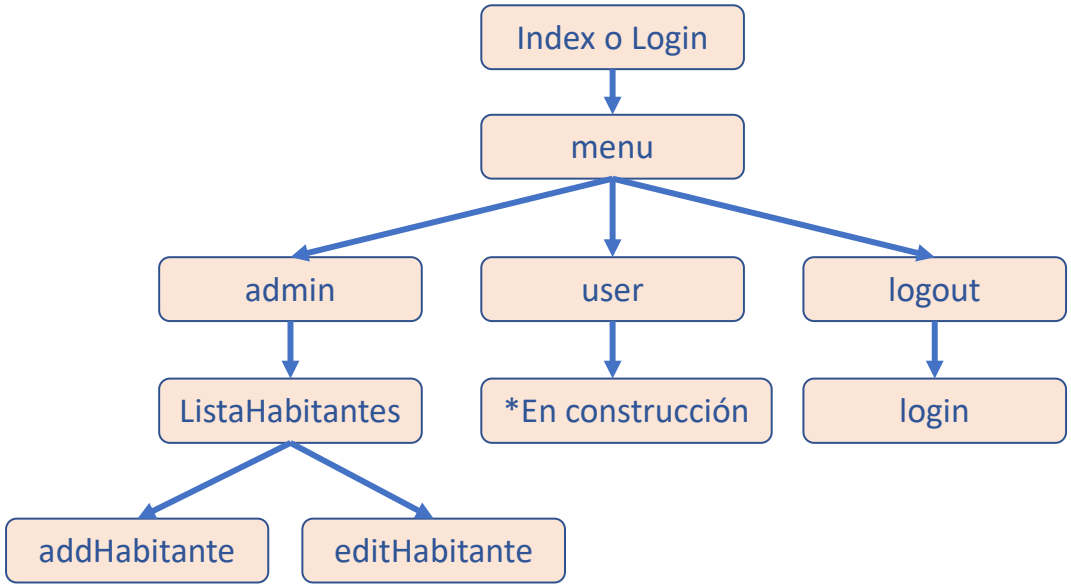

Figura 2. Mapa de navegación de la aplicación web.

Las figuras 3 a 10 muestran las vistas correspondientes a la aplicación web. La descripción de las opciones disponibles en cada vista se detalla en el pie de figura de cada una de las vistas.

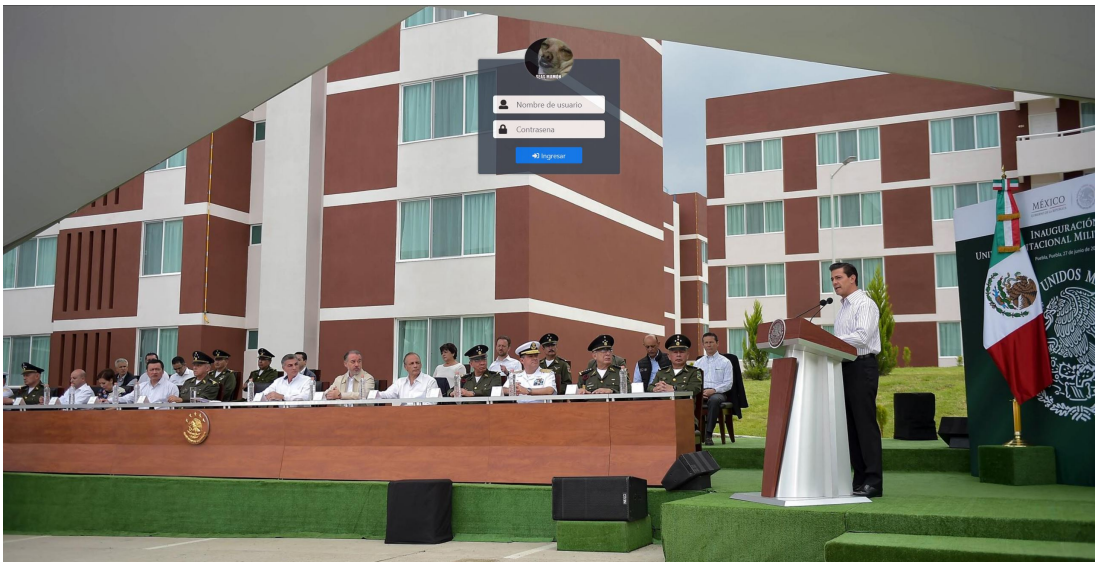

Figura 3. Login o index. Aquí se solicita el nombre de usuario o username y su contraseña. Si un habitante que no está registrado en la base de datos intenta entrar, no se le permitirá el acceso, esto debido a la clase WebSecurityConfig.

Figura 4. Menú principal. Los botones Pagina de administrador, pagina habitante y Cerrar sesión redireccionan a las URL /admin , /user y /login respectivamente

Figura 5. Pagina habitante.

Figura 6. Pagina administrador. Los botones ir al menú, Todos los habitantes y cerrar sesión redireccionan a las URL /menu, /listahabitantes y /login respectivamente

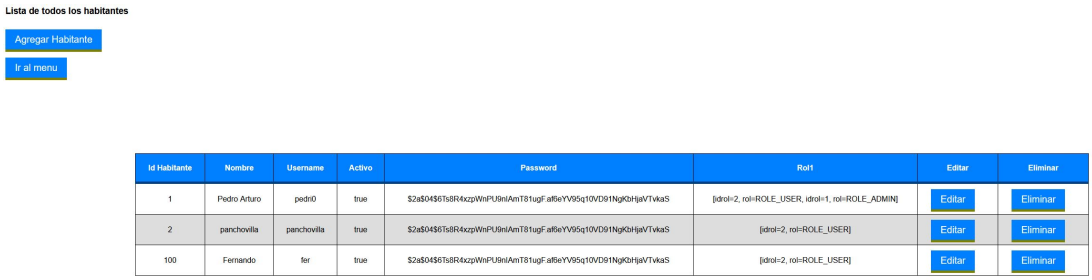

Figura 7. Lista Habitantes. Los botones Agregar habitante, Editar, Ir al menú redireccionan a las URL /addHabitante, /editHabitane, /menu. Mientras que si se acciona el botón eliminar para cualquier habitante, este se eliminará de la base de datos y se rediccionara al /menu

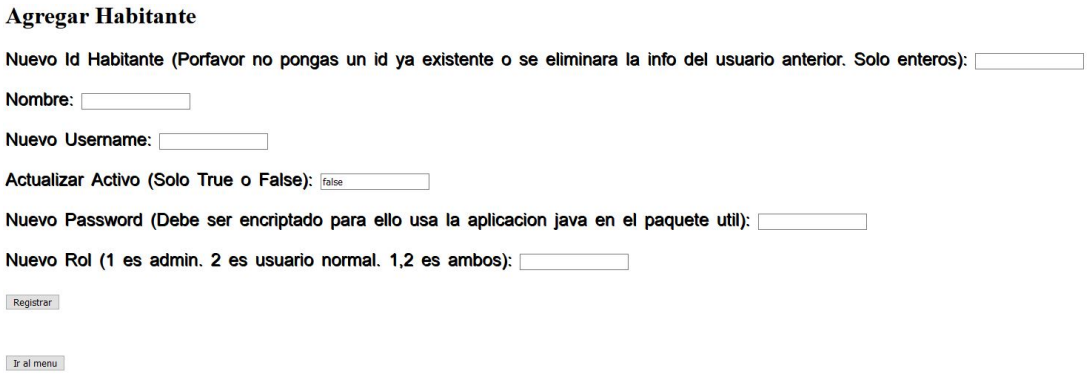

Figura 8. Agregar habitante. Aquí se muestran los campos de la tabla habitante de la base de datos para una nueva instancia de la entidad Habitante. El usuario puede agregar cualquier tipo de texto en las cajas de texto, sin embargo, debido a las validaciones puede que se redireccione al URL /error. Por ejemplo si en la caja de texto id Habitante se pone un carácter del tipo no entero y se aprieta el botón registrar, entonces se rediccionará hacia la pagina error.

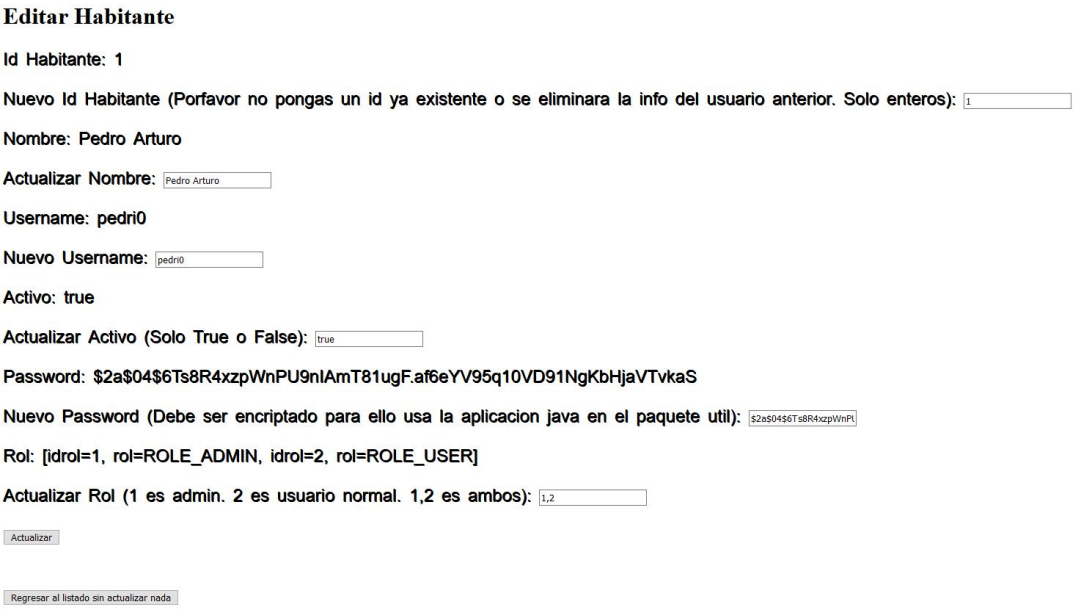

Figura 9. Editar Habitante. Aquí se muestran los campos de la tabla habitante de la base de datos para un habitante con un determinado identificador. El usuario puede agregar o modificar cualquier tipo de texto en las cajas de texto, sin embargo, debido a las validaciones puede que se redireccione al URL /error. Por ejemplo si en la caja de texto id Habitante se pone un carácter del tipo no entero y se aprieta el botón registrar, entonces se rediccionará hacia la pagina error.

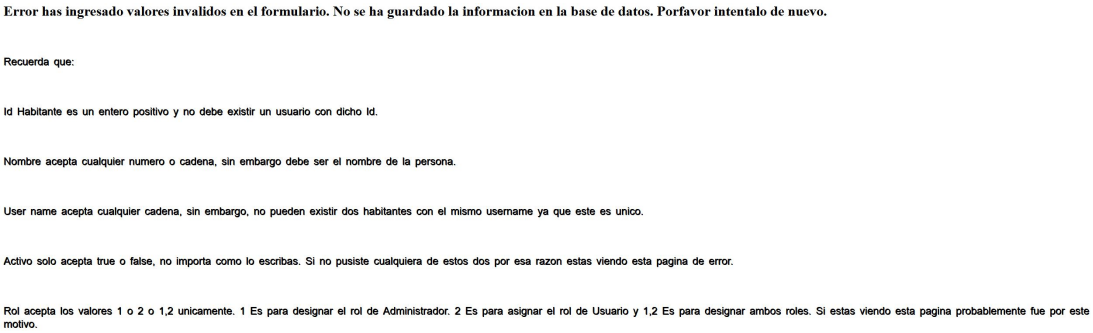

Figura 10. Pagina de error.

4. Conclusiones

Se logró implementar una aplicación web la cual requiere de un login para poder acceder a la pagina de usuario y/o administrador de acuerdo a los privilegios del usuario. Para un usuario con privilegios de administrador es posible administrar la base de datos. La administración de esta base

de datos consiste en la posibilidad de consultar, agregar, cambiar y eliminar los datos correspondientes a la tabla habitante.
Si bien es cierto que la base de datos tal y como se presentó no fue implementada en su totalidad, la aplicación web con sus funcionalidades presentada en este trabajo es la base para la implementación total la base de datos. Pudiendo lograr, de esta manera, un sistema de votación totalmente electrónico. Adicionalmente es posible extender la funcionalidad de la aplicación web, una vez ya terminado el sistema de votación, de tal forma que muestre estadísticas de cada votación empleando la información que provee la tabla categoría. Finalmente, a pesar de no haber cumplido con el objetivo de construir un sistema electrónico de votación el objetivo se cumplió parcialmente ya que fue posible implementar un sistema de login con funcionalidades de consulta, moificacion, adición y eliminación de los datos en la tabla habitante de la base de datos propuesta.

Referencias

1. Emmerich, G.: Las elecciones de 2006 y su impacto sobre la democracia en México. El cotidiano **22**(145), 05–15 (2006)
`https://www.redalyc.org/pdf/325/32514502.pdf`. Recuperado el 19 de Noviembre de 2019.