



WareControl

ALGO ORDENADO

David Pinchete Nebreda
Alcalá de Henares, 7 jun 2024

INTRODUCCIÓN

Hace unos años el diseñador de la aplicación observó el claro problema presente en la vivienda de sus progenitores, donde no tenían conocimiento de lo que disponían ni de donde se encontraba. Allí es donde vio la posibilidad de la creación de un aplicativo que ayude en la organización y control de los objetos cotidianos de su familia, así es como surgió el desarrollo de WareControl.

IDEA DEL PROYECTO

La idea es diseñar una aplicación móvil y web donde los usuarios puedan conocer la situación de almacenaje de los productos y su información, pudiendo encontrar cada elemento guardado en el almacén mediante consultas, listar todos los elementos o escanear ubicaciones de almacenaje y ver su contenido.

OBJETIVOS

El objetivo de la aplicación es ayudar a los usuarios en la gestión de los elementos almacenados en su trastero u otro lugar de almacenaje haciéndoles más fácil la localización y la consulta de los mismos.

Para ello la aplicación utiliza un sistema de escaneo mediante QR que facilita el ingreso de cada objeto en la localización correcta.

El mismo QR de cada localización permite listar los objetos u otras localizaciones contenidas en él.

También aporta al usuario la búsqueda rápida de un objeto o localización mediante palabras clave que él pueda recordar, por ejemplo buscando todos los objetos que sean “leche” y pudiendo observar donde está localizado cada objeto.

FASES DE PROYECTO

- Diseño de Arquitectura de software.
- Diseñar base de datos relacional en MySql.
- Maquetar app móvil/web.
- Desarrollo api de negocio.
- Desarrollar app móvil con Android Studio.
- Desarrollar app web con eclipse y java EE.
- Realizar pruebas de manera local.
- Crear arquitectura de servidores.
- Desplegar aplicaciones.

TECNOLOGÍAS UTILIZADAS

Base de datos en MySql.



MySQL es un sistema de gestión de bases de datos relacional con licencia pública general y está considerada como la base de datos de código abierto más popular del mundo.
Es la encargada de almacenar toda la información de nuestra aplicación.

Java



Elemento base de nuestro desarrollo en la aplicación.

Es un lenguaje orientado a objetos, multiplataforma y además con alternativas gratuitas como openJdk, adicionalmente utilizamos J2EE que es una librería estandarizada que complementa las funciones básicas de Java base.

SpringBoot



Spring es un Framework de desarrollo que nos aporta una robustez y versatilidad a la hora de desarrollar nuestra app.

En nuestra aplicación se utiliza tanto en el componente Backend integrando una Api Rest, como en una aplicación adicional web, proporcionando así el componente multiplataforma.

Thymeleaf



Thymeleaf es una biblioteca Java que implementa un motor de plantillas de XML/XHTML/HTML5 que puede ser utilizado en modo web. Se acopla muy bien para trabajar en la capa vista del MVC de aplicaciones web. Utilizado en nuestra aplicación en el desarrollo de las vistas en el aplicativo web facilitandonos el manejo de la información del controlador a la vista del usuario.

Bootstrap



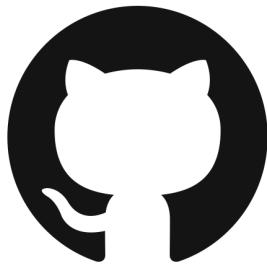
Es una librería de diseño de interfaces web el cual facilita el trabajo en esta materia haciéndola visualmente más amigable para el usuario.

Android Studio.



Es un IDE de desarrollo de aplicaciones para los sistemas operativos Android. Es el programa por excelencia para este tipo de desarrollos ya que está hecho por la misma empresa que diseñó el sistema operativo Android, esto proporciona unas herramientas integrales.

GitHub



Es una plataforma integrada con git utilizada para la gestión y control de versiones del código y otros elementos que estamos realizando

PostMan

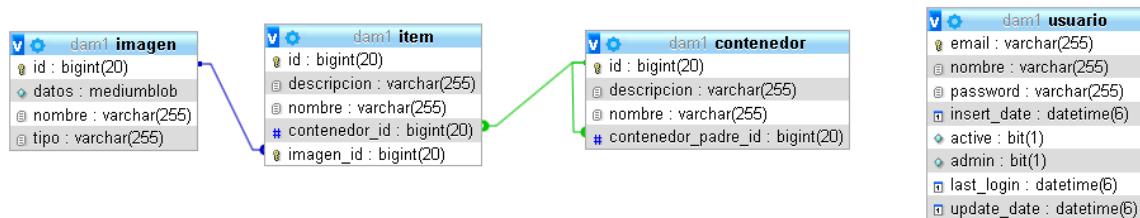


Es una herramienta focalizada en la realización de pruebas de aplicaciones tipo Api Rest. Esta herramienta nos ayuda también a cumplir con los estándares de las peticiones HTTP, sus métodos y sus estándares de autenticación.

BASE DE DATOS

Como ya comentamos anteriormente, se utiliza MySQL como administrador de base de datos.

En ella se almacenan los usuarios que gestionan el almacén como los contenedores que conforman los mismos y los elementos guardados en cada contenedor.



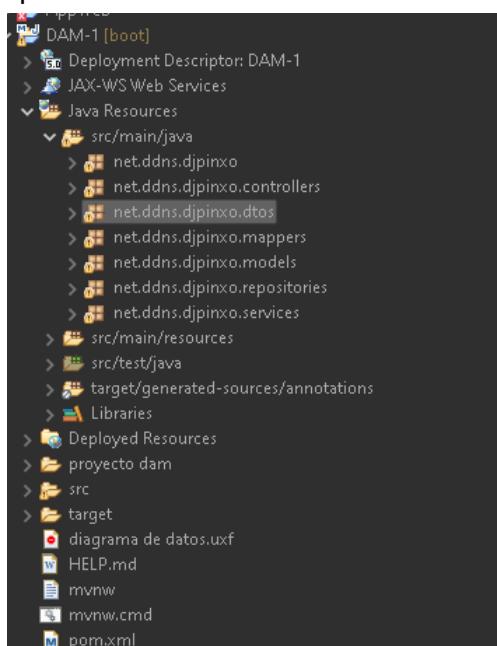
La imagen pertenece al esquema de base de datos donde podemos apreciar por un lado los usuarios que serán capaces de acceder y utilizar nuestra aplicación.

Por otro lado tenemos los verdaderos protagonistas de la aplicación que son los contenedores que tiene asociación con un contenedor padre y los ítems asociados al contenedor.

Por último la tabla **imagen** es la utilizada para guardar los datos de la imagen asociada a los ítems.

API REST SPRING (BACKEND)

Para el desarrollo de la parte backend nos hemos basado en MVC para implementar la arquitectura de clases. Como en este caso, ya que es una aplicación backend, solo tenemos controladores y modelos, a los que se suman la implementación de persistencia de datos y algunas clases sueltas para la gestión de seguridad y la configuración de la aplicación.



Entity

Para comenzar, la parte más importante son las entidades con las que cuenta la aplicación, los llamados modelos.

Spring la forma que tiene de distinguirlos es mediante la etiqueta `@Entity`, adicionalmente utilizamos las anotaciones de JPA para la incorporación automatizada de la persistencia de estas entidades.

```
package net.doms.djpinxo.models;

import java.util.List;

@Entity
public class Contenedor {

    @Id
    @GeneratedValue
    private Long id;
    private String nombre;
    private String descripcion;
    @ManyToOne
    private Contenedor contenedorPadre;
    @OneToMany(mappedBy = "contenedorPadre")
    @JsonIgnore
    private List<Contenedor> contenedorHijos;
    @OneToMany(mappedBy = "contenedor")
    @JsonIgnore
    private List<Item> items;

    public Contenedor() {
        super();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }
}
```

Repository

Para realizar la implementación de la persistencia de datos, nos basamos en la librería de Spring JPA para implementar las diferentes clases de manejo.

Inicialmente se extendió de la clase CrudRepository, pero a medida que fue avanzando el proyecto descubrimos que la clase JpaRespository era más completa que la anterior ya que heredaba de ella y se decidió cambiar a esta última.

Por último utilizamos la potencia de esta librería para implementar nuestras consultas personalizadas basándose en su escritura directamente sin necesidad de implementar una query sql.

```
import java.util.List;

public interface ContenedorRepository extends JpaRepository<Contenedor, Long> {

    List<Contenedor> findByNombreContainsIgnoreCaseOrDescripcionContainsIgnoreCase(String query, String query2);
}
```

Controller

A la hora de desarrollar nuestro controlador, Spring nos provee de una anotación específica de la arquitectura rest llamado @RestController.

Por otro lado, se ha implementado cada uno de los métodos de HTTP para cada una de las operaciones que gestionan las entidades.

```
@RestController
public class ContenedorController {

    @Autowired
    ContenedorRepository contenedorRepository;
    //ContenedorService contenedorService;

    @GetMapping("/contenedores")
    public List<Contenedor> getContenedores(@RequestParam(required = false) String query) {
        if (query==null) {
            return contenedorRepository.findAll();
        }
        else {
            return contenedorRepository.findByNombreContainsIgnoreCaseOrDescripcionContainsIgnoreCase(query, query);
        }
    }

    @GetMapping("/contenedores/{id}")
    public Optional<Contenedor> getContenedor(@PathVariable Long id) {
        return contenedorRepository.findById(id);
    }

    @PostMapping("/contenedores")
    public Optional<Contenedor> postContenedor(@RequestBody Contenedor contenedor) {
        Optional<Contenedor> contenedorSaved = contenedorRepository.findById(contenedor.getId());
        if (contenedorSaved.isEmpty()) {
            return Optional.of(contenedorRepository.save(contenedor));
        }
        else return Optional.empty();
    }

    @PutMapping("/contenedores/{id}")
    public Optional<Contenedor> putContenedor(@RequestBody Contenedor contenedor, @PathVariable Long id) {
        Optional<Contenedor> contenedorSaved = contenedorRepository.findById(id);
        if (contenedorSaved.isPresent()) {
            contenedorSaved.get().setNombre(contenedor.getNombre());
            contenedorSaved.get().setDescripcion(contenedor.getDescripcion());
            contenedorSaved.get().setContenedorPadre(contenedor.getContenedorPadre());
            //contenedor.setId(id);
            contenedorSaved = Optional.of(contenedor);
            contenedorRepository.save(contenedor);
        }
    }
}
```

Adicional a los controladores que gestionan el estándar CRUD, se han implementado algunas gestiones adicionales para darle cabida a la gestión del login y el registro.

```
@RestController
public class LoginController {
    @Autowired
    UsuarioRepository usuarioRepository;

    @PostMapping("/login")
    public ResponseEntity<User> login(@RequestBody User user) {
        if(user==null || user.getEmail()==null || user.getEmail().isEmpty() || user.getPassword()==null || user.getPassword().isEmpty()) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Unauthorized: Missing email or password");
        }
        Optional<User> userSaved = usuarioRepository.findById(user.getEmail());
        if (!userSaved.isPresent() || userSaved.get().getPassword()==null || userSaved.get().getPassword().isEmpty() || !userSaved.get().getPassword().equals(user.getPassword())) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Unauthorized: Missing email or password");
        }

        //procesa de cambio a la nueva ultima fecha
        Date localDateLastLogin = userSaved.get().getLastLogin();
        userSaved.get().setLastLogin(new Date());
        usuarioRepository.save(userSaved.get());

        userSaved.get().setLastLogin(localDateLastLogin);
        //userSaved.get().setPassword(null);
        return ResponseEntity.ok(userSaved);
    }

    @PostMapping("/register")
    public Optional<User> register(@RequestBody User user) {
        Optional<User> userSaved = usuarioRepository.findById(user.getEmail());
        if (userSaved.isEmpty()) {
            //user.setActive(false);
            user.setInsertDate(new Date());
            return Optional.of(usuarioRepository.save(user));
        }
        else return Optional.empty();
    }
}
```

Spring Security

Por último se ha implementado un filtro por el cual manejaremos la seguridad de los recursos entregados por la aplicación, esto se ha realizado mediante Spring Security.

```
12  @Bean
13  public PasswordEncoder passwordEncoder() {
14      return NoCasePasswordEncoder.getInstance();
15  }
16
17  @Bean
18  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
19      return http
20          .httpBasic()
21          .and().authorizeHttpRequests()
22          .requestMatchers("/login", "/register").permitAll()
23          .anyRequest().authenticated()
24          .and().csrf().disable()
25          .build();
26  }
27
28  }
29
30
31  @Service
32  class UserDetailServicePersonalizado implements UserDetailsService{
33
34  @Autowired
35  private UsuarioRepository usuarioRepository;
36
37  @Override
38  public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
39      Optional<User> user = usuarioRepository.findById(email);
40      if (user.isPresent()) {
41          return new UserDetailPersonalizado(user.get());
42      }
43      throw new UsernameNotFoundException("Usuario no encontrado: " + email);
44  }
45
46  }
47
48
49  class UserDetailPersonalizado implements UserDetails{
50
51      private User user;
52
53      public UserDetailPersonalizado(User user) {
54          this.user=user;
55      }
56
57  @Override
```

APLICACIÓN MÓVIL

Una de las capas del usuario es la aplicación móvil. Esta es la que toma mayor relevancia en el proyecto, ya que es la más cercana al usuario y la que mayor funcionalidad proporciona debido a los recursos hardware del terminal.

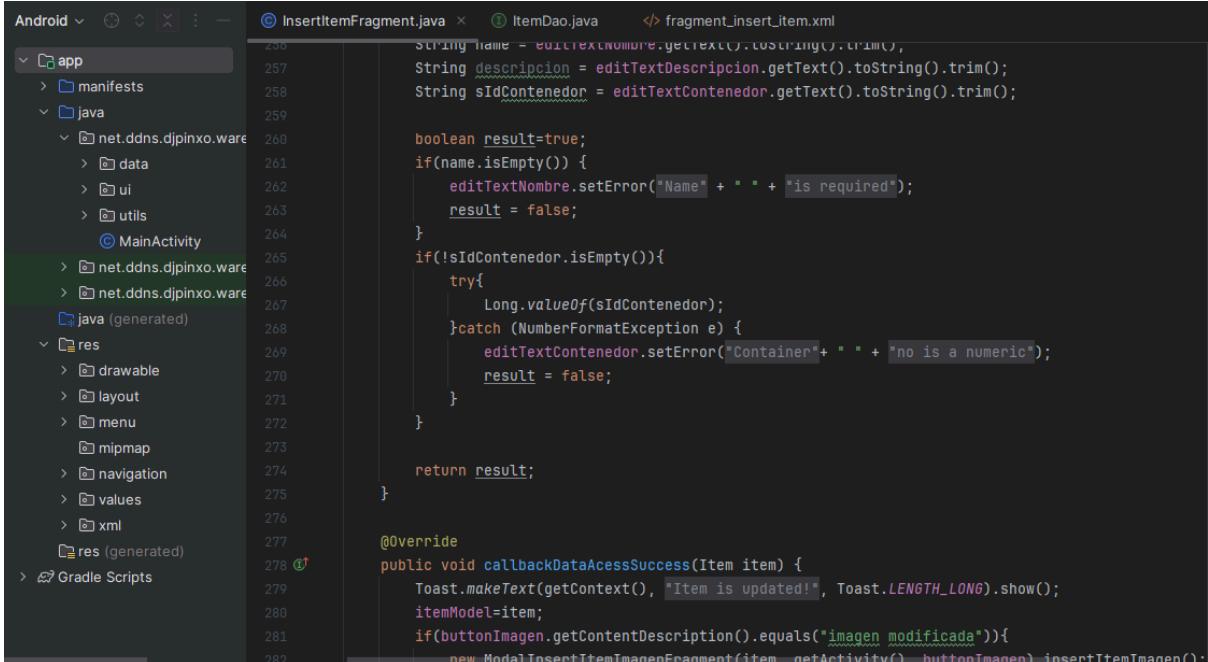
El aplicativo está concebido para ser usado de forma rápida y cómoda para los usuarios, esto es gracias a la posibilidad de escanear códigos QR para localizar y organizar objetos. También posibilita la toma de una foto del objeto en cuestión.

La aplicación móvil también se basa en el modelo MVC.

Tenemos 3 grandes áreas de gestión, los objetos, los contenedores y los usuarios, estos últimos gestionados únicamente por usuarios administradores.

Toda comunicación realizada a la persistencia de datos es realizada mediante la librería retrofit y su conversor gson.

Zxing es la librería para la generación de QR mientras que la librería de lectura de los mismos es la MLkit.



The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The project structure includes modules like app, manifests, java, res, and Gradle Scripts. The code editor displays Java code for `InsertItemFragment.java`, specifically the `callbackDataAccessSuccess` method which handles item updates and triggers a modal fragment for image insertion.

```
String nombre = editTextNombre.getText().toString().trim();
String descripcion = editTextDescripcion.getText().toString().trim();
String sIdContenedor = editTextContenedor.getText().toString().trim();

boolean result=true;
if(name.isEmpty()) {
    editTextNombre.setError("Name" + " " + "is required");
    result = false;
}

if(!sIdContenedor.isEmpty()){
    try{
        Long.valueOf(sIdContenedor);
    }catch (NumberFormatException e) {
        editTextContenedor.setError("Container" + " " + "no is a numeric");
        result = false;
    }
}

return result;
}

@Override
public void callbackDataAccessSuccess(Item item) {
    Toast.makeText(getContext(), "Item is updated!", Toast.LENGTH_LONG).show();
    itemModel=item;
    if(buttonImagen.getContentDescription().equals("imagen modificada")){
        new ModalInsertItemImagenFragment(item, getActivity(), buttonImagen).insertItemImagen();
    }
}
```

CASOS DE USO

Registro de usuario:

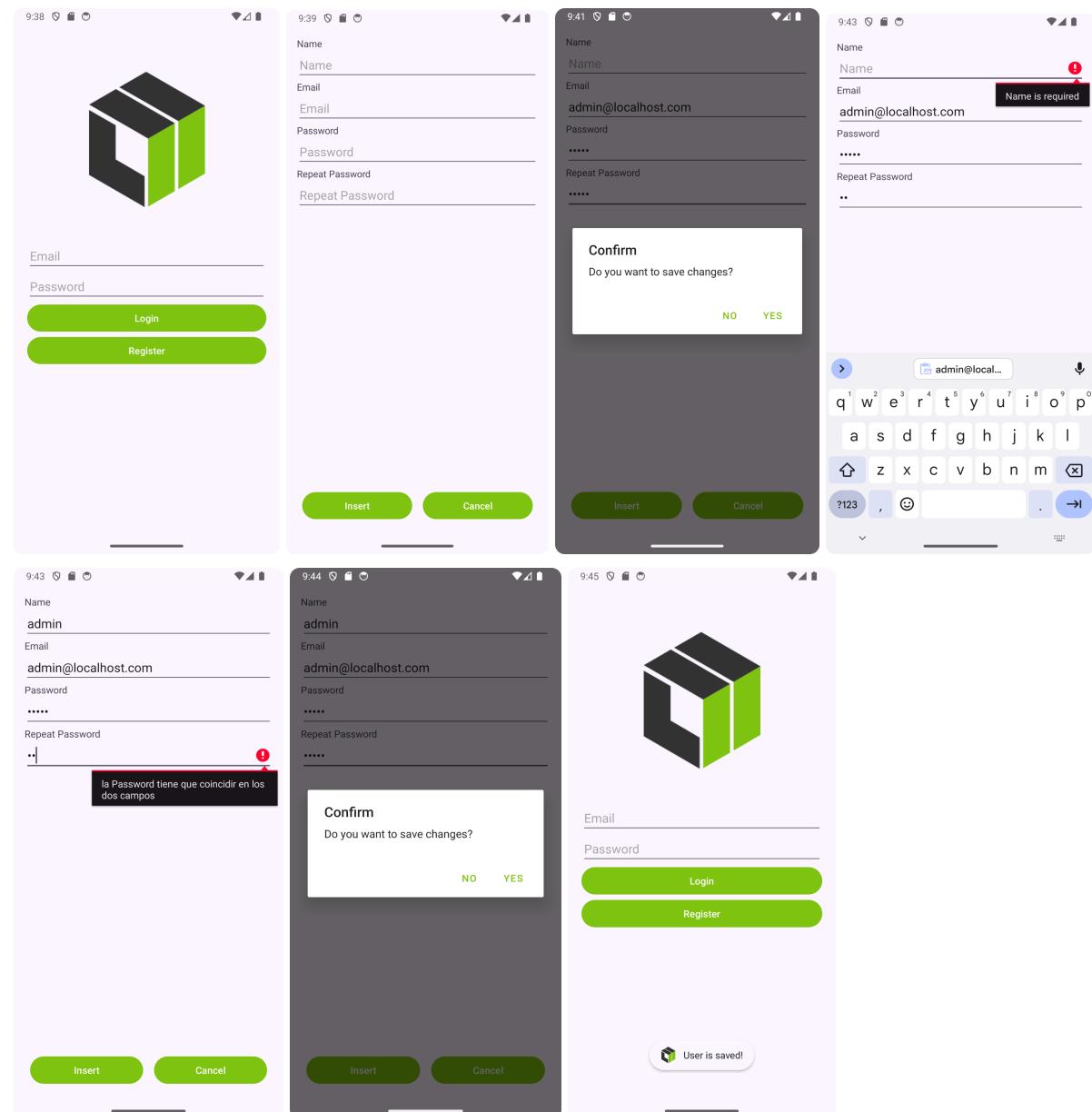
El usuario se puede registrar en la aplicación introduciendo el nombre, el correo electrónico y la contraseña.

Internamente se usa el algoritmo de hash SHA-256 para encriptar la contraseña y se comprueba que el email no está registrado en la aplicación.

Si hay algún problema con los campos la aplicación validará y notificará de los mismos al usuario.

Una vez realizado el registro con éxito se notificará al usuario mediante notificación emergente, llevándolo de nuevo a la ventana de login.

Por último el usuario quedará por defecto inaccesible hasta que un usuario administrador le conceda permiso a la aplicación.



Esta gestión es realizada por el controller login.

```

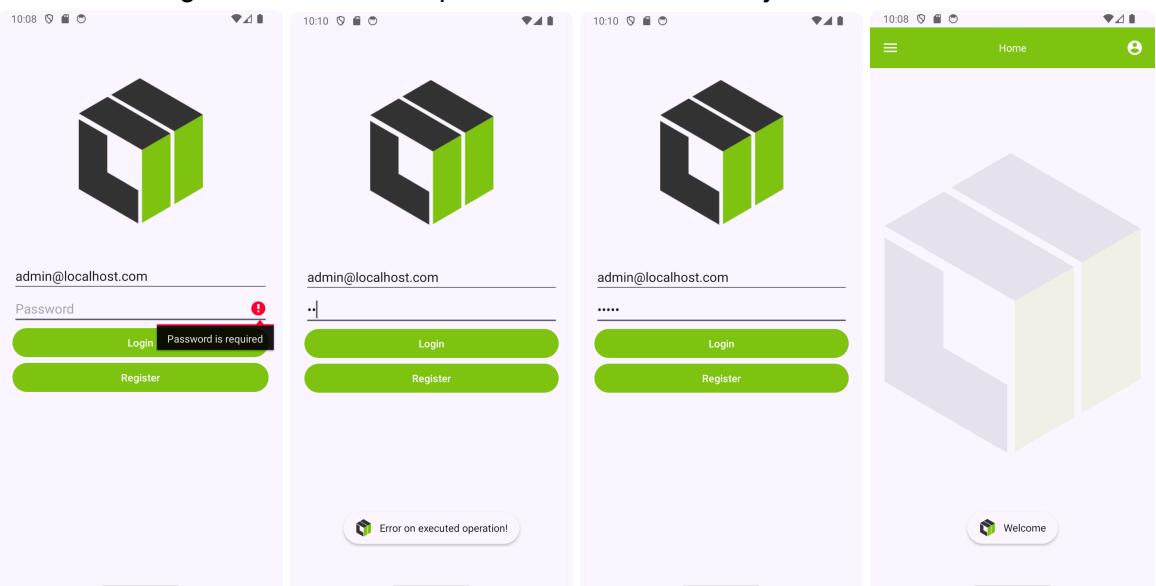
@PostMapping("/register")
public Optional<Usuario> register(@RequestBody Usuario usuario) {
    Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuario.getEmail());
    if (usuarioSaved.isEmpty()) {
        //usuario.setActive(false);
        usuario.setInsertDate(new Date());
        return Optional.of(usuarioRepository.save(usuario));
    }
    else return Optional.empty();
}

```

Login de usuario:

Para poder acceder a la aplicación se solicitará el correo electrónico y su contraseña, esta será hasheada en SHA-256 y enviada al servidor.

De nuevo si algún dato faltara el aplicativo mostrará mensajería informativa.



Para poder acceder a la aplicación será necesario que el usuario esté en estado activo en la base de datos.

admin@localhost.com	admin	8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a8...	2024-06-06 23:45:07.000000	1	0	2024-06-06
---------------------	-------	---	----------------------------	---	---	------------

Una vez el paso de login sea correcto, los datos del usuario se almacenarán en una variable interna para su posterior utilización en la navegación.

Esta gestión es realizada por el controller login..

```

@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody Usuario usuario ) {
    if(usuario==null || usuario.getEmail()==null || usuario.getEmail().isEmpty() || usuario.getPassword()==null || usuario.getPassword().isEmpty()) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Unauthorized: Missing email or password");
    }
    Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuario.getEmail());
    if (usuarioSaved.isEmpty() || usuarioSaved.get().getPassword()==null || usuarioSaved.get().getPassword().isEmpty() || !usuarioSaved.get().getPassword().equals(usuario.getPassword())) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Unauthorized: Missing email or password");
    }

    //proceso de cambio a la nueva ultima fecha
    Date localDateLastLogin = usuarioSaved.get().getLastLogin();
    usuarioSaved.get().setLastLogin(new Date());
    usuarioRepository.save(usuarioSaved.get());

    usuarioSaved.get().setLastLogin(localDateLastLogin);
    //usuarioSaved.get().setPassword(null);
    return ResponseEntity.ok(usuarioSaved);
}

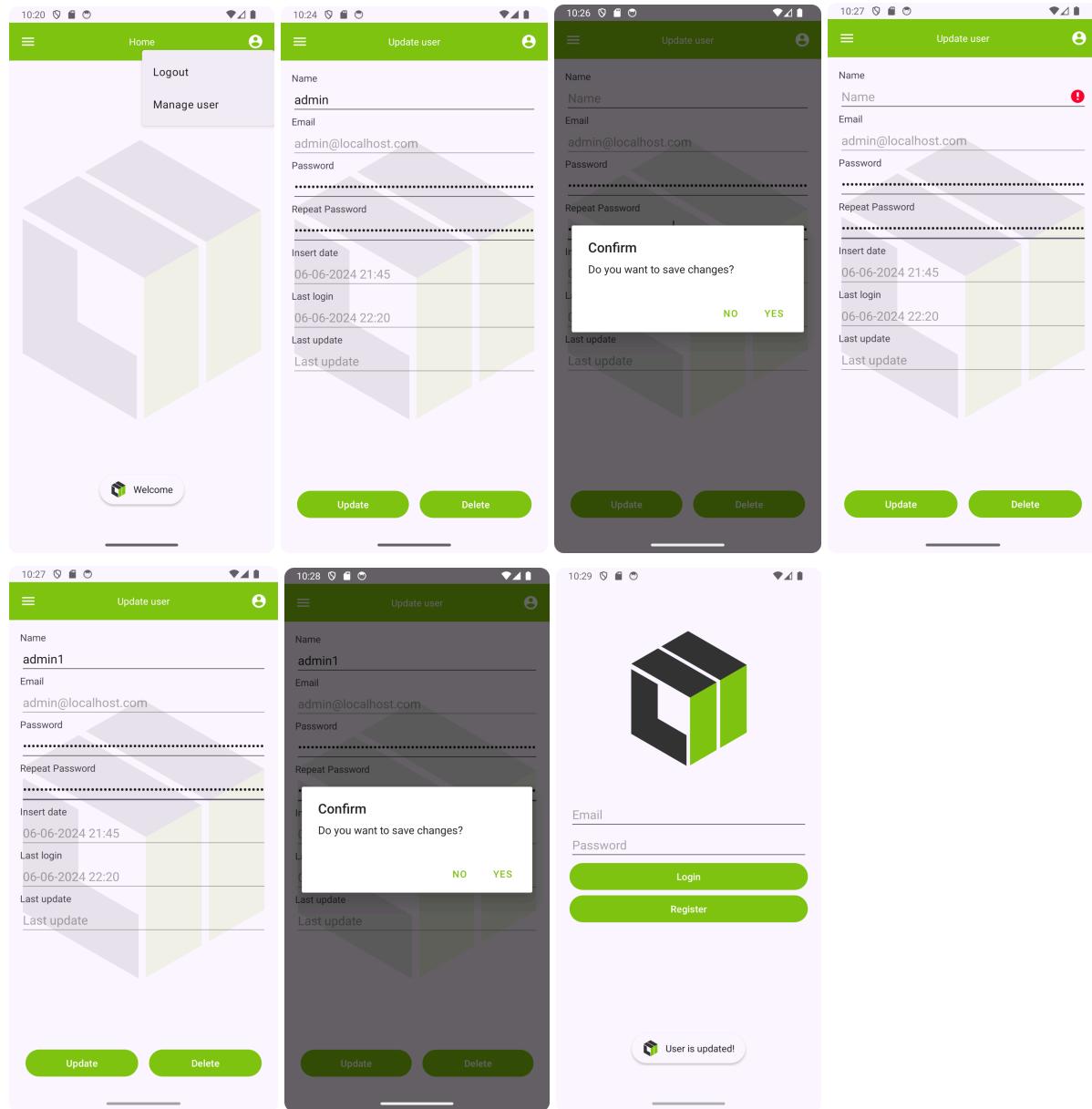
```

Modificación de usuario logado

Una vez logueado, desde cualquier punto de la aplicación podemos acceder a la gestión de nuestro usuario.

En la misma tendremos dos operativas, en este caso la actualización del usuario, los datos del usuario modificables por uno mismo es el nombre y la contraseña.

Una vez completado el cambio nos retorna al login de nuevo y actualiza en el usuario la fecha del cambio.



Esta gestión es realizada en el mismo controller que la operativa general de usuarios.

```

@PutMapping("/usuarios/{email}")
public ResponseEntity<Optional<Usuario>> putUsuario(@RequestHeader("Authorization") String authorizationHeader, @RequestBody Usuario usuario, @PathVariable String email) {
    if(validarUsuario(authorizationHeader) || decodeAuthorizationHeader(authorizationHeader).getEmail().equals(email)) {
        Optional<Usuario> usuarioSaved = usuarioRepository.findById(email);
        if (!usuarioSaved.isEmpty()) {
            usuario.setEmail(email);
            usuario.setUpdateDate(new Date());
            if(permiteSetAdmin(authorizationHeader, usuarioSaved.get())) {
                usuario.setAdmin(false);
            }
            usuarioSaved = Optional.of(usuarioRepository.save(usuario));
        }
        return ResponseEntity.ok().body(usuarioSaved);
    } else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}

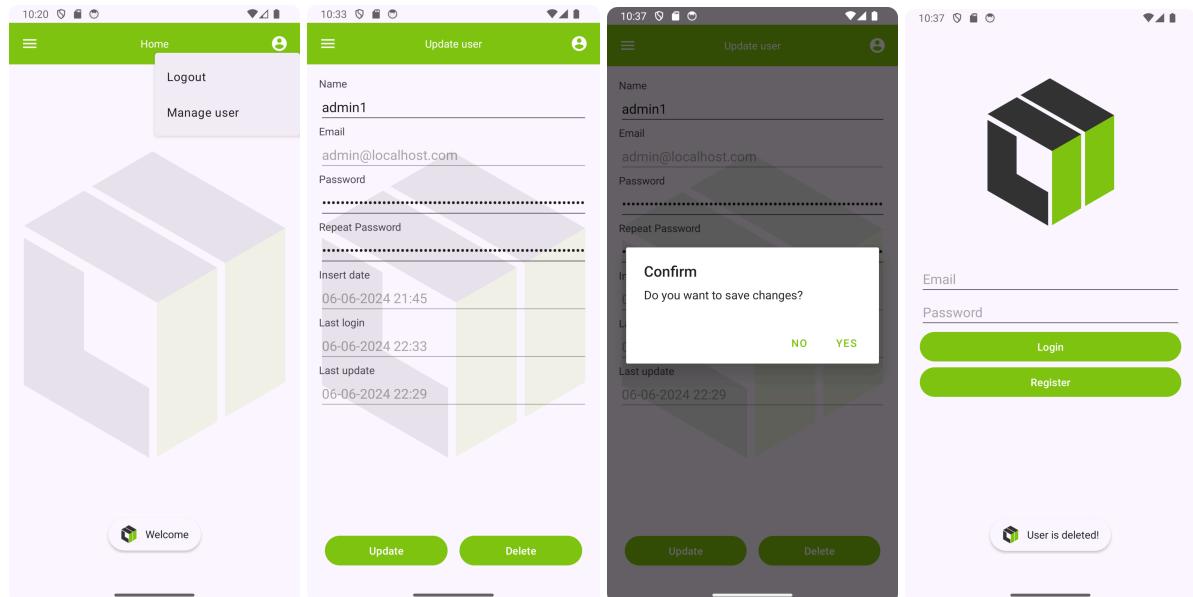
```

Eliminación del usuario logado:

Una vez logueado, desde cualquier punto de la aplicación podemos acceder a la gestión de nuestro usuario.

En la misma tendremos dos operativas, en este caso el borrado del usuario.

Una vez completado el borrado nos retorna al login de nuevo quedando el usuario no accesible.



Esta gestión es realizada en el mismo controller que la operativa general de usuarios.

```

@DeleteMapping("/usuarios/{email}")
public ResponseEntity<Void> deleteUsuario(@RequestHeader("Authorization") String authorizationHeader, @PathVariable String email) {
    if(validarUsuario(authorizationHeader) || decodeAuthorizationHeader(authorizationHeader).getEmail().equals(email)) {
        usuarioRepository.deleteById(email);
        return ResponseEntity.ok().body(null);
    } else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}

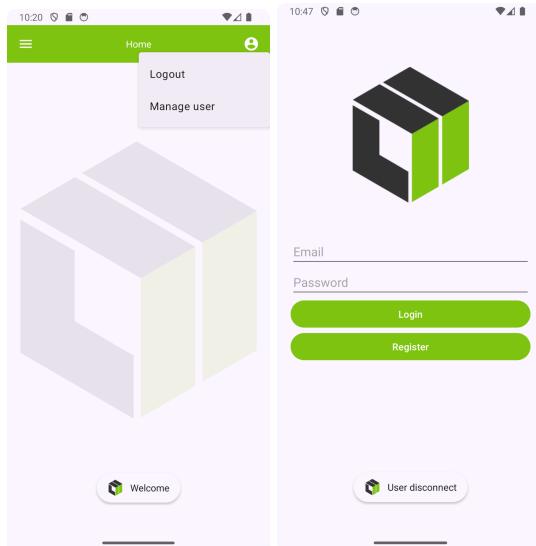
```

Desconectar sesión

Una vez logueado, desde cualquier punto de la aplicación podemos acceder a la desconexión de nuestro usuario.

Esta acción no tiene contraparte en backend, por que las aplicaciones Rest no tienen estado, esto significa que el encargado de mantener la sesión activa es el dispositivo mediante el envío de los datos del usuario peticionario en cada llamada.

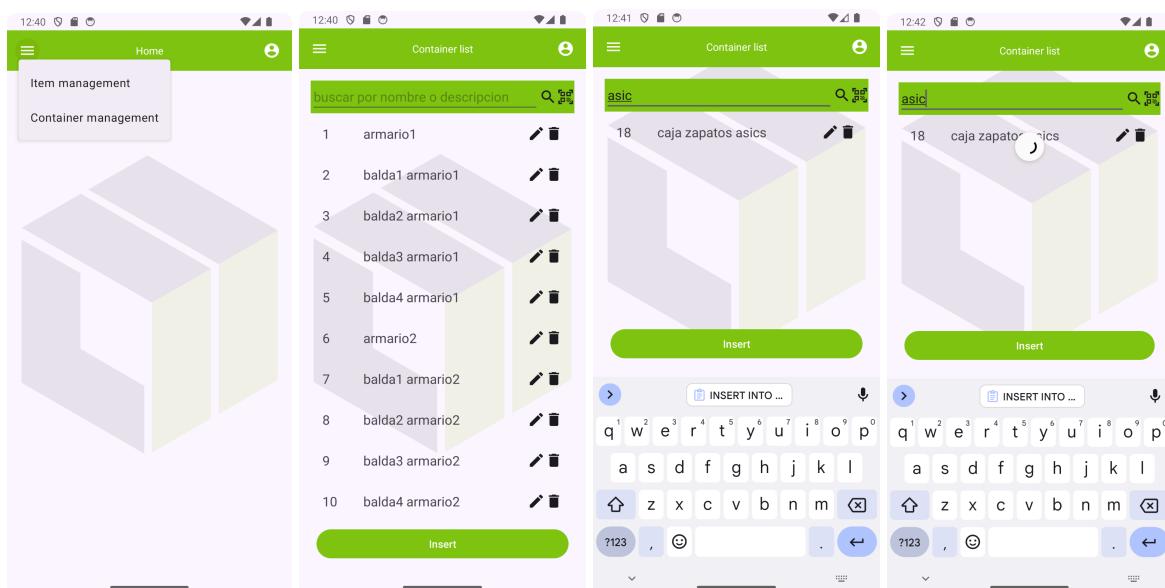
El proceso de desconexión es el borrado en el dispositivo de los datos del usuario.



Listar Contenedores y filtrado de los mismos

Dentro de la aplicación el usuario podrá realizar una búsqueda de los contenedores existentes en la aplicación.

Dentro de la búsqueda el usuario podrá actualizar la búsqueda mediante los botones de búsqueda o deslizando hacia abajo y filtrar por cualquier texto contenido en la información de los contenedores.



Esta gestión es realizada por el controller de contenedores.

```

@GetMapping("/contenedores")
public List<Contenedor> getContenedores(@RequestParam(required = false) String query) {
    if (query==null) {
        return contenedorRepository.findAll();
    }
    else {
        return contenedorRepository.findByNombreContainsIgnoreCaseOrDescripcionContainsIgnoreCase(query, query);
    }
}

```

Información del contenedor

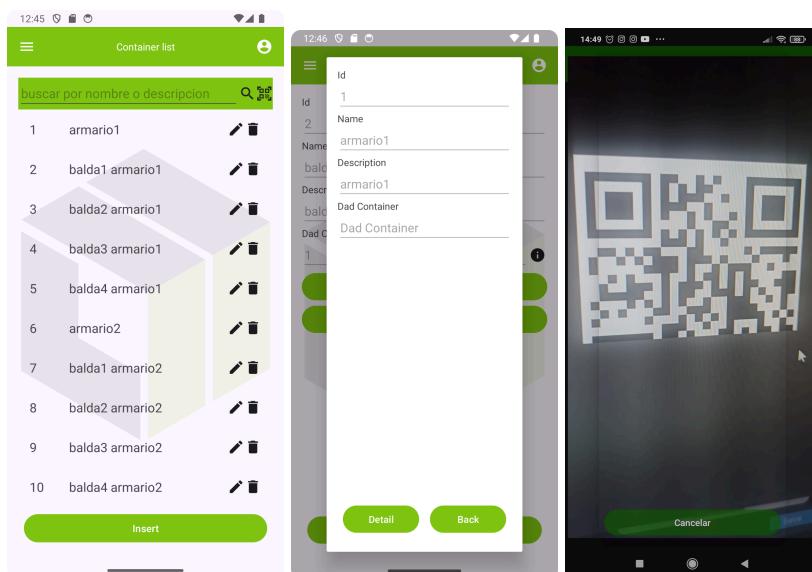
Dentro de la aplicación el usuario podrá realizar una consulta de un contenedor existente en la aplicación.

El acceso a la pantalla de consulta del contenedor se realizará desde varios puntos.

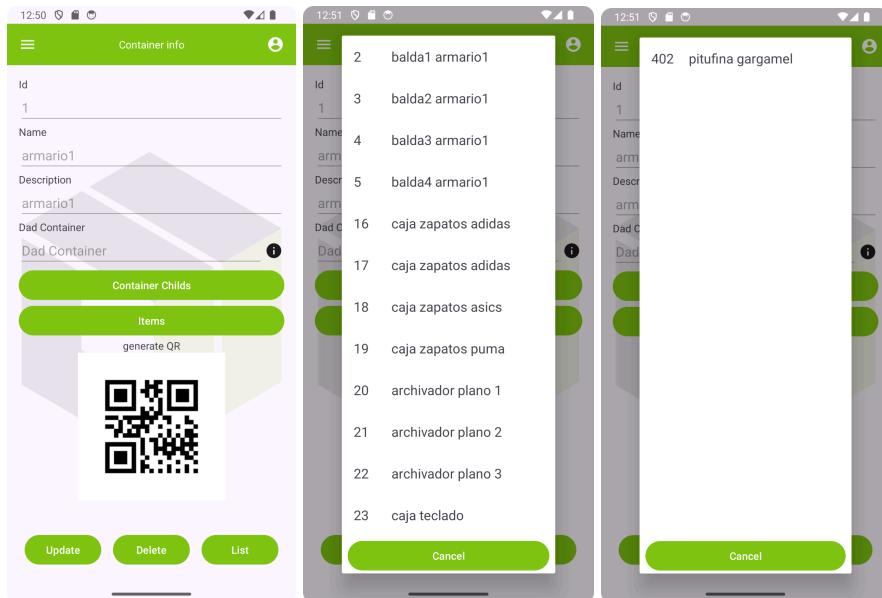
El primero desde la búsqueda listada de contenedores podrá seleccionar un registro llevando a esta pantalla.

El segundo desde la consulta de contenedores el usuario podrá escanear un QR de un contenedor accediendo directamente a la consulta.

Por último, desde el botón detalle de las ventanas modales de contenedor.



Dentro de la consulta el usuario podrá actualizar la información deslizando hacia abajo, consultar el id, nombre, descripción y contenedor al que pertenece, la lista de contenedores asociados o ítems guardados en dicho contenedor y por último el QR generado de dicho contenedor.



Esta gestión es realizada por el controller de contenedores.

```
@GetMapping("/contenedores/{id}")
public Optional<Contenedor> getContenedor(@PathVariable Long id) {
    return contenedorRepository.findById(id);
}

@GetMapping("/contenedores/{id}/contenedorhijos")
public List<Contenedor> getContenedorContenedorHijos(@PathVariable Long id) {
    Optional<Contenedor> contenedor = contenedorRepository.findById(id);
    if(!contenedor.isEmpty()) {
        return contenedor.get().getContenedorHijos();
    }
    return null;
}

@GetMapping("/contenedores/{id}/items")
public List<Item> getContenedorItems(@PathVariable Long id) {
    Optional<Contenedor> contenedor = contenedorRepository.findById(id);
    if(!contenedor.isEmpty()) {
        return contenedor.get().getItems();
    }
    return null;
}
```

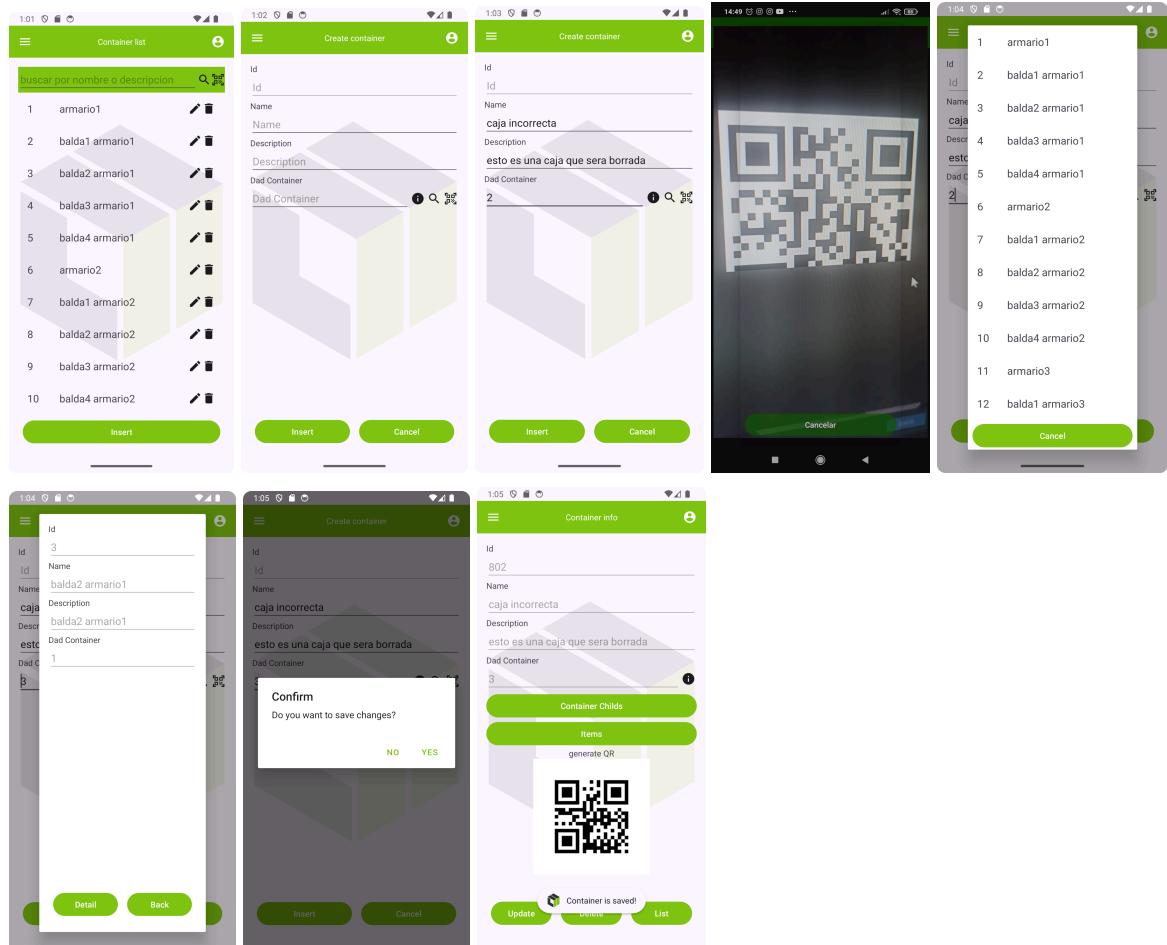
Alta de contenedor

Dentro de la aplicación el usuario podrá realizar un alta de un contenedor.

El acceso a la pantalla de alta del contenedor se realizará desde la búsqueda lista de contenedores desde donde dispone de un botón que le dará acceso al formulario de alta.

Dentro de la pantalla de alta el usuario podrá digitar el nombre, descripción y contenedor al que pertenece, este último puede buscarlo mediante una lista o escaneando directamente el qr del contenedor al que va a pertenecer, también una vez seleccionado puede consultar la información de ese contenedor.

El campo id del contenedor es único autogenerado por la aplicación.



Esta gestión es realizada por el controller de contenedores.

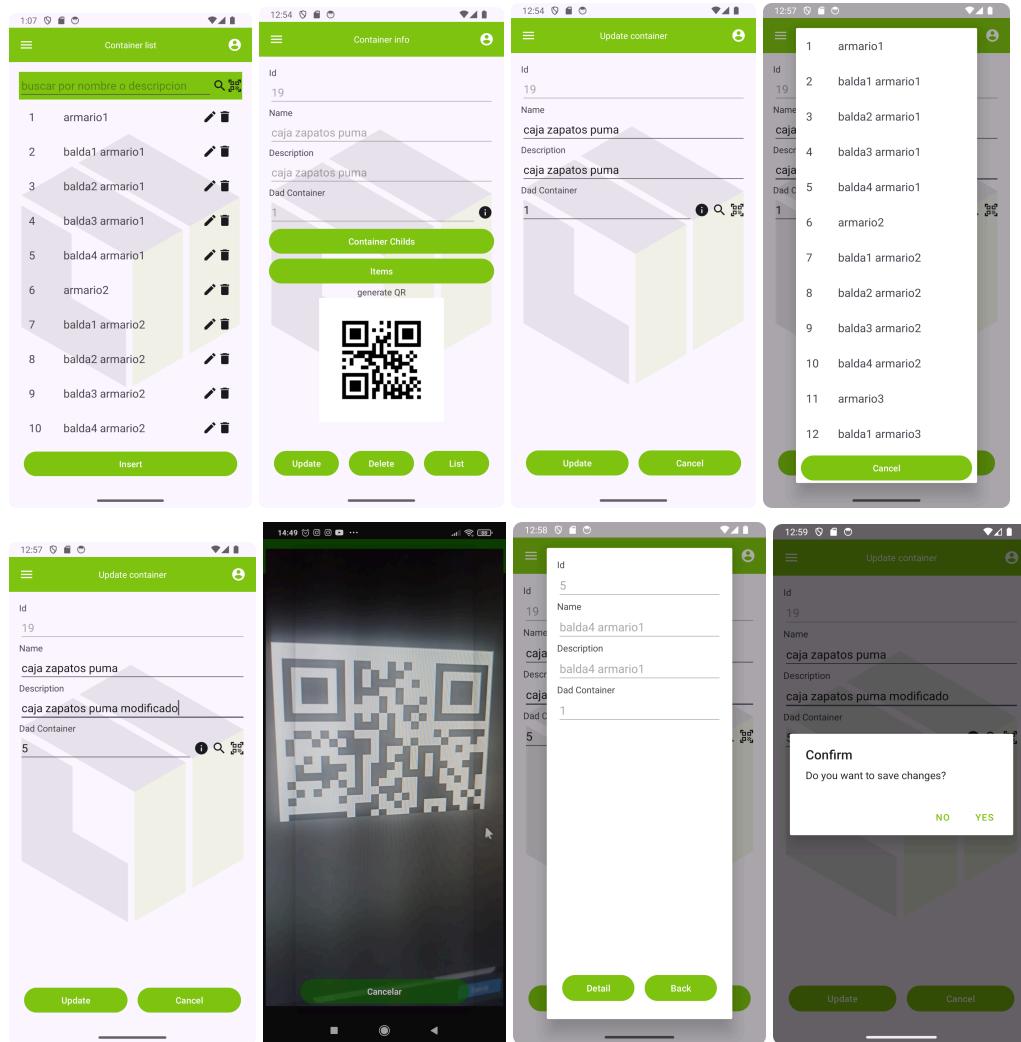
```
@PostMapping("/contenedores")
public Optional<Contenedor> postContenedor(@RequestBody Contenedor contenedor) {
    Optional<Contenedor> contenedorSaved = contenedorRepository.findById(contenedor.getId());
    if (contenedorSaved.isEmpty()) {
        return Optional.of(contenedorRepository.save(contenedor));
    }
    else return Optional.empty();
}
```

Modificación de contenedor

Dentro de la aplicación el usuario podrá realizar una modificación de un contenedor. El acceso a la pantalla de modificación del contenedor se realizará desde la consulta del propio contenedor desde donde dispone de un botón modificar o pulsando el botón lápiz del registro en el listado que le dará acceso al formulario.

Dentro de la pantalla de modificación el usuario podrá digitar el nombre, descripción y contenedor al que pertenece, este último puede buscarlo mediante una lista o escaneando directamente el qr del contenedor al que va a pertenecer, también una vez seleccionado puede consultar la información de ese contenedor.

El campo id del contenedor es único y no se puede modificar.



Como resultado de la operación, nos llevará a la consulta del contenedor que hemos modificado.



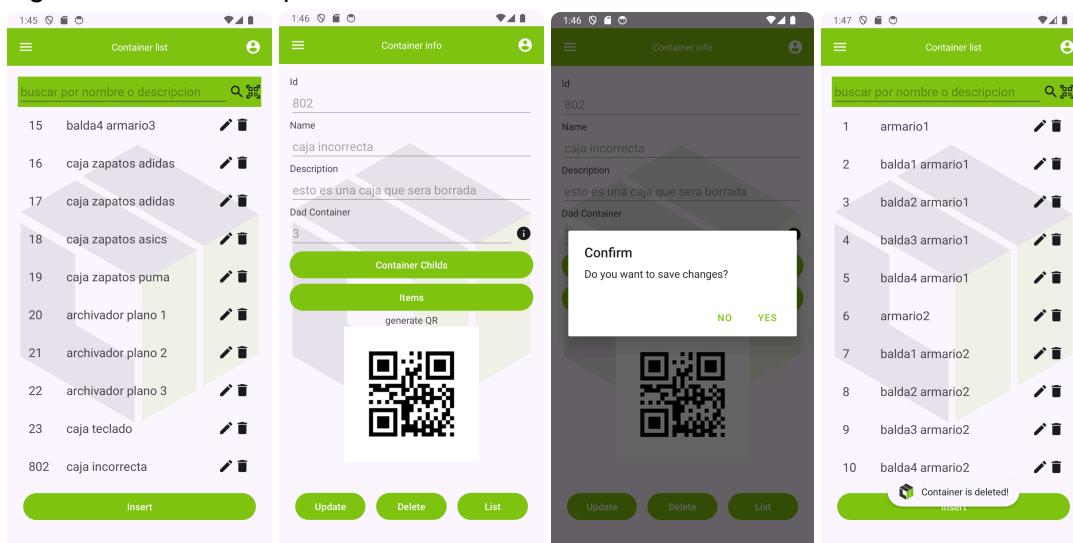
Esta gestión es realizada por el controller de contenedores.

```
@PutMapping("/contenedores/{id}")
public Optional<Contenedor> putContenedor(@RequestBody Contenedor contenedor,@PathVariable Long id) {
    Optional<Contenedor> contenedorSaved = contenedorRepository.findById(id);
    if (contenedorSaved.isPresent()) {
        contenedorSaved.get().setNombre(contenedor.getNombre());
        contenedorSaved.get().setDescripcion(contenedor.getDescripcion());
        contenedorSaved.get().setContenedorPadre(contenedor.getContenedorPadre());
        //contenedor.setId(id);
        contenedorSaved =Optional.of(contenedor);
        contenedorRepository.save(contenedor);
    }
    return contenedorSaved;
}
```

Borrado de contenedor

Dentro de la aplicación el usuario podrá realizar el borrado de un contenedor.

El acceso a la operativa de borrado del contenedor se realizará desde la consulta del propio contenedor desde donde dispone de un botón borrar o pulsando el botón papelera del registro en el listado que le dará acceso a la acción directamente.



Una vez ejecutado el borrado nos llevará a la pantalla de listado.

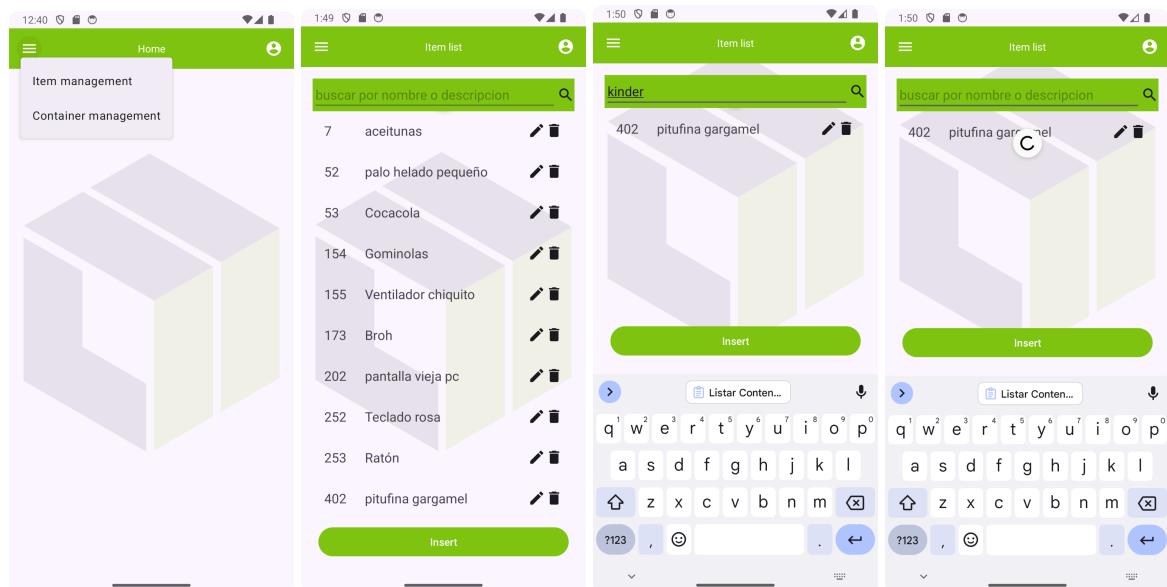
Esta gestión es realizada por el controller de contenedores.

```
@DeleteMapping("/contenedores/{id}")
public void deleteContenedor(@PathVariable Long id) {
    contenedorRepository.deleteById(id);
}
```

Listar Objetos y filtrado de los mismos

Dentro de la aplicación el usuario podrá realizar una búsqueda de los objetos existentes en la aplicación.

Dentro de la búsqueda el usuario podrá actualizar la búsqueda mediante los botones de búsqueda o deslizando hacia abajo y filtrar por cualquier texto contenido en la información de los objetos.



Esta gestión es realizada por el controller de items.

```
@GetMapping("/items")
public List<Item> getItems(@RequestParam(required = false) String query) {
    if (query==null) {
        return itemRepository.findAll();
    }
    else {
        return itemRepository.findByNombreContainsIgnoreCaseOrDescripcionContainsIgnoreCase(query, query);
    }
}
```

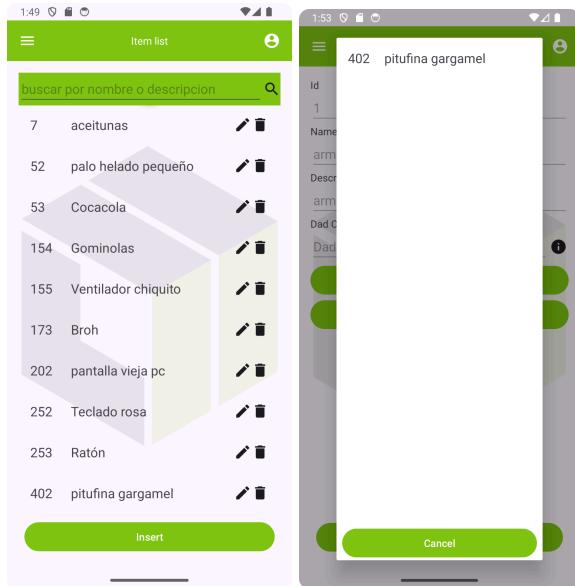
Información del objeto

Dentro de la aplicación el usuario podrá realizar una consulta de un objeto existente en la aplicación.

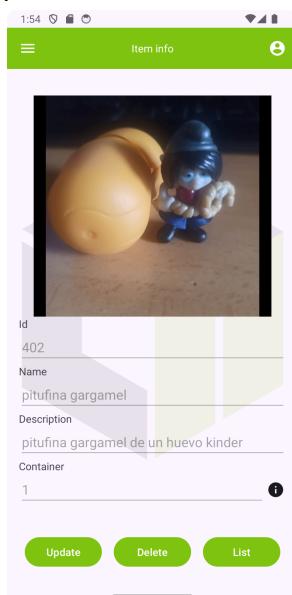
El acceso a la pantalla de consulta del objeto se realizará desde varios puntos.

El primero desde la búsqueda listada de objetos podrá seleccionar un registro llevando a esta pantalla.

El segundo desde la consulta de objetos asociados a un contenedor.



Dentro de la consulta el usuario podrá actualizar la información deslizando hacia abajo, consultar la imagen asociada si la tiene, el id, nombre, descripción y contenedor al que pertenece.



Esta gestión es realizada por el controller de items.

```
@GetMapping("/items/{id}")
public Optional<Item> getItem(@PathVariable Long id) {
    return itemRepository.findById(id);
}
```

```
@GetMapping("/items/{id}/imagen")
public ResponseEntity<byte[]> obtenerImagen(@PathVariable Long id) {
    Optional <Item> itemSaved = itemRepository.findById(id);
    if (itemSaved.isPresent() && itemSaved.get().getImagen()!=null) {

        Imagen imagen = itemSaved.get().getImagen();
        return ResponseEntity.ok()
            .contentType(MediaType.parseMediaType(imagen.getTipo()))
            .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" + imagen.getNombre() + "\"")
            .body(imagen.getDato());
    }
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
}
```

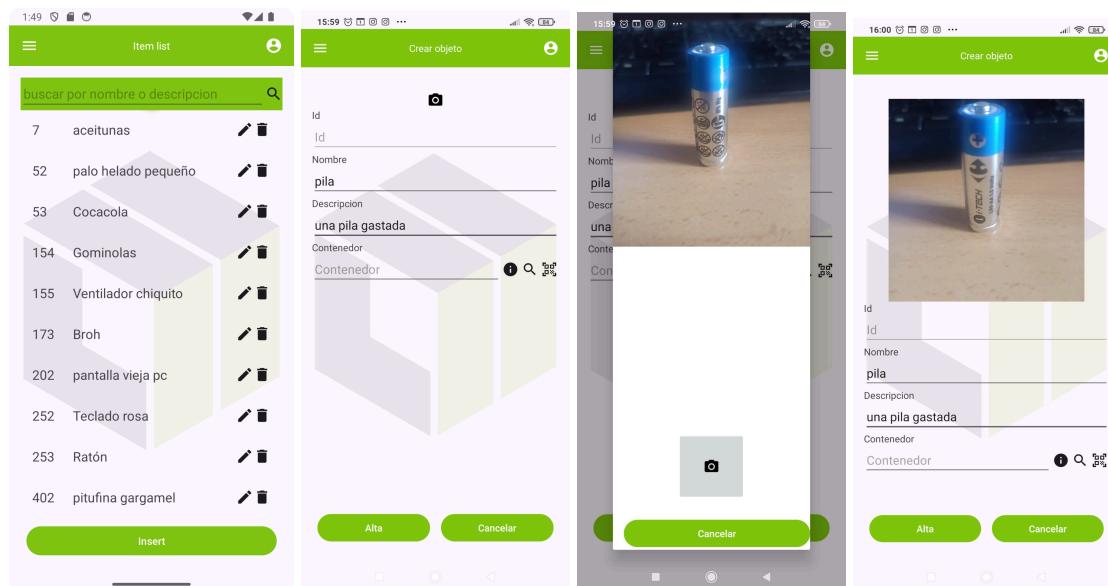
Alta de objeto

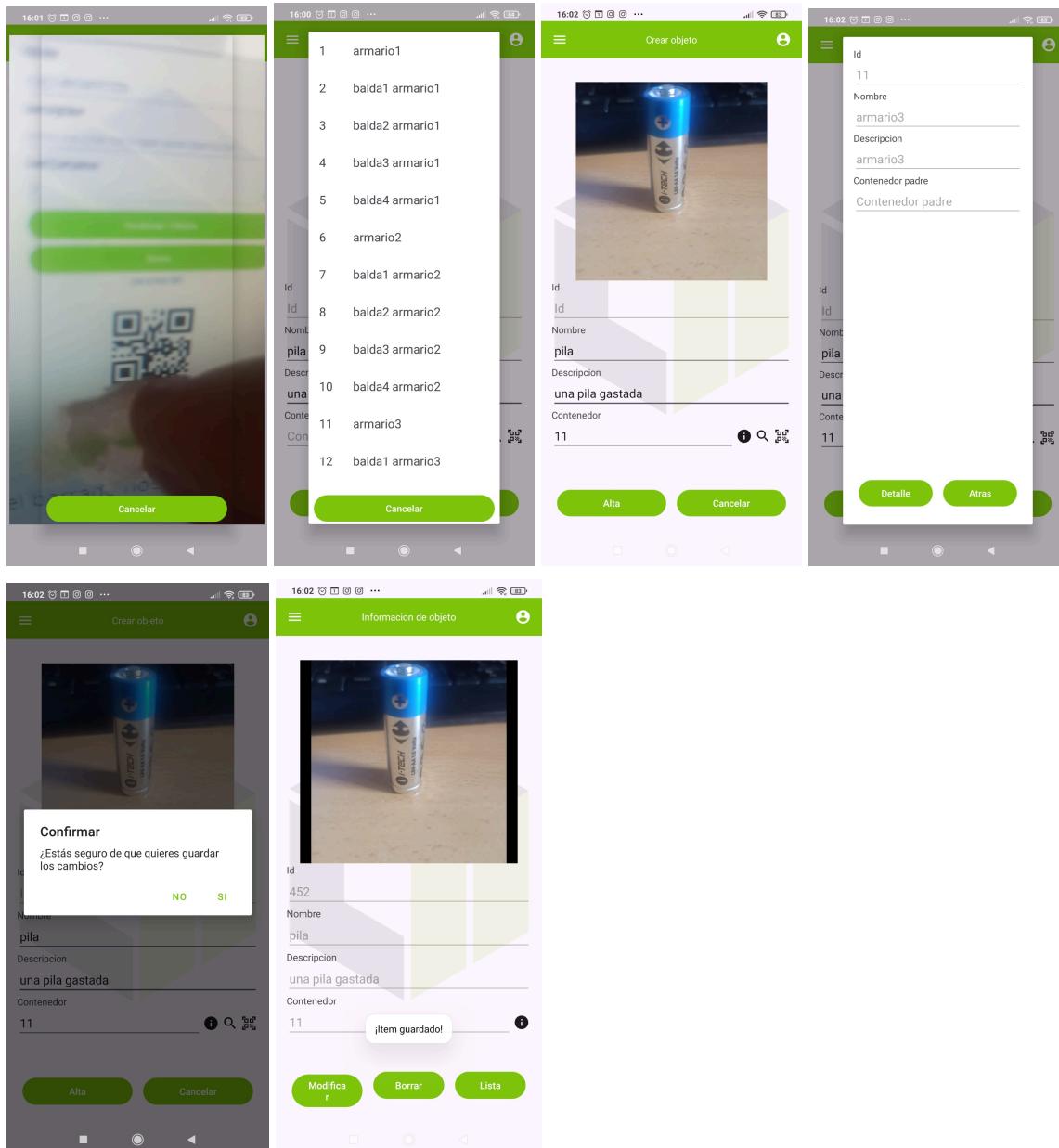
Dentro de la aplicación el usuario podrá realizar un alta de un objeto.

El acceso a la pantalla de alta del objeto se realizará desde la búsqueda lista de objetos desde donde dispone de un botón que le dará acceso al formulario de alta.

Dentro de la pantalla de alta el usuario podrá tomar una foto, digitar el nombre, descripción y contenedor al que pertenece, este último puede buscarlo mediante una lista o escaneando directamente el qr del contenedor al que va a pertenecer, también una vez seleccionado puede consultar la información de ese contenedor.

El campo id del objeto es único autogenerado por la aplicación.





Esta gestión es realizada por el controller de items.

```
@PostMapping("/items")
public Optional<Item> postItem(@RequestBody Item item) {
    Optional<Item> itemSaved = itemRepository.findById(item.getId());
    if (itemSaved.isEmpty()) {
        return Optional.of(itemRepository.save(item));
    }
    else return Optional.empty();
}
```

```
@PostMapping("/items/{id}/imagen")
public ResponseEntity<String> subirImagen(@PathVariable Long id,@RequestPart MultipartFile file) {
    try {
        Optional <Item> itemSaved = itemRepository.findById(id);
        if (itemSaved.isPresent()) {
            Imagen imagen = new Imagen();
            if (itemSaved.get().getImagen() != null) {
                imagen = itemSaved.get().getImagen();
            }
            imagen.setNombre(file.getOriginalFilename());
            imagen.setTipo(file.getContentType());
            imagen.setDatos(file.getBytes());
            itemSaved.get().setImagen(imagen);
            imagen=imagenRepository.save(imagen);
            Item itemSavedFinal = itemRepository.save(itemSaved.get());
            return ResponseEntity.status(HttpStatus.OK).body("Imagen subida con éxito: " + itemSavedFinal.getId());
        }
    } catch (IOException e) {
    }
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
}
```

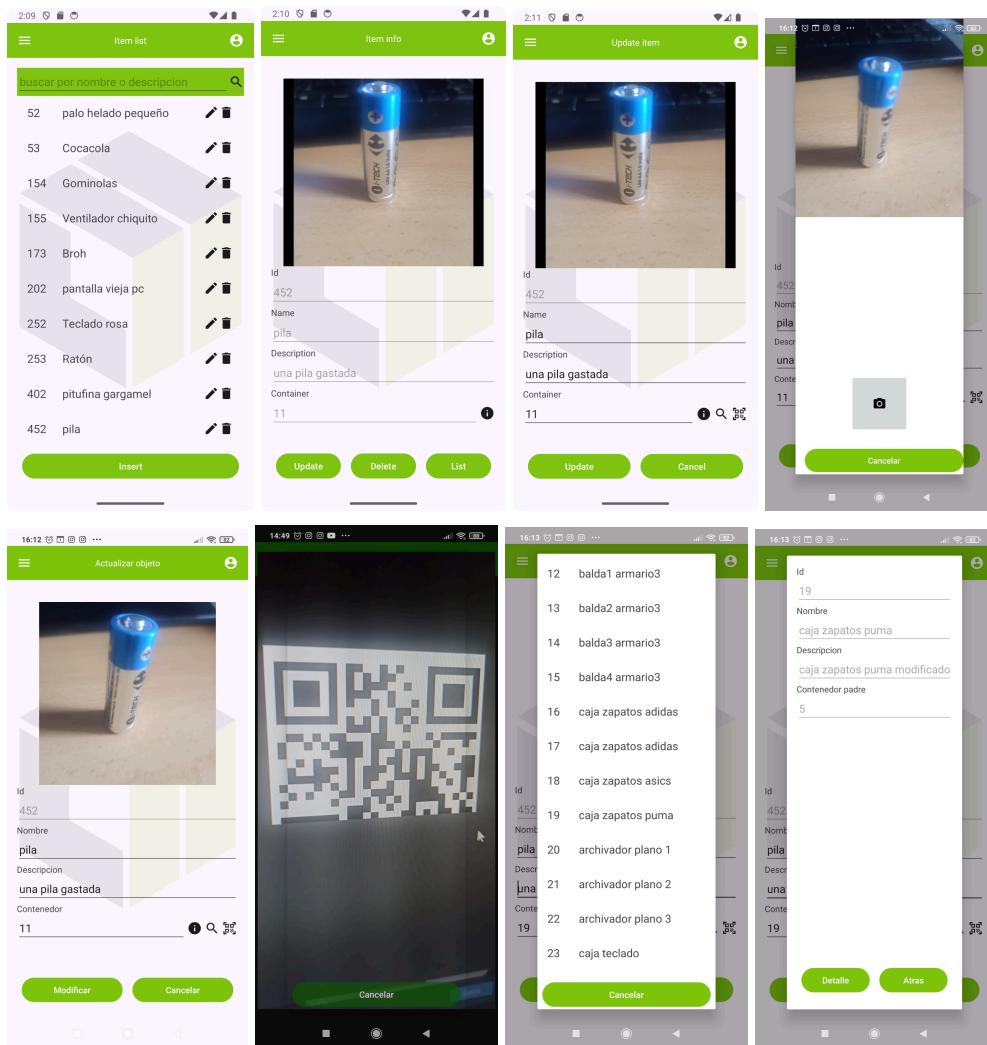
Modificación de objeto

Dentro de la aplicación el usuario podrá realizar una modificación de un objeto.

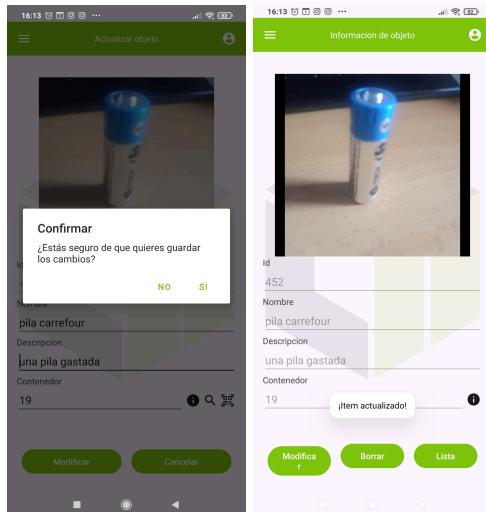
El acceso a la pantalla de modificación del objeto se realizará desde la consulta del propio objeto desde donde dispone de un botón modificar o pulsando el botón lápiz del registro en el listado que le dará acceso al formulario.

Dentro de la pantalla de modificación el usuario podrá tomar una nueva foto del objeto, digitar el nombre, descripción y contenedor al que pertenece, este último puede buscarse mediante una lista o escaneando directamente el qr del contenedor al que va a pertenecer, también una vez seleccionado puede consultar la información de ese contenedor.

El campo id del objeto es único y no se puede modificar.



Como resultado de la operación, nos llevará a la consulta del objeto que hemos modificado.



Esta gestión es realizada por el controller de items.

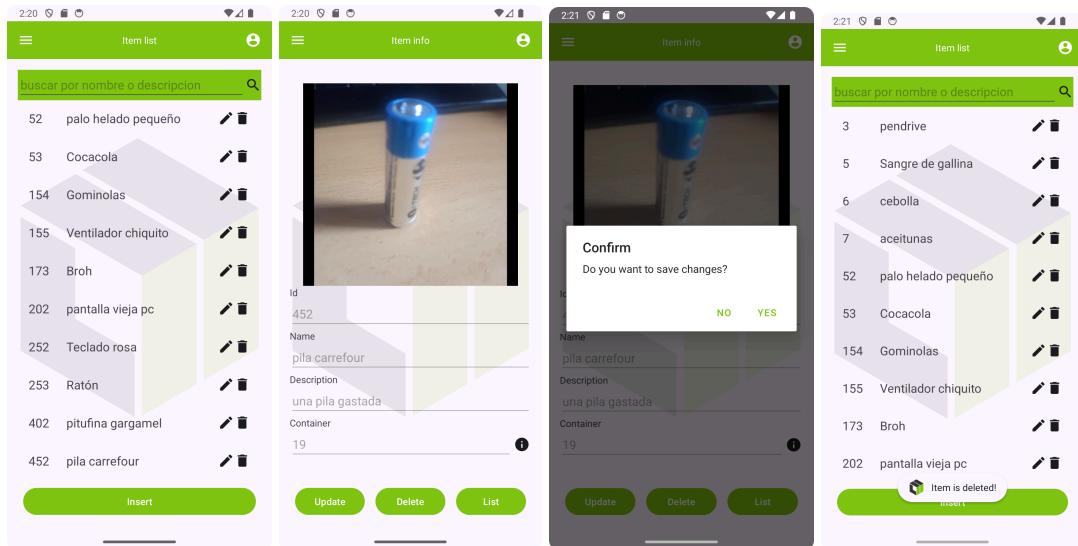
```
@PutMapping("/items/{id}")
public Optional<Item> putItem(@RequestBody Item item,@PathVariable Long id) {
    Optional<Item> itemSaved = itemRepository.findById(id);
    if (itemSaved.isPresent()) {
        itemSaved.get().setNombre(item.getNombre());
        itemSaved.get().setDescripcion(item.getDescripcion());
        itemSaved.get().setContenedor(item.getContenedor());
        //item.setId(id);
        itemSaved =Optional.of(itemRepository.save(itemSaved.get()));
    }
    return itemSaved;
}

@PostMapping("/items/{id}/imagen")
public ResponseEntity<String> subirImagen(@PathVariable Long id,@RequestPart MultipartFile file) {
    try {
        Optional <Item> itemSaved = itemRepository.findById(id);
        if (itemSaved.isPresent()) {
            Imagen imagen = new Imagen();
            if (itemSaved.get().getImagen()!=null) {
                imagen = itemSaved.get().getImagen();
            }
            imagen.setNombre(file.getOriginalFilename());
            imagen.setTipo(file.getContentType());
            imagen.setDatos(file.getBytes());
            itemSaved.get().setImagen(imagen);
            itemSaved=ImagenRepository.save(imagen);
            Item itemSavedFinal = itemRepository.save(itemSaved.get());
            return ResponseEntity.status(HttpStatus.OK).body("Imagen subida con éxito: " + itemSavedFinal.getId());
        }
    } catch (IOException e) {
    }
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
}
```

Borrado de objeto

Dentro de la aplicación el usuario podrá realizar el borrado de un objeto.

El acceso a la operativa de borrado del objeto se realizará desde la consulta del propio objeto desde donde dispone de un botón borrar o pulsando el botón papelera del registro en el listado que le dará acceso a la acción directamente.



Una vez ejecutado el borrado nos llevará a la pantalla de listado.

Esta gestión es realizada por el controller de items.

```
@DeleteMapping("/items/{id}")
public void deleteItem(@PathVariable Long id) {
    Optional<Item> itemSaved = itemRepository.findById(id);
    itemRepository.deleteById(id);
    if (itemSaved.isPresent() && itemSaved.get().getImagen()!=null) {
        imagenRepository.delete(itemSaved.get().getImagen());
    }
}
```

Listar Usuarios

Dentro de la aplicación el usuario con permisos de administrador podrá realizar una búsqueda de los usuarios existentes en la aplicación.

email	nombre	password	insert_date	active	admin	last_login
admin@localhost.com	admin	8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a8...	2024-06-07 00:46:46 000000 1	1	1	2024-06-07

Dentro de la búsqueda, el usuario podrá actualizar la búsqueda deslizando hacia abajo.

Esta gestión es realizada por el controller de usuarios, donde se comprueba que el usuario logado tenga permisos de administrador.

```

@GetMapping("/usuarios")
public ResponseEntity<List<Usuario>> getUsuarios(@RequestHeader("Authorization") String authorizationHeader) {
    if(validarUsuario(authorizationHeader)) {
        return ResponseEntity.ok().body(usuarioRepository.findAll());
    }
    else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}

private boolean validarUsuario(String authorizationHeader) {
    Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
    Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
    if (usuarioSaved.isPresent() && usuarioSaved.get().getPassword().equals(usuarioAuthorizationHeader.getPassword()) && usuarioSaved.get().isAdmin()) {
        return true;
    }
    return false;
}

private Usuario decodeAuthorizationHeader(String authorizationHeader) {
    String base64Credentials = authorizationHeader.substring("Basic ".length());
    byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);
    String decodedString = new String(decodedBytes, StandardCharsets.UTF_8);

    // Split username and password
    String[] credentials = decodedString.split(":", 2);
    String username = credentials[0];
    String password = credentials[1];
    Usuario usuario = new Usuario();
    usuario.setEmail(username);
    usuario.setPassword(password);
    return usuario;
}

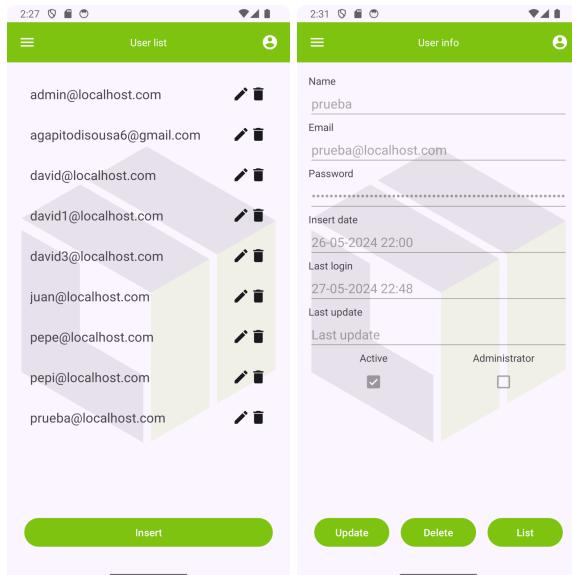
private boolean permitSetAdmin(String authorizationHeader, Usuario usuario) {
    if (usuario.isAdmin()) {
        return true;
    }
    else {
        Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
        Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
        if (usuarioSaved.isPresent() && usuarioSaved.get().isAdmin()) {
            return true;
        }
    }
    return false;
}

```

Información del usuario

Dentro de la aplicación el usuario podrá realizar una consulta de un usuario existente en la aplicación.

El acceso a la pantalla de consulta del usuario se realizará desde la búsqueda listada de usuarios seleccionando un registro y llevándolo a la pantalla.



Dentro de la consulta el usuario podrá actualizar la información deslizando hacia abajo, consultar el email, nombre, la fecha de alta, la modificación, la última conexión y los permisos que tiene dentro de la aplicación.

Esta gestión es realizada por el controller de usuarios, donde se comprueba que el usuario logado tenga permisos de administrador o se esté consultado sobre el usuario logado.

```
@GetMapping("/usuarios/{email}")
public ResponseEntity<Optional<Usuario>> getUsuario(@RequestHeader("Authorization") String authorizationHeader, @PathVariable String email) {
    if(validarUsuario(authorizationHeader) || decodeAuthorizationHeader(authorizationHeader).getEmail().equals(email)) {
        return ResponseEntity.ok().body(usuarioRepository.findById(email));
    } else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}

private boolean validarUsuario(String authorizationHeader) {
    Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
    Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
    if (usuarioSaved.isPresent() && usuarioSaved.get().getPassword().equals(usuarioAuthorizationHeader.getPassword()) && usuarioSaved.get().isAdmin()) {
        return true;
    }
    return false;
}

private Usuario decodeAuthorizationHeader(String authorizationHeader) {
    String base64Credentials = authorizationHeader.substring("Basic ".length());
    byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);
    String decodedString = new String(decodedBytes, StandardCharsets.UTF_8);

    // Split username and password
    String[] credentials = decodedString.split(":", 2);
    String username = credentials[0];
    String password = credentials[1];
    Usuario usuario = new Usuario();
    usuario.setEmail(username);
    usuario.setPassword(password);
    return usuario;
}

private boolean permitSetAdmin(String authorizationHeader, Usuario usuario) {
    if (usuario.isAdmin()) {
        return true;
    } else {
        Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
        Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
        if (usuarioSaved.isPresent() && usuarioSaved.get().isAdmin()) {
            return true;
        }
    }
    return false;
}
```

Alta de usuario

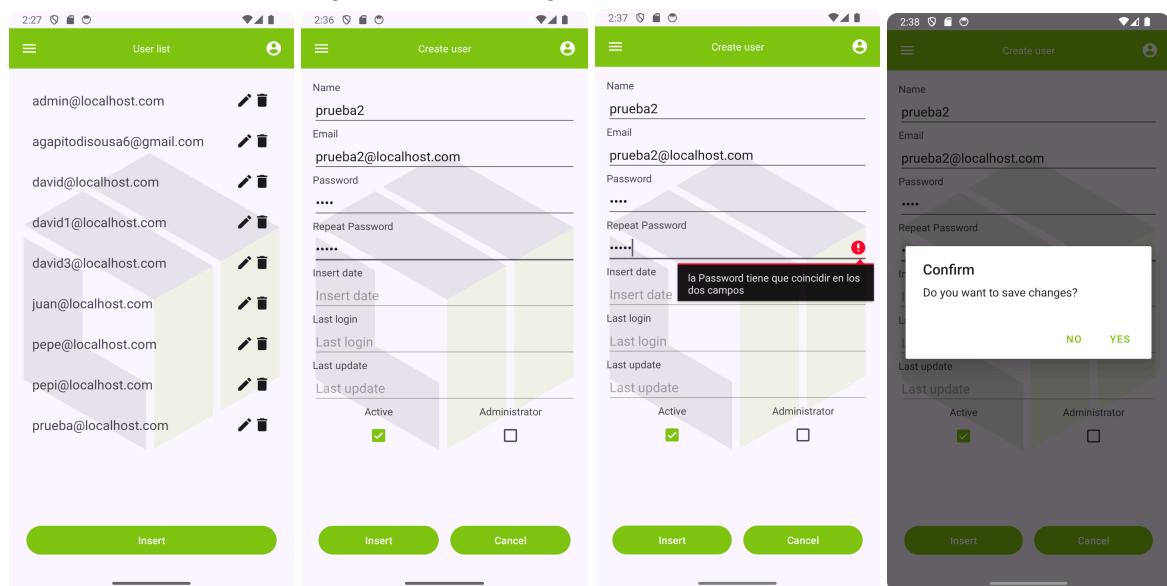
Dentro de la aplicación el usuario podrá realizar un alta de otro usuario. El acceso a la pantalla de alta de usuario se realizará desde la búsqueda lista de usuario desde donde dispone de un botón que le dará acceso al formulario de alta.

Dentro de la pantalla de alta el usuario podrá digitar el email, nombre, contraseñas y los permisos dentro de la aplicación.

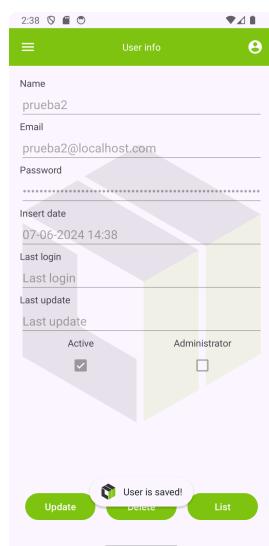
El campo password y repeat password, se validará para que coincidan.

El email es único dentro de la aplicación.

El resto de campos, son gestionados íntegramente por la aplicación.



Una vez insertado nos retornará a la pantalla de consulta del usuario.



Esta gestión es realizada por el controller de usuarios, donde se comprueba que el usuario logado tenga permisos de administrador.

```
@PostMapping("/usuarios")
public ResponseEntity<Optional<Usuario>> postUsuario(@RequestHeader("Authorization") String authorizationHeader, @RequestBody Usuario usuario) {
    if(validarUsuario(authorizationHeader)) {
        Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuario.getEmail());
        if (usuarioSaved.isEmpty()) {
            //usuario.setActive(false);
            usuario.setInsertDate(new Date());
            return ResponseEntity.ok().body(Optional.of(usuarioRepository.save(usuario)));
        }
        else return ResponseEntity.ok().body(Optional.empty());
    }
    else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}

private boolean validarUsuario(String authorizationHeader) {
    Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
    Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
    if (usuarioSaved.isPresent() && usuarioSaved.get().getPassword().equals(usuarioAuthorizationHeader.getPassword()) && usuarioSaved.get().isAdmin()) {
        return true;
    }

    return false;
}

private Usuario decodeAuthorizationHeader(String authorizationHeader) {
    String base64Credentials = authorizationHeader.substring("Basic ".length());
    byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);
    String decodedString = new String(decodedBytes, StandardCharsets.UTF_8);

    // Split username and password
    String[] credentials = decodedString.split(":", 2);
    String username = credentials[0];
    String password = credentials[1];
    Usuario usuario = new Usuario();
    usuario.setEmail(username);
    usuario.setPassword(password);
    return usuario;
}

private boolean permiteSetAdmin(String authorizationHeader, Usuario usuario) {
    if (usuario.isAdmin()) {
        return true;
    }
    else {
        Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
        Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
        if (usuarioSaved.isPresent() && usuarioSaved.get().isAdmin()) {
            return true;
        }
    }

    return false;
}
```

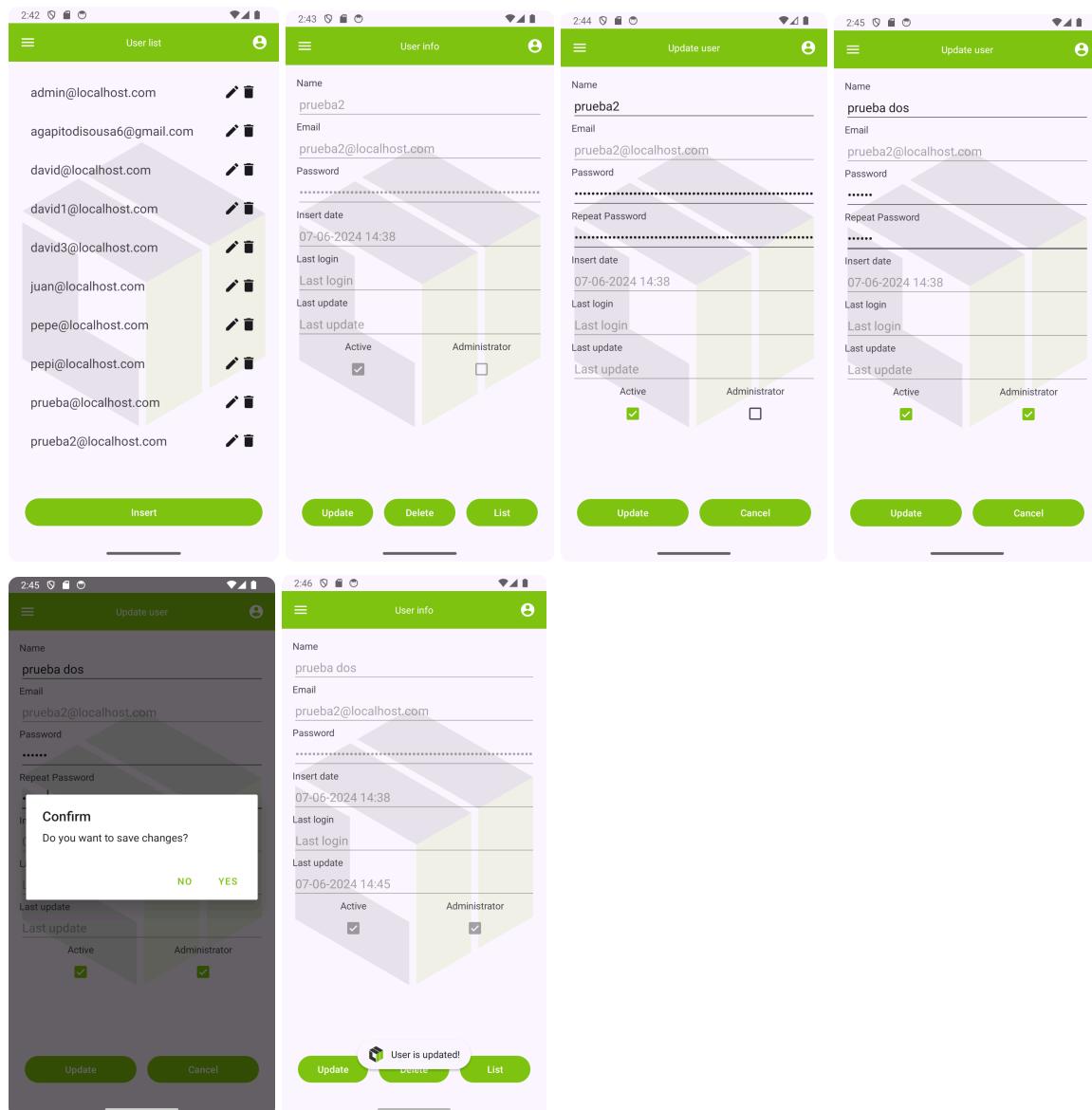
Modificación de usuario

Dentro de la aplicación el usuario podrá realizar una modificación de otro usuario.

El acceso a la pantalla de modificación del usuario se realizará desde la consulta del propio usuario a modificar desde donde dispone de un botón modificar o pulsando el botón lápiz del registro en el listado que le dará acceso al formulario.

Dentro de la pantalla de modificación el usuario podrá digitar el nombre, cambiar la contraseña o modificar los permisos de la aplicación.

El campo email del usuario es único y no se puede modificar como tampoco los campos de fecha de alta, fecha de modificación y fecha de último acceso.



Como resultado de la operación, nos llevará a la consulta del usuario que hemos modificado.

Cualquier usuario logado con el usuario al que se le modifique la contraseña o los permisos se le aplicará en el momento quedando inaccesible en función de las opciones modificadas.

Esta gestión es realizada por el controller de usuarios, donde se comprueba que el usuario logado tenga permisos de administrador o se esté modificando sobre el usuario logado.

```
@PutMapping("/usuarios/{email}")
public ResponseEntity<Optional<Usuario>> putUsuario(@RequestHeader("Authorization") String authorizationHeader, @RequestBody Usuario usuario,@PathVariable String email) {
    if(validarUsuario(authorizationHeader) || decodeAuthorizationHeader(authorizationHeader).getEmail().equals(email)) {
        Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuario.getEmail());
        if (!usuarioSaved.isEmpty()) {
            usuario.setEmail(email);
            usuario.setUpdateDate(new Date());
            if(!permiteSetAdmin(authorizationHeader, usuarioSaved.get())) {
                usuario.setAdmin(false);
            }
            usuarioSaved = Optional.of(usuarioRepository.save(usuario));
        }
        return ResponseEntity.ok().body(usuarioSaved);
    }
    else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}

private boolean validarUsuario(String authorizationHeader) {
    Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
    Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
    if (usuarioSaved.isPresent() && usuarioSaved.get().getPassword().equals(usuarioAuthorizationHeader.getPassword()) && usuarioSaved.get().isAdmin()) {
        return true;
    }
    return false;
}

private Usuario decodeAuthorizationHeader(String authorizationHeader) {
    String base64Credentials = authorizationHeader.substring("Basic ".length());
    byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);
    String decodedString = new String(decodedBytes, StandardCharsets.UTF_8);

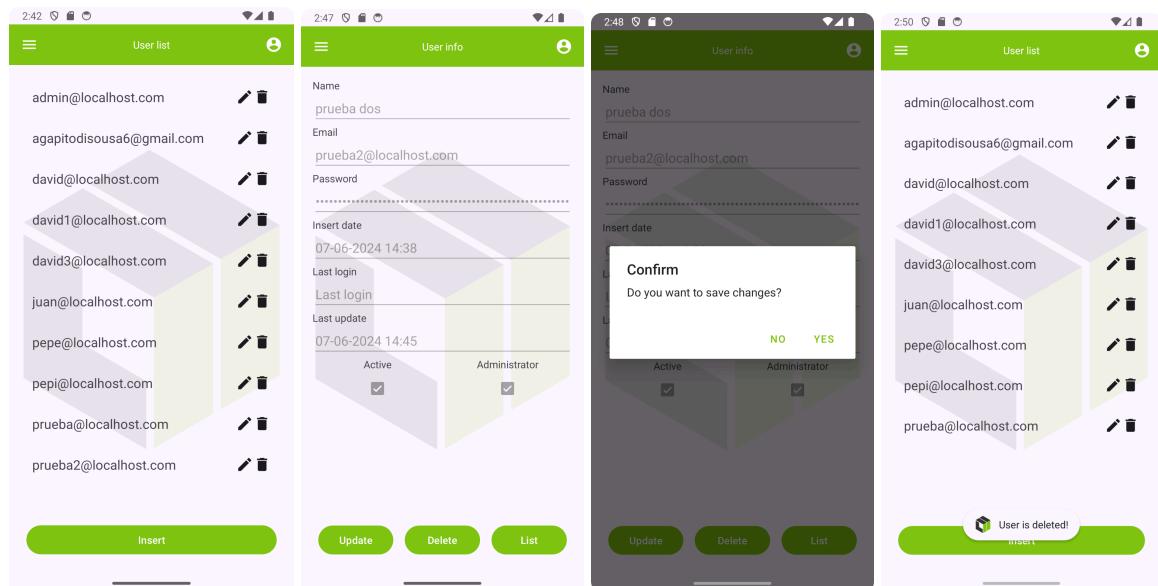
    // Split username and password
    String[] credentials = decodedString.split(":", 2);
    String username = credentials[0];
    String password = credentials[1];
    Usuario usuario = new Usuario();
    usuario.setEmail(username);
    usuario.setPassword(password);
    return usuario;
}

private boolean permiteSetAdmin(String authorizationHeader, Usuario usuario) {
    if (usuario.isAdmin()) {
        return true;
    }
    else {
        Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
        Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
        if (usuarioSaved.isPresent() && usuarioSaved.get().isAdmin()) {
            return true;
        }
    }
    return false;
}
```

Borrado de usuario

Dentro de la aplicación el usuario podrá realizar el borrado de un usuario.

El acceso a la operativa de borrado del usuario se realizará desde la consulta del propio usuario, desde donde dispone de un botón borrar o pulsando el botón papelera del registro en el listado que le dará acceso a la acción directamente.



Una vez ejecutado el borrado nos llevará a la pantalla de listado.

Cualquier usuario logado con el usuario eliminado dejará de tener acceso a la aplicación.

Esta gestión es realizada por el controller de usuarios, donde se comprueba que el usuario logado tenga permisos de administrador o se esté borrando sobre el usuario logado.

```
@DeleteMapping("/usuarios/{email}")
public ResponseEntity<Void> deleteUser(@RequestHeader("Authorization") String authorizationHeader, @PathVariable String email) {
    if(validarUsuario(authorizationHeader) || decodeAuthorizationHeader(authorizationHeader).getEmail().equals(email)) {
        usuarioRepository.deleteById(email);
        return ResponseEntity.ok().body(null);
    }
    else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}

private boolean validarUsuario(String authorizationHeader) {
    Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
    Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
    if (usuarioSaved.isPresent() && usuarioSaved.get().getPassword().equals(usuarioAuthorizationHeader.getPassword()) && usuarioSaved.get().isAdmin()) {
        return true;
    }
    return false;
}

private Usuario decodeAuthorizationHeader(String authorizationHeader) {
    String base64Credentials = authorizationHeader.substring("Basic ".length());
    byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);
    String decodedString = new String(decodedBytes, StandardCharsets.UTF_8);

    // Split username and password
    String[] credentials = decodedString.split(":", 2);
    String username = credentials[0];
    String password = credentials[1];
    Usuario usuario = new Usuario();
    usuario.setEmail(username);
    usuario.setPassword(password);
    return usuario;
}

private boolean permitSetAdmin(String authorizationHeader, Usuario usuario) {
    if (usuario.isAdmin()) {
        return true;
    }
    else {
        Usuario usuarioAuthorizationHeader = decodeAuthorizationHeader(authorizationHeader);
        Optional<Usuario> usuarioSaved = usuarioRepository.findById(usuarioAuthorizationHeader.getEmail());
        if (usuarioSaved.isPresent() && usuarioSaved.get().isAdmin()) {
            return true;
        }
    }
    return false;
}
```

APLICACIÓN WEB

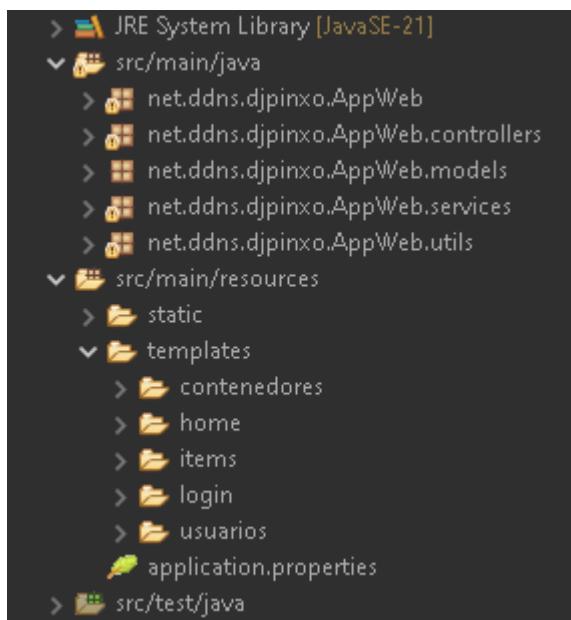
Una de las capas del usuario es la página web. Si bien no tiene tantas funcionalidades como el aplicativo web permite observar datos más rápidamente y ejecutar prácticamente todas las operativas a excepción de las que requieren recursos hardware.

El aplicativo está concebido para ser usado de forma rápida y cómoda para los usuarios, esto es gracias al gran formato de las pantallas donde se ejecutan este tipo de aplicaciones. La aplicación web también se basa en el modelo MVC.

Las 3 grandes áreas de gestión son las mismas que en el aplicativo móvil, los usuarios son gestionados únicamente por otros usuarios administradores.

Toda comunicación realizada a la persistencia de datos es realizada mediante la librería de spring y su conversor json.

En este caso la generación de QR utilizamos un web de terceros que nos facilita dicha acción.



CASOS DE USO

Registro de usuario:

El usuario se puede registrar en la aplicación introduciendo el nombre, el correo electrónico y la contraseña.

Internamente se usa el algoritmo de hash SHA-256 para encriptar la contraseña y se comprueba que el email no está registrado en la aplicación.

Si hay algún problema con los campos la aplicación validará y notificará de los mismos al usuario.

localhost:8081/login



Introduzca las credenciales

Email
Password

Acceder

Registrar

localhost:8081/register

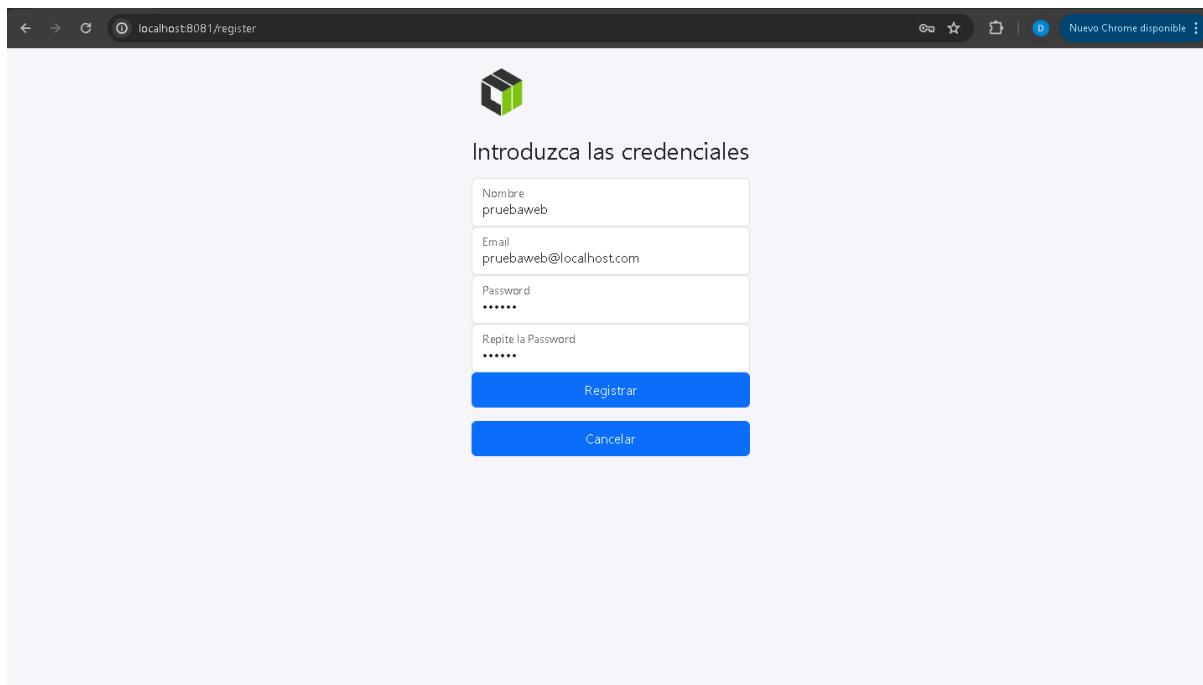


Introduzca las credenciales

Nombre
Email
Password
Repite la Password

Registrar

Cancelar



localhost:8081/register

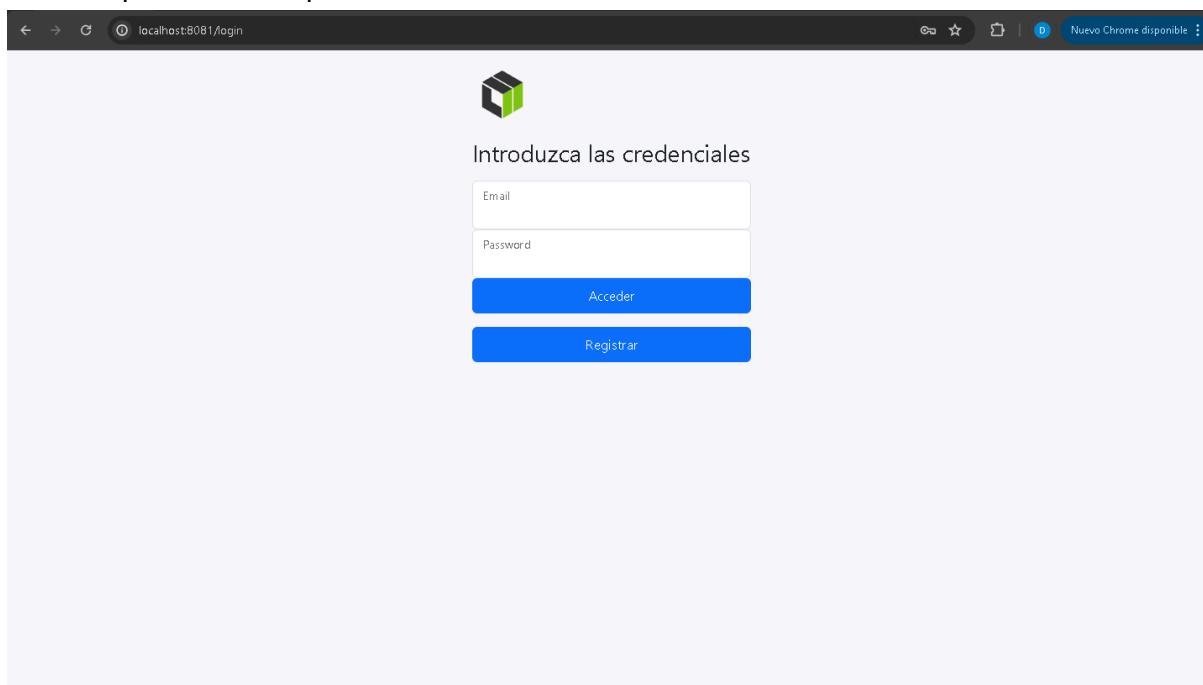
Introduzca las credenciales

Nombre pruebaweb
Email pruebaweb@localhost.com
Password *****
Repite la Password *****

Registrar

Cancelar

Una vez realizada la acción, nos llevará de nuevo a la ventana de login.
Por último el usuario quedará por defecto inaccesible hasta que un usuario administrador le conceda permiso a la aplicación.



localhost:8081/login

Introduzca las credenciales

Email
Password

Acceder

Registrar

```

@PostMapping("/register")
public String postRegister(HttpSession session, Model model, @RequestParam("email") String email, @RequestParam("nombre") String nombre, @Re
    try {
        Usuario usuario = insertUser(email, nombre, password, repeatPassword);

        if (usuario != null) {
            return "redirect:/login";
        }
    } catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error en el registro del usuario");
    } catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "login/register";
}

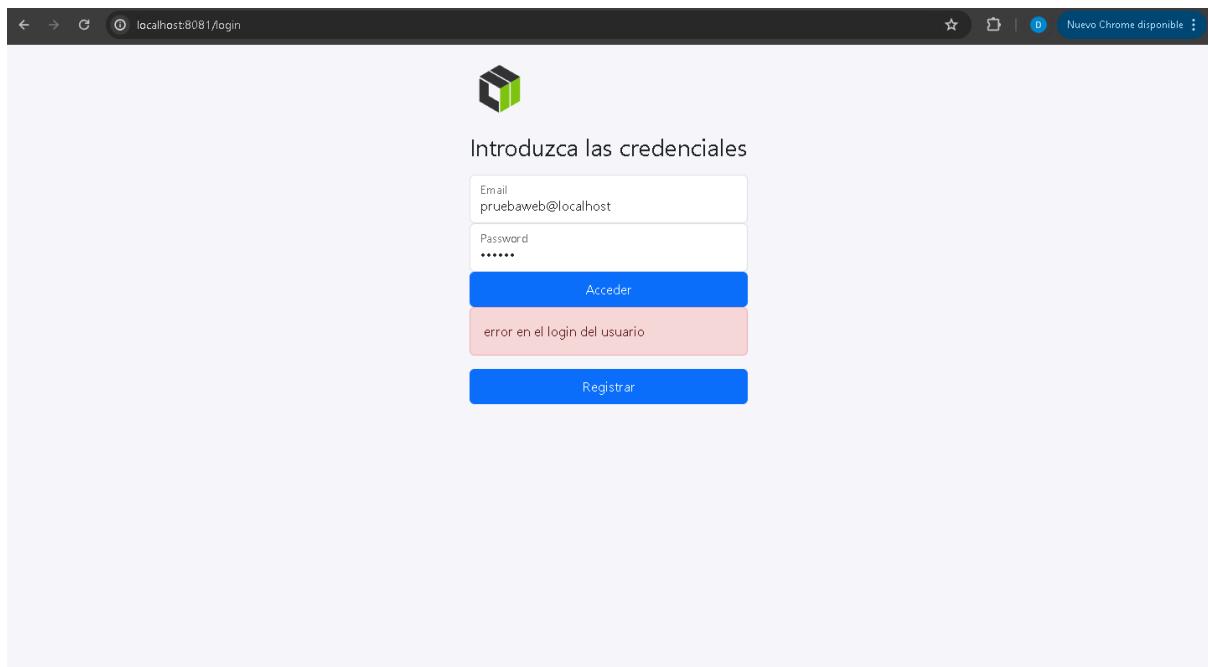
private Usuario insertUser(String email, String nombre, String password, String repeatPassword){
    if(validateUserForm(email, nombre, password, repeatPassword)){
        password = HashUtils.hashString(password);
        Usuario usuario = new Usuario(email, nombre, password, false, false);
        return apiservice.registerUsuario(usuario);
    }
    else {
        return null;
    }
}

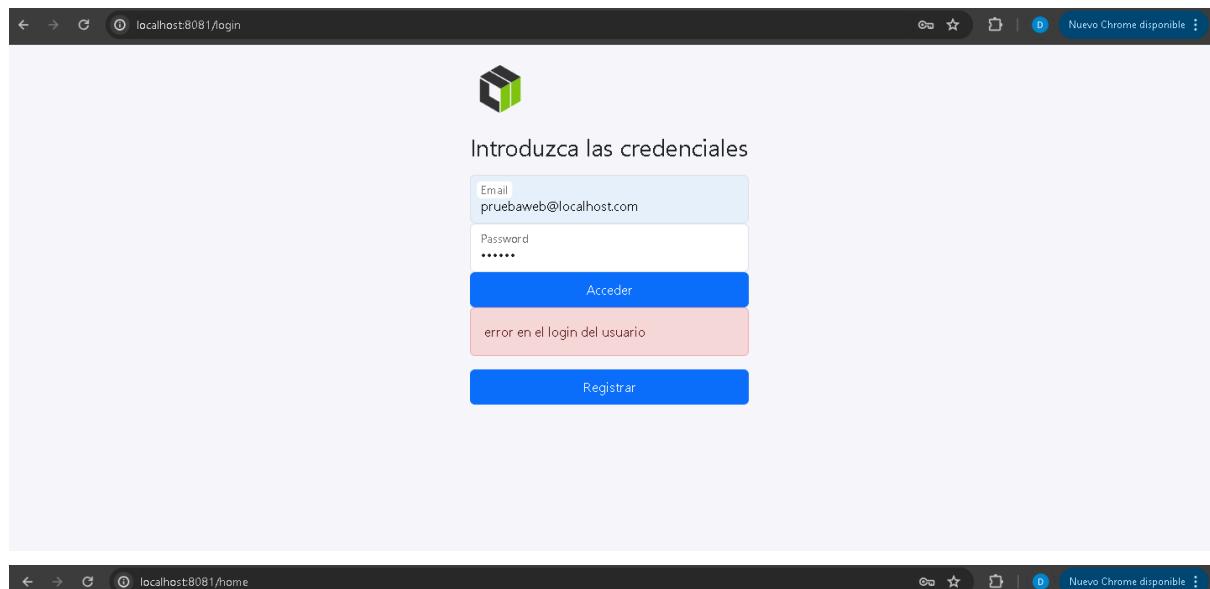
private boolean validateUserForm(String email, String nombre, String password, String repeatPassword){
    boolean result = true;
    if(email.isEmpty()) {
        result = false;
    }
    if(nombre.isEmpty()) {
        result = false;
    }
    if(password.isEmpty()) {
        result = false;
    }
    if(repeatPassword.isEmpty()) {
        result = false;
    }
    return result;
}

```

Login de usuario:

Para poder acceder a la aplicación se solicitará el correo electrónico y su contraseña, esta será hasheada en SHA-256 y enviada al servidor.





Para poder acceder a la aplicación será necesario que el usuario esté en estado activo en la base de datos.

pruebaweb@localhost.com pruebaweb 655e786674d9d3e77bc05ed1de37b4b6bc89f788829f9f3c67... 2024-06-07 17:18:24.000000 1 0 NULL

Una vez el paso de login sea correcto, los datos del usuario se almacenarán en una variable en sesión para su posterior utilización en la navegación.

```
private ApiService apiService;

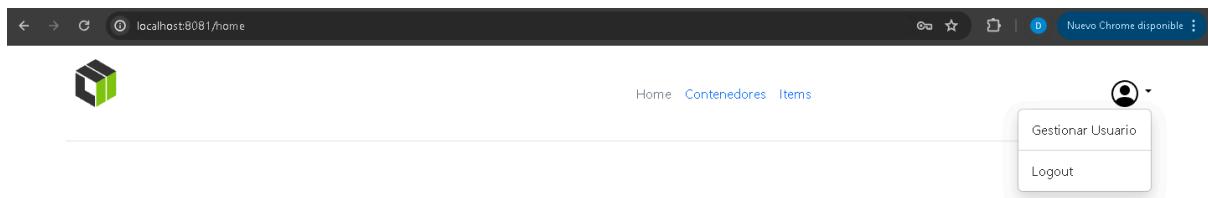
@GetMapping("/login")
public String getLogin(HttpSession session, Model model) {
    session.invalidate();
    return "login/login";
}

@PostMapping("/login")
public String postLogin(HttpSession session, Model model, @RequestParam("email") String email, @RequestParam("password") String password) {
    try {
        Usuario usuario = new Usuario();

        usuario.setEmail(email);
        usuario.setPassword(HashUtils.hashString(password));
        usuario = apiService.loginUsuario(usuario);
        session.setAttribute("usuarioLogin", usuario);
        apiService.setInterceptor(usuario);
        return "redirect:/home";
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error en el login del usuario");
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    apiService.removeInterceptor();
    return "login/login";
}
```

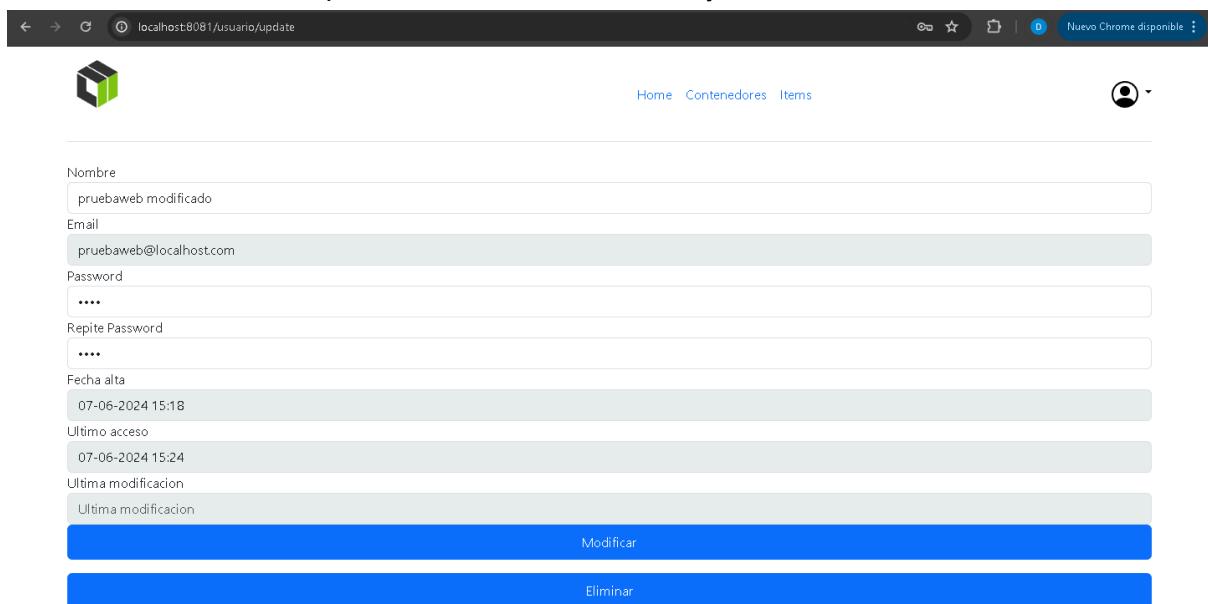
Modificación de usuario logado

Una vez logueado, desde cualquier punto de la aplicación podemos acceder a la gestión de nuestro usuario.



The screenshot shows a browser window with the URL 'localhost:8081/home'. The top navigation bar includes a logo, links for 'Home', 'Contenedores', and 'Items', and a user icon with a dropdown menu containing 'Gestionar Usuario' and 'Logout'.

En la misma tendremos dos operativas, en este caso la actualización del usuario, los datos del usuario modificables por uno mismo es el nombre y la contraseña.

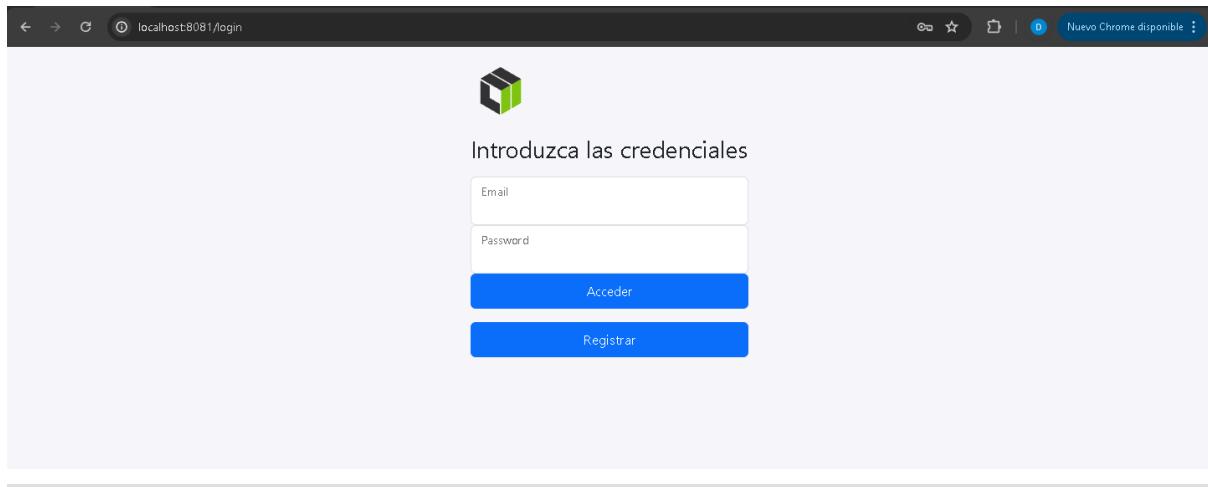


The screenshot shows a form titled 'usuario/update' with the following fields:

- Nombre: pruebaweb modificado
- Email: pruebaweb@localhost.com
- Password: ****
- Repite Password: ****
- Fecha alta: 07-06-2024 15:18
- Ultimo acceso: 07-06-2024 15:24
- Ultima modificacion: Ultima modificacion

At the bottom are two blue buttons: 'Modificar' and 'Eliminar'.

Una vez completado el cambio nos retorna al login de nuevo y actualiza en el usuario la fecha del cambio.



The screenshot shows a login page with the following fields:

- Email: pruebaweb@localhost.com
- Password: pruebaweb modificado

Below the form is a table with the following data:

pruebaweb@localhost.com	pruebaweb modificado	d74ff0ee8da3b9806b18c877dbf29bbde50b5bd8e4dad7a3a7...	2024-06-07 17:18:00.000000 1	0	2024-06-0
-------------------------	----------------------	---	------------------------------	---	-----------

Esta gestión es realizada en el mismo controller que la operativa general de usuarios.

```
@GetMapping("/usuario/update")
public String getUpdateUsuarioLoged(HttpServletRequest session, Model model) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        model.addAttribute("usuario", apiService.getUsuario(((Usuario)session.getAttribute("usuarioLogin")).getEmail()));
        return "usuarios/updateLoged";
    }
    catch (HttpClientErrorException.Unauthorized e) []
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/logout";
}

@PostMapping("/usuario/update")
public String postUpdateUsuarioLoged(HttpServletRequest session, Model model, @RequestParam(value="nombre") String nombre, @RequestParam(value="pa
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        Usuario usuario = updateUsuario(nombre, ((Usuario)session.getAttribute("usuarioLogin")).getEmail(), password, repeatPassword, ((Usuario)ses
        if (usuario != null) {
            return "redirect:/logout";
        }
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    model.addAttribute("usuario", session.getAttribute("usuarioLogin"));
    return "usuarios/updateLoged";
}
}

private Usuario updateUsuario(String nombre, String email, String password, String repeatPassword, boolean active, boolean admin) throws Ex
if(validateUpdateUsuarioForm(nombre, email, password, repeatPassword)){
    Usuario usuario = apiService.getUsuario(email);

    //userModel=new User(email, name, password, isAdmin, isActive);
    usuario.setNombre(nombre);
    //se verifica si la contraseña ha cambiado, si no lo hizo no volver a lanzar el hash
    if(!password.equals(usuario.getPassword())){
        password = Hashutils.hashString(password);
        usuario.setPassword(password);
        usuario.setActive(active);
        usuario.setAdmin(admin);

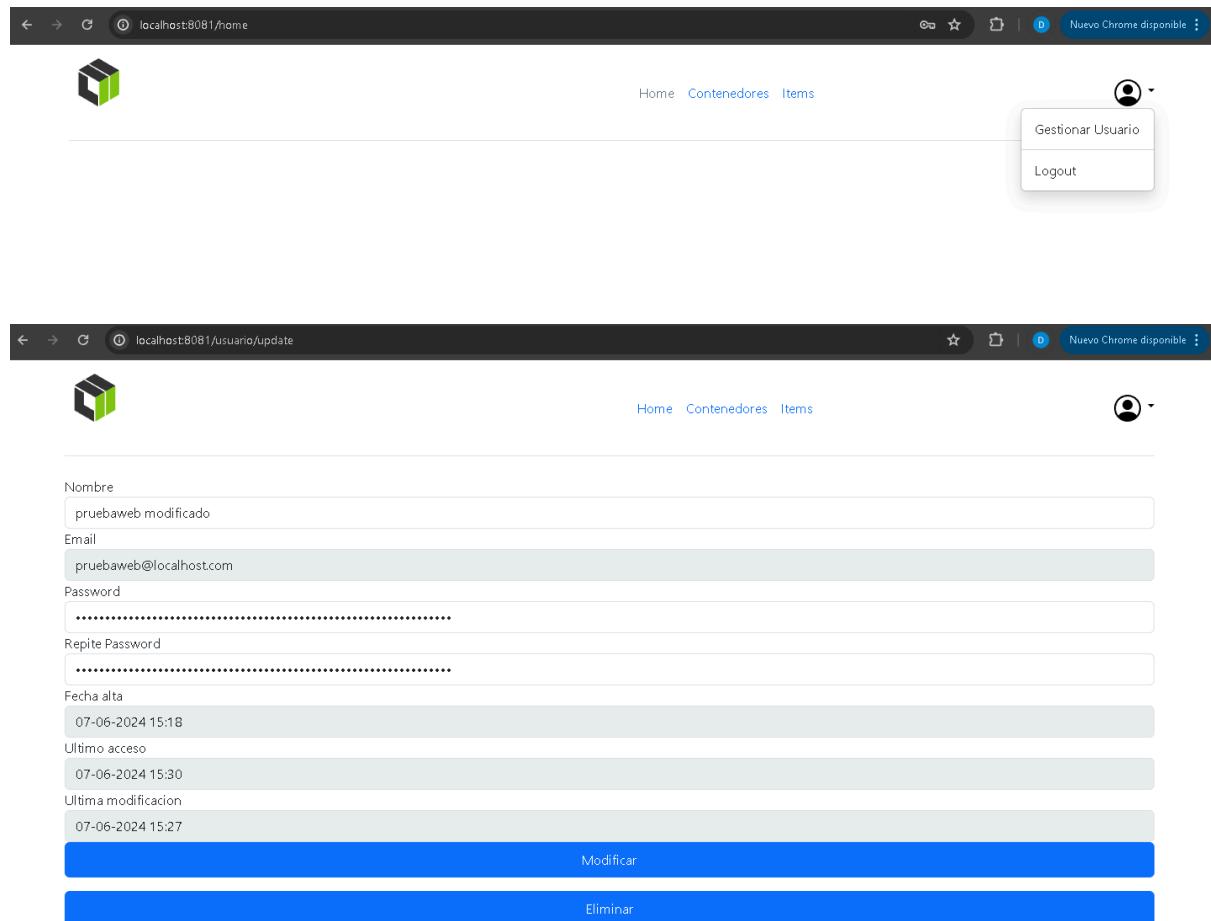
        return apiService.updateUsuario(email, usuario);
    }
    else {
        return null;
    }
}

private boolean validateUpdateUsuarioForm(String nombre, String email, String password, String repeatPassword) {
    boolean result = true;
    if(email.isEmpty()) {
        result = false;
    }
    if(nombre.isEmpty()) {
        result = false;
    }
    if(password.isEmpty()) {
        result = false;
    }
    if(repeatPassword.isEmpty()) {
        result = false;
    }
    if(result && !password.equals(repeatPassword)) {
        result = false;
    }
    return result;
}
```

Eliminación del usuario logado:

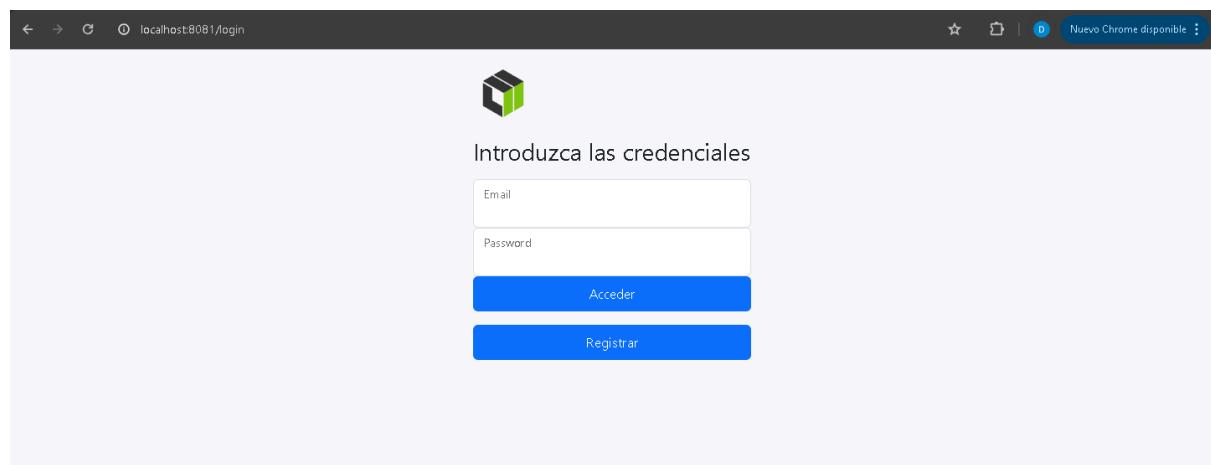
Una vez logueado, desde cualquier punto de la aplicación podemos acceder a la gestión de nuestro usuario.

En la misma tendremos dos operativas, en este caso el borrado del usuario.



The screenshot shows two browser windows. The top window is at `localhost:8081/home`, displaying a navigation bar with Home, Contenedores, Items, and a user icon with a dropdown menu containing 'Gestionar Usuario' and 'Logout'. The bottom window is at `localhost:8081/usuario/update`, showing a form for updating a user. The form fields are: Nombre (pruebaweb modificado), Email (pruebaweb@localhost.com), Password (redacted), Repite Password (redacted), Fecha alta (07-06-2024 15:18), Ultimo acceso (07-06-2024 15:30), Ultima modificacion (07-06-2024 15:27). Below the form are two blue buttons: 'Modificar' and 'Eliminar'.

Una vez completado el borrado nos retorna al login de nuevo quedando el usuario no accesible



The screenshot shows a login page at `localhost:8081/login`. It features a logo, a title 'Introduzca las credenciales', and a form with Email and Password fields. Below the form are two blue buttons: 'Acceder' and 'Registrar'.

email	nombre	password	insert_date	active	admin	last_login
admin@localhost.com	admin	8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a...	2024-06-07 00:46:46.000000	1	1	2024-06-0
agapitodisousa6@gmail.com	Agapito Di Sousa	5994471abb01112afcc18159f6cc74b4f51b99806da59b3ca...	2024-05-31 13:32:00.000000	1	1	2024-06-0
david@localhost.com	david	07d046d5fac12b3f82da5035b9aae86db5adc8275ebfb05e...	2024-06-04 00:34:51.000000	1	0	2024-06-0
david1@localhost.com	david1	bca563c46da4b684e4c29950c33263017e94bd58974207e4f...	2024-06-04 21:10:04.000000	0	0	NULL
david3@localhost.com	david3	07d046d5fac12b3f82da5035b9aae86db5adc8275ebfb05e...	2024-06-05 18:58:16.000000	0	0	NULL
juan@localhost.com	juan	ed08c290d7e22f7bb324b15cbadce35b0b348564fd2d5f9575...	NULL	1	0	2024-06-0
pepe@localhost.com	pepe	7c9e7c1494b2684ab7c19d6aff737e460fa9e98d5a234da131...	NULL	1	1	2024-05-2
pepi@localhost.com	pepi	e899cf89ab27d2105413963a8f40bcd2635250aeabfd9b409b...	2024-05-28 19:33:03.000000	0	0	NULL
prueba@localhost.com	prueba	655e786674d9d3e77bc05ed1de37b4b6bc89f788829f93c67...	2024-05-27 00:00:00.000000	1	0	2024-05-2

Esta gestión es realizada en el mismo controller que la operativa general de usuarios.

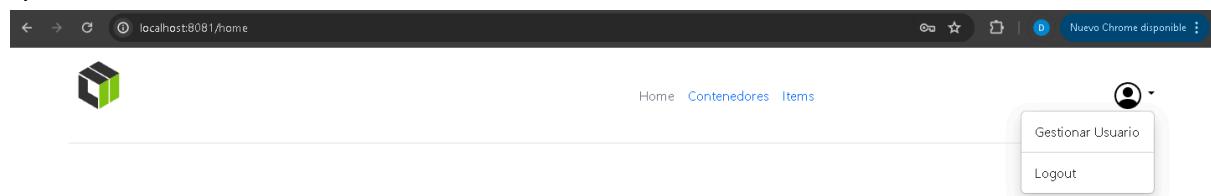
```
@GetMapping("/usuario/delete")
public String getDeleteUsuario(HttpServletRequest session, Model model) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        apiService.deleteUsuario(((Usuario)session.getAttribute("usuarioLogin")).getEmail());
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/logout";
}
}
```

Desconectar sesión

Una vez logueado, desde cualquier punto de la aplicación podemos acceder a la desconexión de nuestro usuario.

Esta acción no tiene contraparte en backend, por que las aplicaciones Rest no tienen estado, esto significa que el encargado de mantener la sesión activa es el dispositivo mediante el envío de los datos del usuario petionario en cada llamada.

El proceso de desconexión es el borrado de los datos del usuario en la sesión de la aplicación web





```
@GetMapping("/logout")
public String getLogout(HttpServletRequest session, Model model) {
    session.invalidate();
    apiService.removeInterceptor();
    return "redirect:/login";
}
```

Listar Contenedores y filtrado de los mismos

Dentro de la aplicación el usuario podrá realizar una búsqueda de los contenedores existentes en la aplicación.

Dentro de la búsqueda el usuario podrá actualizar la búsqueda mediante el botón de búsqueda y filtrar por cualquier texto contenido en la información de los contenedores.

A screenshot of a web browser showing a list of containers. The page has a header with links for Home, Usuarios, Contenedores, and Items. A user icon is also present. The main content area has a blue header bar with the word "Alta" and a search bar. Below is a table with the following data:

Id	Nombre	Descripción	Acciones
1	armario1	armario1	
2	balda1 armario1	balda1 armario1	
3	balda2 armario1	balda2 armario1	
4	balda3 armario1	balda3 armario1	
5	balda4 armario1	balda4 armario1	
6	armario2	armario2	
7	balda1 armario2	balda1 armario2	
8	balda2 armario2	balda2 armario2	
9	balda3 armario2	balda3 armario2	

Alta			
<input type="text" value="puma"/> Search icon			
Id	Nombre	Descripcion	Acciones
19	caja zapatos puma	caja zapatos puma modificado	Edit icon Delete icon

Esta gestión es realizada por el controller de contenedores.

```

@GetMapping("/contenedores")
public String homeContenedores(HttpServletRequest session, Model model) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    return "redirect:/contenedores/view";
}

@GetMapping("/contenedores/view")
public String getContenedores(HttpServletRequest session, Model model, @RequestParam(value="query", required = false) String query) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        if (query==null) {
            model.addAttribute("contenedores", apiService.getContenedores());
        }
        else {
            model.addAttribute("contenedores", apiService.getContenedores(query));
            model.addAttribute("query", query);
        }
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "contenedores/view";
}

```

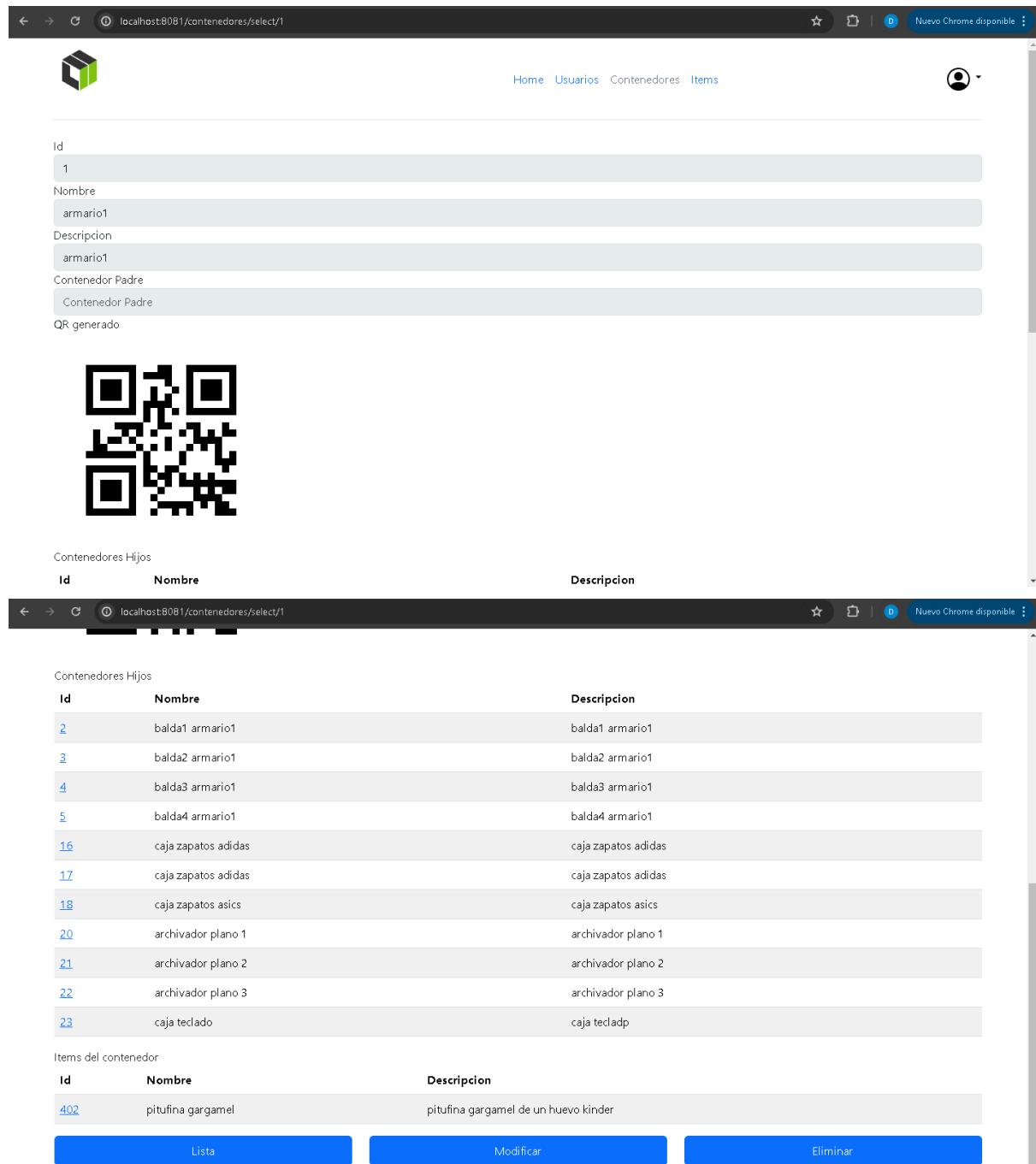
Información del contenedor

Dentro de la aplicación el usuario podrá realizar una consulta de un contenedor existente en la aplicación.

El acceso a la pantalla de consulta del contenedor se realizará desde la búsqueda lista de contenedores podrá seleccionar un registro llevando a esta pantalla.

Alta			
<input type="text" value="armario1"/> Search icon			
Id	Nombre	Descripcion	Acciones
1	armario1	armario1	Edit icon Delete icon
2	balda1 armario1	balda1 armario1	Edit icon Delete icon
3	balda2 armario1	balda2 armario1	Edit icon Delete icon
4	balda3 armario1	balda3 armario1	Edit icon Delete icon
5	balda4 armario1	balda4 armario1	Edit icon Delete icon

Dentro de la consulta el usuario podrá consultar el id, nombre, descripción y contenedor al que pertenece, la lista de contenedores asociados o ítems guardados en dicho contenedor y por último el QR generado de dicho contenedor.



The screenshot shows a web application interface for managing containers. At the top, there is a header bar with navigation links: Home, Usuarios, Contenedores, and Ítems. On the right side of the header is a user profile icon.

The main content area displays the following information:

- Container Details:**
 - Id: 1
 - Nombre: armario1
 - Descripción: armario1
 - Contenedor Padre: Contenedor Padre
 - QR generado: A large QR code is displayed.
- Container Children:**

Contenedores Hijos

Id	Nombre	Descripción
2	balda1 armario1	balda1 armario1
3	balda2 armario1	balda2 armario1
4	balda3 armario1	balda3 armario1
5	balda4 armario1	balda4 armario1
16	caja zapatos adidas	caja zapatos adidas
17	caja zapatos adidas	caja zapatos adidas
18	caja zapatos asics	caja zapatos asics
20	archivador plano 1	archivador plano 1
21	archivador plano 2	archivador plano 2
22	archivador plano 3	archivador plano 3
23	caja teclado	caja tecladp
- Items of the Container:**

Ítems del contenedor

Id	Nombre	Descripción
402	pitufina garmamel	pitufina garmamel de un huevo kinder

At the bottom of the page are three blue buttons: "Lista", "Modificar", and "Eliminar".

Esta gestión es realizada por el controller de contenedores.

```
@GetMapping("/contenedores/select/{id}")
public String getContenedor(HttpServletRequest session, Model model, @PathVariable Long id) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        model.addAttribute("contenedor", apiService.getContenedor(id));
        model.addAttribute("contenedorContenedorHijos", apiService.getContenedorContenedorHijos(id));
        model.addAttribute("contenedorItems", apiService.getContenedorItems(id));
        return "contenedores/select";
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/contenedores/view";
}
```

Alta de contenedor

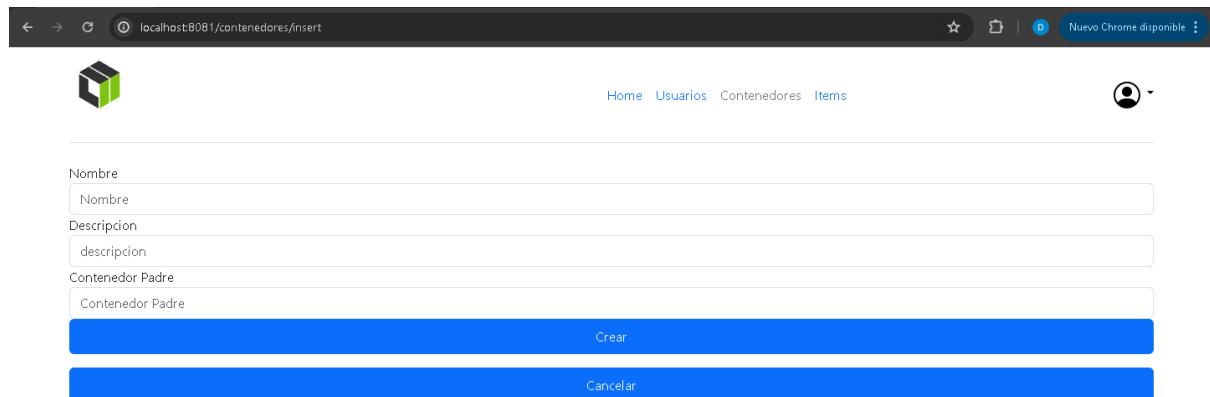
Dentro de la aplicación el usuario podrá realizar un alta de un contenedor.

El acceso a la pantalla de alta del contenedor se realizará desde la búsqueda lista de contenedores desde donde dispone de un botón que le dará acceso al formulario de alta.

Id	Nombre	Descripcion	Acciones
1	armario1	armario1	
2	balda1 armario1	balda1 armario1	
3	balda2 armario1	balda2 armario1	
4	balda3 armario1	balda3 armario1	
5	balda4 armario1	balda4 armario1	
6	armario2	armario2	
7	balda1 armario2	balda1 armario2	
8	balda2 armario2	balda2 armario2	
9	balda3 armario2	balda3 armario2	

Dentro de la pantalla de alta el usuario podrá digitar el nombre, descripción y contenedor al que pertenece.

El campo id del contenedor es único autogenerado por la aplicación.



localhost:8081/contenedores/insert

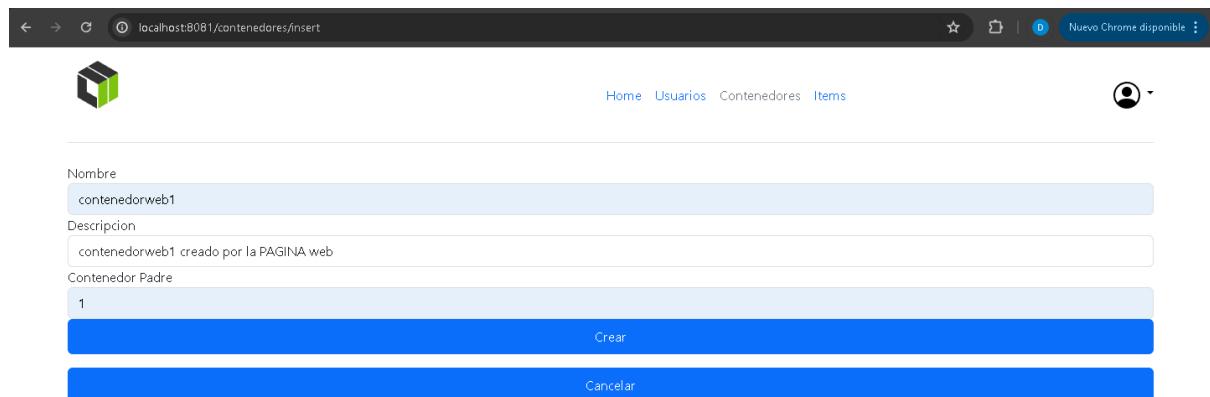
Nombre
Nombre

Descripcion
descripcion

Contenedor Padre
Contenedor Padre

Crear

Cancelar



localhost:8081/contenedores/insert

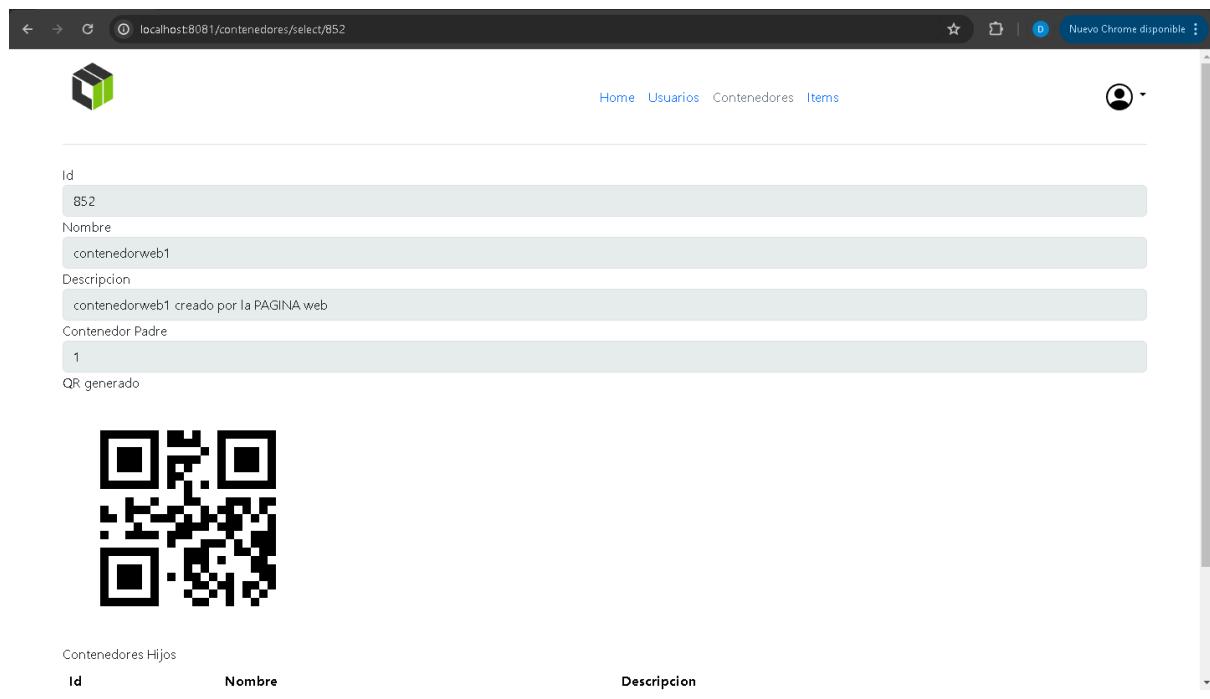
Nombre
contenedorweb1

Descripcion
contenedorweb1 creado por la PAGINA web

Contenedor Padre
1

Crear

Cancelar



localhost:8081/contenedores/select/852

Id
852

Nombre
contenedorweb1

Descripcion
contenedorweb1 creado por la PAGINA web

Contenedor Padre
1

QR generado

Contenedores Hijos

Id	Nombre	Descripcion

Esta gestión es realizada por el controller de contenedores.

```
@GetMapping("/contenedores/insert")
public String getInsertContenedor(HttpServletRequest session, Model model) {
    if(!securityInterceptor.validateSession(session, model))return "redirect:/logout";
    return "contenedores/insert";
}

@PostMapping("/contenedores/insert")
public String postInsertContenedor(HttpServletRequest session, Model model, @RequestParam(value="nombre") String nombre, @RequestParam(value="descri
try {
    Contenedor contenedor = insertContenedor(nombre, descripcion, sContenedorPadreId);
    if (contenedor != null) {
        return "redirect:contenedores/select/" + contenedor.getId();
    }
} catch (HttpClientErrorException.Unauthorized e) {
    model.addAttribute("error", "error - " + e.getMessage());
    return "redirect:/logout";
}
catch (Exception e) {
    e.printStackTrace();
    model.addAttribute("error", "error - " + e.getMessage());
}
Contenedor contenedorPadre = (sContenedorPadreId.isEmpty())?null:new Contenedor(Long.valueOf(sContenedorPadreId), null, null, null, null, null);
model.addAttribute("contenedor", new Contenedor(null, nombre, descripcion, contenedorPadre, null, null));
return "contenedores/insert";
}

private Contenedor insertContenedor(String nombre, String descripcion, String sContenedorPadreId) throws Exception{
if(validateInsertContenedorForm(nombre, descripcion, sContenedorPadreId)){
    Contenedor contenedorPadre = (sContenedorPadreId.isEmpty())?null:new Contenedor(Long.valueOf(sContenedorPadreId), null, null, null, null, null);
    Contenedor contenedor=new Contenedor(0l, nombre, descripcion, contenedorPadre, null, null);
    return apiService.insertContenedor(contenedor);
}
else {
    return null;
}
}

private boolean validateInsertContenedorForm(String nombre, String descripcion, String sContenedorPadreId){
boolean result=true;
if(nombre.isEmpty()){
    result = false;
}
if(!sContenedorPadreId.isEmpty()){
    try{
        Long.valueOf(sContenedorPadreId);
    }catch (NumberFormatException e) {
        result = false;
    }
}
return result;
}
```

Modificación de contenedor

Dentro de la aplicación el usuario podrá realizar una modificación de un contenedor. El acceso a la pantalla de modificación del contenedor se realizará desde la consulta del propio contenedor desde donde dispone de un botón modificar o pulsando el botón lápiz del registro en el listado que le dará acceso al formulario.

The screenshot shows two views of a container modification interface. The top view is a list view titled 'Alta' (Create) with a search bar. It shows one item: 'contenedorweb1' with description 'contenedorweb1 creado por la PAGINA web'. The bottom view is a detailed edit form for 'contenedorweb1' with fields for 'Nombre' (contenedorweb1), 'Descripción' (contenedorweb1 creado por la PAGINA web), and 'Contenedor Padre' (1). It includes a QR code and a section for 'Contenedores Hijos' which is currently empty. At the bottom are buttons for 'Lista', 'Modificar', and 'Eliminar'.

Dentro de la pantalla de modificación el usuario podrá digitar el nombre, descripción y contenedor al que pertenece.

El campo id del contenedor es único y no se puede modificar.

This screenshot shows the container modification interface with an invalid ID. The 'Id' field contains '852' instead of the expected unique value. The other fields ('Nombre' and 'Descripción') are correctly filled. The 'Contenedor Padre' dropdown is set to '2'. At the bottom are 'Cancelar' and 'Modificar' buttons.

Como resultado de la operación, nos llevará a la consulta del contenedor que hemos modificado.

The screenshot shows a web application interface for managing containers. At the top, there's a header bar with a logo, navigation links for Home, Usuarios, Contenedores, and Items, and a user profile icon. Below the header, a form displays the following data:

Id	852
Nombre	contenedorweb1modificado
Descripción	contenedorweb1 creado por la PAGINA web
Contenedor Padre	2
QR generado	

Below the form, there's a section titled "Contenedores Hijos" (Container Children) with a table:

Id	Nombre	Descripción

Esta gestión es realizada por el controller de contenedores.

```
@GetMapping("/contenedores/update/{id}")
public String getUpdateContenedor(HttpServletRequest session, Model model, @PathVariable Long id) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        model.addAttribute("contenedor", apiService.getContenedor(id));
        return "contenedores/update";
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/contenedores/view";
}

@PostMapping("/contenedores/update/{id}")
public String postUpdateContenedor(HttpServletRequest session, Model model, @PathVariable Long id, @RequestParam(value="nombre") String nombre, @RequestPart("contenedorPadreId") Long sContenedorPadreId) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        Contenedor contenedor = updateContenedor(id, nombre, descripcion, sContenedorPadreId);
        if (contenedor != null) {
            return "redirect:/contenedores/select/" + contenedor.getId();
        }
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    Contenedor contenedorPadre = (sContenedorPadreId.isEmpty())?null:new Contenedor(Long.valueOf(sContenedorPadreId), null, null, null, null);
    model.addAttribute("contenedor", new Contenedor(id, nombre, descripcion, contenedorPadre, null, null));
    return "contenedores/update";
}
```

```

private Contenedor updateContenedor(long id, String nombre, String descripcion, String sContenedorPadreId) throws Exception{
    if(validateUpdateContenedorForm(id, nombre, descripcion, sContenedorPadreId)){
        Contenedor contenedor = apiService.getContenedor(id);
        Contenedor contenedorPadre = (sContenedorPadreId.isEmpty())?null:new Contenedor(Long.valueOf(sContenedorPadreId), null, null, null);
        contenedor.setNombre(nombre);
        contenedor.setDescripcion(descripcion);
        contenedor.setContenedorPadre(contenedorPadre);
        return apiService.updateContenedor(id, contenedor);
    }
    else {
        return null;
    }
}

private boolean validateUpdateContenedorForm(long id, String nombre, String descripcion, String sContenedorPadreId){
    boolean result = true;
    if(nombre.isEmpty()){
        result = false;
    }
    if(!sContenedorPadreId.isEmpty()){
        try{
            Long.valueOf(sContenedorPadreId);
        }catch (NumberFormatException e) {
            result = false;
        }
    }
    return result;
}

```

Borrado de contenedor

Dentro de la aplicación el usuario podrá realizar el borrado de un contenedor. El acceso a la operativa de borrado del contenedor se realizará desde la consulta del propio contenedor desde donde dispone de un botón borrar o pulsando el botón papelera del registro en el listado que le dará acceso a la acción directamente.

The screenshot shows the application's interface for managing containers. It includes a header with navigation links (Home, Usuarios, Contenedores, Items) and a search bar. Below the header is a blue header bar labeled "Alta". A search icon is located in the top right corner of this bar. The main content area displays a table with two rows of data:

ID	Nombre	Descripción	Acciones
19	caja zapatos puma	caja zapatos puma modificada	
852	contenedorweb1modificado	contenedorweb1 creado por la PAGINA web	

Below the table, there is a form for editing a container, showing fields for Nombre, Descripción, Contenedor Padre, and QR generado. A large QR code is displayed below the form. At the bottom of the page, there are sections for "Contenedores Hijos" and "Items del contenedor", each with their own tables and buttons for Lista, Modificar, and Eliminar.

Una vez ejecutado el borrado nos llevará a la pantalla de listado.

The screenshot shows the application's interface after the deletion of the container with ID 852. The header and search bar are visible. The main content area now displays a table with only one row of data:

ID	Nombre	Descripción	Acciones
19	caja zapatos puma	caja zapatos puma modificada	

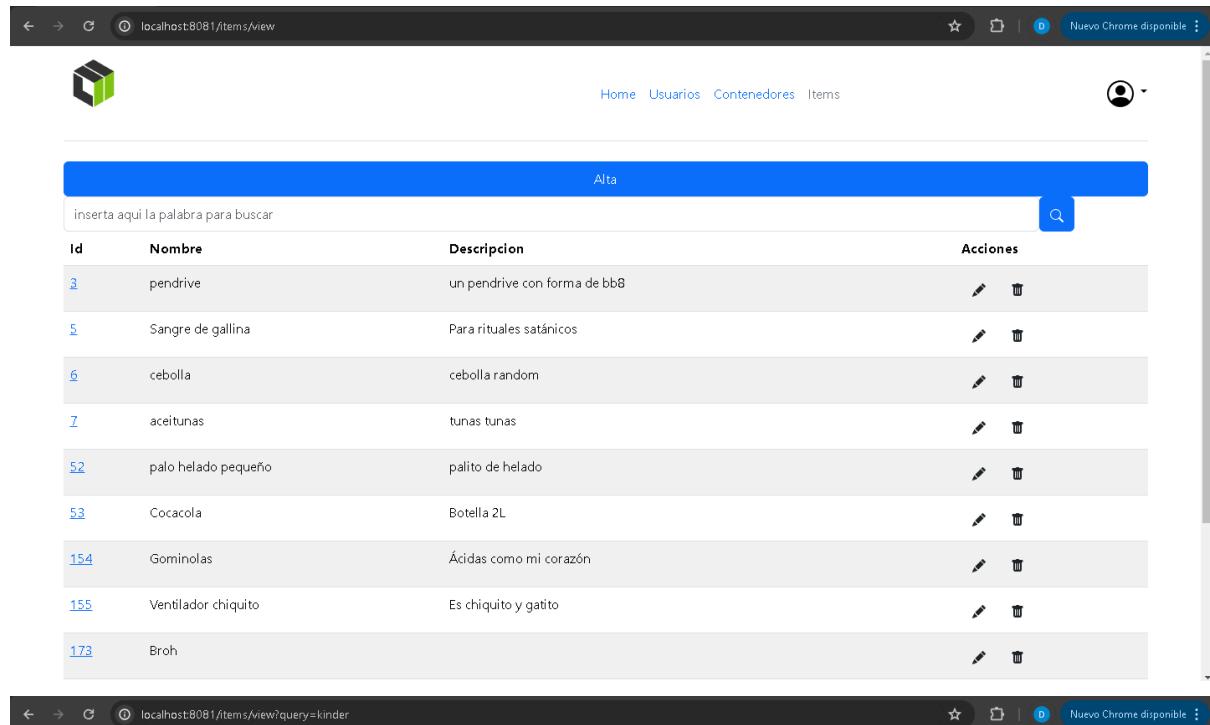
Esta gestión es realizada por el controller de contenedores.

```
@GetMapping("/contenedores/delete/{id}")
public String getDeleteContenedor(HttpServletRequest session, Model model, @PathVariable Long id) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        apiService.deleteContenedor(id);
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/contenedores/view";
}
```

Listar Objetos y filtrado de los mismos

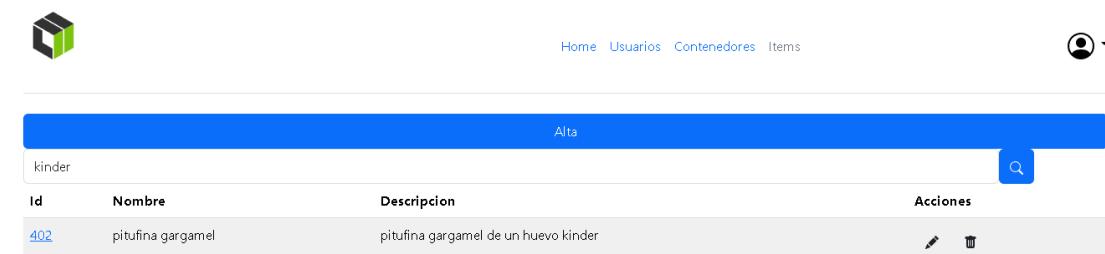
Dentro de la aplicación el usuario podrá realizar una búsqueda de los objetos existentes en la aplicación.

Dentro de la búsqueda el usuario podrá actualizar la búsqueda y filtrar por cualquier texto contenido en la información de los objetos.



The screenshot shows a web application interface. At the top, there is a navigation bar with links for Home, Usuarios, Contenedores, and Items. On the far right, there is a user profile icon. Below the navigation bar, there is a blue header bar with the word 'Alta' in white. To the right of this bar is a search input field with a magnifying glass icon. Below the header, there is a table with columns: Id, Nombre, Descripción, and Acciones. The table contains several rows of data. At the bottom of the page, there is a footer bar with the same navigation links as the top bar.

Id	Nombre	Descripción	Acciones
3	pendrive	un pendrive con forma de bb8	
5	Sangre de gallina	Para rituales satánicos	
6	cebolla	cebolla random	
7	aceitunas	tunas tunas	
52	palo helado pequeño	palito de helado	
53	Cocacola	Botella 2L	
154	Gominolas	Ácidas como mi corazón	
155	Ventilador chiquito	Es chiquito y gatito	
173	Broh		



This screenshot shows the same web application interface as the previous one, but with a different search term. The search bar at the top now contains the word 'kinder'. The table below shows a single row of data that matches this search term.

Id	Nombre	Descripción	Acciones
402	pitufina gargamel	pitufina gargamel de un huevo kinder	

Esta gestión es realizada por el controller de items.

```
@GetMapping("/items")
public String homeItems(HttpServletRequest session, Model model) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    return "redirect:/items/view";
}

@GetMapping("/items/view")
public String getItems(HttpServletRequest session, Model model, @RequestParam(required = false) String query) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        if (query==null) {
            model.addAttribute("Items", apiService.getItems());
        }
        else {
            model.addAttribute("Items", apiService.getItems(query));
            model.addAttribute("query", query);
        }
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "items/view";
}
```

Información del objeto

Dentro de la aplicación el usuario podrá realizar una consulta de un objeto existente en la aplicación.

El acceso a la pantalla de consulta del objeto se realizará desde varios puntos.

El primero desde la búsqueda listada de objetos podrá seleccionar un registro llevando a esta pantalla.

El segundo desde la consulta de objetos asociados a un contenedor.

The screenshot displays two separate browser windows side-by-side, both showing parts of a web application interface.

Top Window (localhost:8081/items/view?query=kinder):

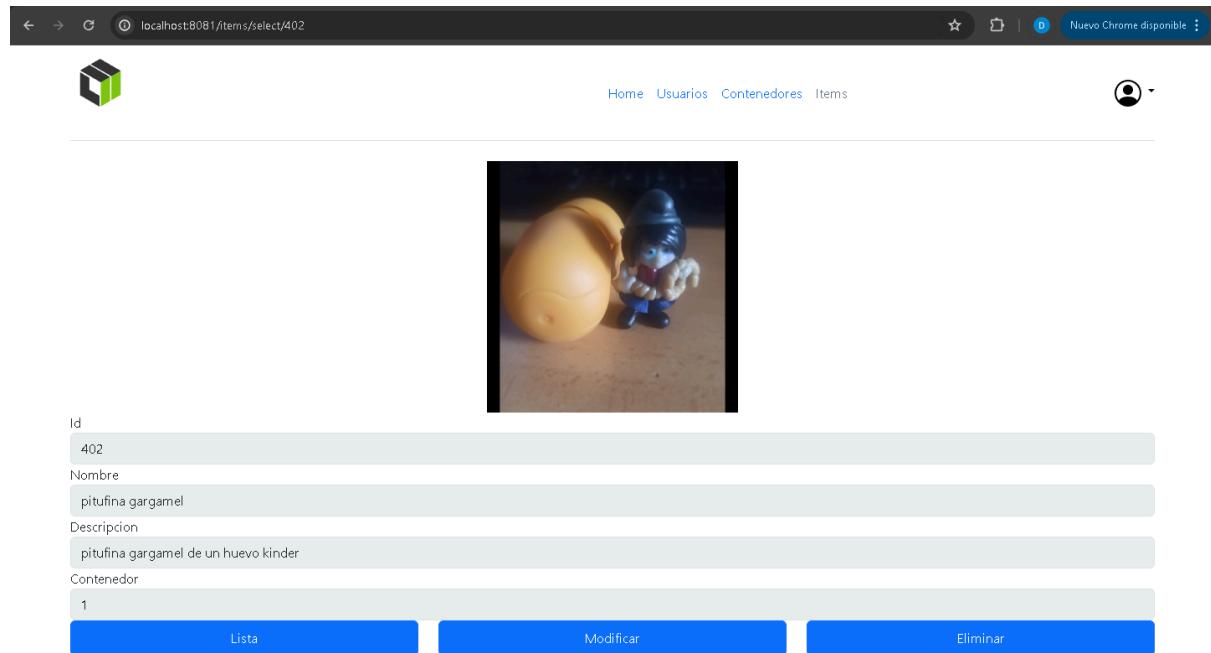
- Header:** Shows a logo, navigation links (Home, Usuarios, Contenedores, Items), and a user icon.
- Section Title:** "Alta" (Add) in a blue header bar.
- Table:** A grid showing a single item with columns: Id, Nombre, Descripción, and Acciones (Edit, Delete).
- Data:** Row 402: pitufina gargamel, pitufina gargamel de un huevo kinder.

Bottom Window (localhost:8081/contenedores/select/1):

- Header:** Shows a logo, navigation links (Home, Usuarios, Contenedores, Items), and a user icon.
- Section Title:** "Contenedores Hijos" (Container Children).
- Table:** A grid listing various containers under a single parent container.
- Data:** Rows include: balda1 armario1, balda2 armario1, balda3 armario1, balda4 armario1, caja zapatos adidas, caja zapatos adidas, caja zapatos asics, archivador plano 1, archivador plano 2, archivador plano 3, and caja tecladp.
- Section Title:** "Items del contenedor" (Container Items).
- Table:** A grid showing items contained within the selected container.
- Data:** Row 402: pitufina gargamel, pitufina gargamel de un huevo kinder.

Buttons at the bottom: Lista, Modificar, Eliminar.

Dentro de la consulta el usuario podrá consultar la imagen asociada si la tiene, el id, nombre, descripción y contenedor al que pertenece.



localhost:8081/items/select/402

Home Usuarios Contenedores Items

Id
402

Nombre
pitufina garmel

Descripción
pitufina garmel de un huevo kinder

Contenedor
1

[Lista](#) [Modificar](#) [Eliminar](#)

Esta gestión es realizada por el controller de items.

```
@GetMapping("/items/select/{id}")
public String getItem(HttpServletRequest session, Model model, @PathVariable Long id) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        model.addAttribute("item", apiService.getItem(id));
        return "items/select";
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/items/view";
}

@GetMapping("/items/select/{id}/imagen")
public ResponseEntity<byte[]> obtenerImagen(HttpServletRequest session, Model model, @PathVariable Long id) {
    if(!SecurityInterceptor.validateSession(session, model))return null;
    try{
        return apiService.getItemImagen(id);
    }
    catch (HttpClientErrorException.Unauthorized e) {
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

Y la vista puede cargar la imagen como cualquier recurso web, pero securizado

```

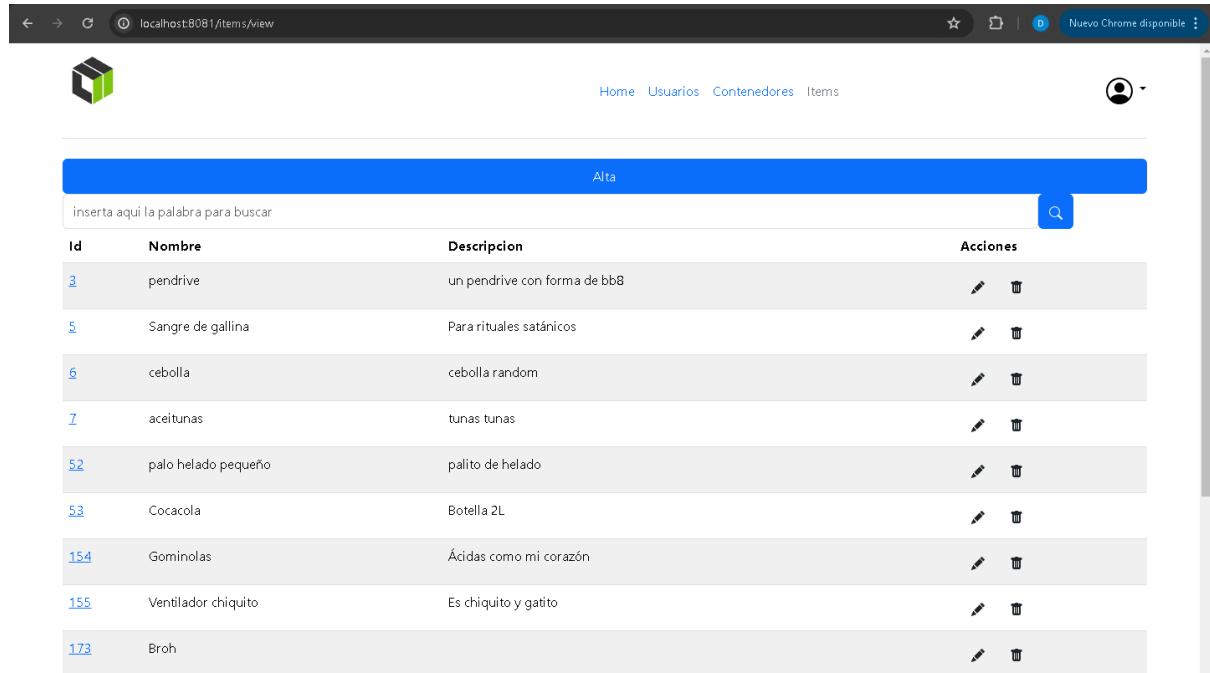
8         <div>
9             <div class="row">
10                 <div class="col">
11                     <div class="form-group has-danger" style="text-align: center;">
12                         
13                     </div>
14                 </div>
15             </div>
16         </div>

```

Alta de objeto

Dentro de la aplicación el usuario podrá realizar un alta de un objeto.

El acceso a la pantalla de alta del objeto se realizará desde la búsqueda lista de objetos desde donde dispone de un botón que le dará acceso al formulario de alta.

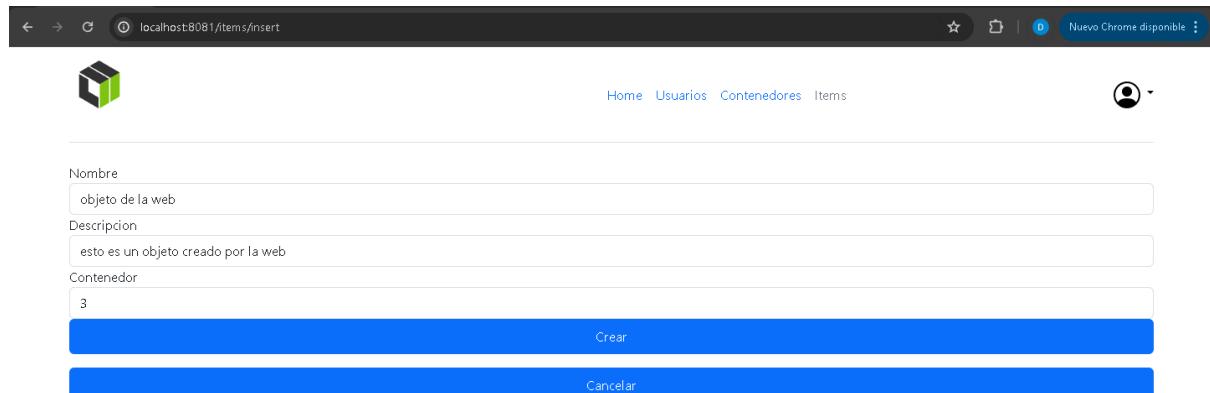


The screenshot shows a web browser window with the URL `localhost:8081/items/view`. The page title is "Alta". A search bar at the top contains the placeholder "inserta aqui la palabra para buscar". Below the search bar is a table with the following columns: **Id**, **Nombre**, **Descripción**, and **Acciones**. The table lists 173 items. Each row contains an ID link, a name, a description, and edit/delete icons. The last row, "Broh", is highlighted with a blue background.

Id	Nombre	Descripción	Acciones
3	pendrive	un pendrive con forma de bb8	
5	Sangre de gallina	Para rituales satánicos	
6	cebolla	cebolla random	
7	aceitunas	tunas tunas	
52	palo helado pequeño	palito de helado	
53	Cocacola	Botella 2L	
154	Gominolas	Ácidas como mi corazón	
155	Ventilador chiquito	Es chiquito y gatito	
173	Broh		

Dentro de la pantalla de alta el usuario podrá digitar el nombre, descripción y contenedor al que pertenece.

En este caso, vemos que no podemos tomar una imagen por lo que no podremos darla de alta desde la web.



The screenshot shows a web browser window with the URL `localhost:8081/items/insert`. The page title is "Alta". The form has three input fields: **Nombre** (value: "objeto de la web"), **Descripción** (value: "esto es un objeto creado por la web"), and **Contenedor** (value: "3"). At the bottom of the form are two buttons: a large blue "Crear" button and a smaller blue "Cancelar" button.

El campo id del objeto es único autogenerado por la aplicación.

localhost:8081/items/select/502

Home Usuarios Contenedores Items

Imagen Item

Id
502

Nombre
objeto de la web

Descripción
esto es un objeto creado por la web

Contenedor
3

[Lista](#) [Modificar](#) [Eliminar](#)

Esta gestión es realizada por el controller de items.

```
@GetMapping("/items/insert")
public String getInsertItem.HttpSession session, Model model) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    return "items/insert";
}

@PostMapping("/items/insert")
public String postInsertItem.HttpSession session, Model model, @RequestParam(value="nombre") String nombre, @RequestParam(value="descripcion"
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        Item item = insertItem(nombre, descripcion, sContenedorId);
        if (item != null) {
            return "redirect:/items/select/" + item.getId();
        }
    } catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    } catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    Contenedor contenedor = (sContenedorId.isEmpty())?null:new Contenedor(Long.valueOf(sContenedorId), null, null, null, null, null);
    model.addAttribute("item", new Item(null, nombre, descripcion, contenedor));
    return "items/insert";
}

private Item insertItem(String nombre, String descripcion, String sContenedorId) throws Exception{
    if(validateInsertItemForm(nombre, descripcion, sContenedorId)){
        Contenedor contenedor = (sContenedorId.isEmpty())?null:new Contenedor(Long.valueOf(sContenedorId), null, null, null, null, null);
        Item item=new Item(0l, nombre, descripcion, contenedor);

        return apiService.insertItem(item);
    } else{
        return null;
    }
}

private boolean validateInsertItemForm(String nombre, String descripcion, String sContenedorId) {
    boolean result=true;
    if(nombre.isEmpty()){
        result = false;
    }
    if(!sContenedorId.isEmpty()){
        try{
            Long.valueOf(sContenedorId);
        }catch (NumberFormatException e) {
            result = false;
        }
    }
    return result;
}
```

Modificación de objeto

Dentro de la aplicación el usuario podrá realizar una modificación de un objeto.

El acceso a la pantalla de modificación del objeto se realizará desde la consulta del propio objeto desde donde dispone de un botón modificar o pulsando el botón lápiz del registro en el listado que le dará acceso al formulario.

The screenshot shows two consecutive views of the application's modification screen. The first view is a list view titled 'Alta' (Create) with a single item: 'Id: 502, Nombre: objeto de la web, Descripción: esto es un objeto creado por la web'. The second view is a detailed modification form for the same object, showing fields for 'Nombre' (Nombre), 'Descripción' (Descripción), and 'Contenedor' (Container). The 'Nombre' field contains 'objeto de la web' and has a placeholder 'Imagen Item'. The 'Descripción' field contains 'esto es un objeto creado por la web'. The 'Contenedor' field contains '3'. At the bottom are three buttons: 'Lista' (List), 'Modificar' (Modify), and 'Eliminar' (Delete).

Dentro de la pantalla de modificación el usuario podrá digitar el nombre, descripción y contenedor al que pertenece.

El campo id del objeto es único y no se puede modificar.

This screenshot shows the modification screen after changes have been made. The 'Nombre' field now contains 'objeto de la web modificado'. The other fields ('Descripción' and 'Contenedor') remain the same as in the previous screenshot. The bottom buttons are 'Cancelar' (Cancel) and 'Modificar' (Modify).

Como resultado de la operación, nos llevará a la consulta del objeto que hemos modificado.

The screenshot shows a web application interface. At the top, there's a header with a logo, navigation links for 'Home', 'Usuarios', 'Contenedores', and 'Items', and a user profile icon. Below the header, there's a table with a single row of data. The table has columns for 'Id' (containing '502'), 'Nombre' (containing 'objeto de la web modificado'), 'Descripción' (containing 'esto es un objeto modificado por la web'), and 'Contenedor' (containing '6'). To the right of the table are three buttons: 'Lista' (List), 'Modificar' (Modify), and 'Eliminar' (Delete). Above the table, there's a small placeholder for an image labeled 'Imagen Item'.

Esta gestión es realizada por el controller de items.

```
@GetMapping("/items/update/{id}")
public String getUpdateItem(HttpServletRequest session, Model model, @PathVariable Long id) {
    if(!securityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        model.addAttribute("item", apiService.getItem(id));
        return "items/update";
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/items/view";
}

@PostMapping("/items/update/{id}")
public String postUpdateItem(HttpServletRequest session, Model model, @PathVariable Long id, @RequestParam(value="nombre") String nombre, @RequestP...
try {
    Item item = updateItem(id, nombre, descripcion, sContenedorId);
    if (item != null) {
        return "redirect:/items/select/" + item.getId();
    }
}
catch (HttpClientErrorException.Unauthorized e) {
    model.addAttribute("error", "error - " + e.getMessage());
    return "redirect:/logout";
}
catch (Exception e) {
    e.printStackTrace();
    model.addAttribute("error", "error - " + e.getMessage());
}
Contenedor contenedor = (sContenedorId.isEmpty())?null:new Contenedor(Long.valueOf(sContenedorId), null, null, null, null, null);
model.addAttribute("item", new Item(id, nombre, descripcion, contenedor));
return "items/update";
}

private Item updateItem(long id, String nombre, String descripcion, String sContenedorId) throws Exception{
    if(validateUpdateItemForm(id, nombre, descripcion, sContenedorId)){
        Item item = apiService.getItem(id);
        Contenedor contenedor = (sContenedorId.isEmpty())?null:new Contenedor(Long.valueOf(sContenedorId), null, null, null, null, null);
        item.setNombre(nombre);
        item.setDescripcion(descripcion);
        item.setContenedor(contenedor);

        return apiService.updateItem(id, item);
    }
    else {
        return null;
    }
}

private boolean validateUpdateItemForm(long id, String nombre, String descripcion, String sContenedorId) throws Exception{
    boolean result = true;
    if(nombre.isEmpty()){
        result = false;
    }
    if(!sContenedorId.isEmpty()){
        try{
            Long.valueOf(sContenedorId);
        }catch (NumberFormatException e) {
            result = false;
        }
    }
    return result;
}
```

Borrado de objeto

Dentro de la aplicación el usuario podrá realizar el borrado de un objeto.

El acceso a la operativa de borrado del objeto se realizará desde la consulta del propio objeto desde donde dispone de un botón borrar o pulsando el botón papelera del registro en el listado que le dará acceso a la acción directamente.

Alta			
<input type="text" value="web modificado"/> <input type="button" value=""/>			
Id	Nombre	Descripcion	Acciones
502	objeto de la web modificado	esto es un objeto modificado por la web	<input type="button" value=""/> <input type="button" value=""/>

Imagen Item

Id 502	Nombre objeto de la web modificado	Descripcion esto es un objeto modificado por la web
Contenedor 6	<input type="button" value="Lista"/> <input type="button" value="Modificar"/> <input type="button" value="Eliminar"/>	

Una vez ejecutado el borrado nos llevará a la pantalla de listado.

Alta			
<input type="text" value="web modi"/> <input type="button" value=""/>			
Id	Nombre	Descripcion	Acciones

Esta gestión es realizada por el controller de items.

```
@GetMapping("/items/delete/{id}")
public String getDeleteItem(HttpServletRequest session, Model model, @PathVariable Long id) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        apiService.deleteItem(id);
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/items/view";
}
```

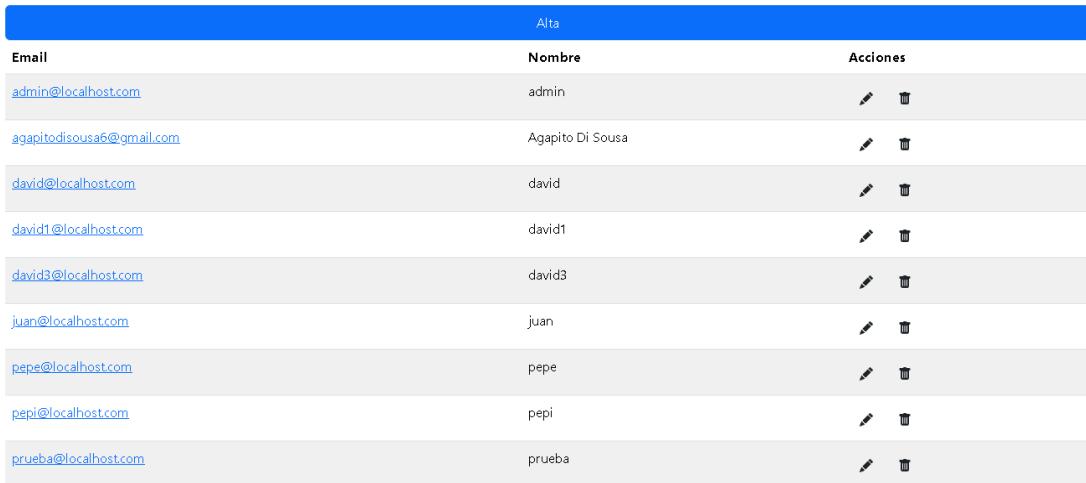
Listar Usuarios

Dentro de la aplicación el usuario con permisos de administrador podrá realizar una búsqueda de los usuarios existentes en la aplicación.



email	nombre	password	insert_date	active	admin	last_login
admin@localhost.com	admin	8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a8...	2024-06-07 00:46:46 000000	1		2024-06-07

localhost:8081/usuarios/view



Alta		
Email	Nombre	Acciones
admin@localhost.com	admin	
agapitodisousa6@gmail.com	Agapito Di Sousa	
david@localhost.com	david	
david1@localhost.com	david1	
david3@localhost.com	david3	
juan@localhost.com	juan	
pepe@localhost.com	pepe	
pepi@localhost.com	pepi	
prueba@localhost.com	prueba	

Esta gestión es realizada por el controller de usuarios, donde se comprueba que el usuario logado tenga permisos de administrador.

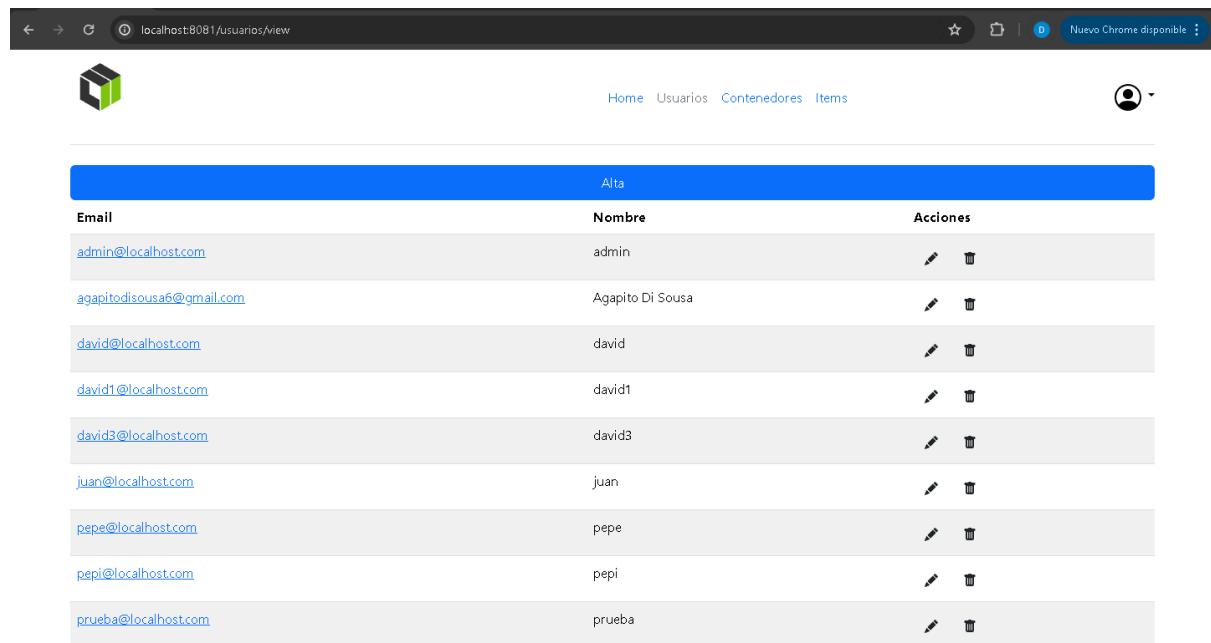
```
public String homeUsuarios(HttpServletRequest session, Model model) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    return "redirect:/usuarios/view";
}

@GetMapping("/usuarios/view")
public String getUsuarios(HttpServletRequest session, Model model) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        model.addAttribute("usuarios", apiService.getUsuarios());
    } catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "usuarios/view";
}
```

Información del usuario

Dentro de la aplicación el usuario podrá realizar una consulta de un usuario existente en la aplicación.

El acceso a la pantalla de consulta del usuario se realizará desde la búsqueda listada de usuarios seleccionando un registro y llevándolo a la pantalla.



Alta		
Email	Nombre	Acciones
admin@localhost.com	admin	
agapitodisousa6@gmail.com	Agapito Di Sousa	
david@localhost.com	david	
david1@localhost.com	david1	
david3@localhost.com	david3	
juan@localhost.com	juan	
pepe@localhost.com	pepe	
pepi@localhost.com	pepi	
prueba@localhost.com	prueba	

The screenshot shows a web application interface for managing users. At the top, the URL is `localhost:8081/usuarios/select/pepi@localhost.com`. The page title is "pepi". The navigation menu includes Home, Usuarios, Contenedores, and Items. On the right, there is a user profile icon.

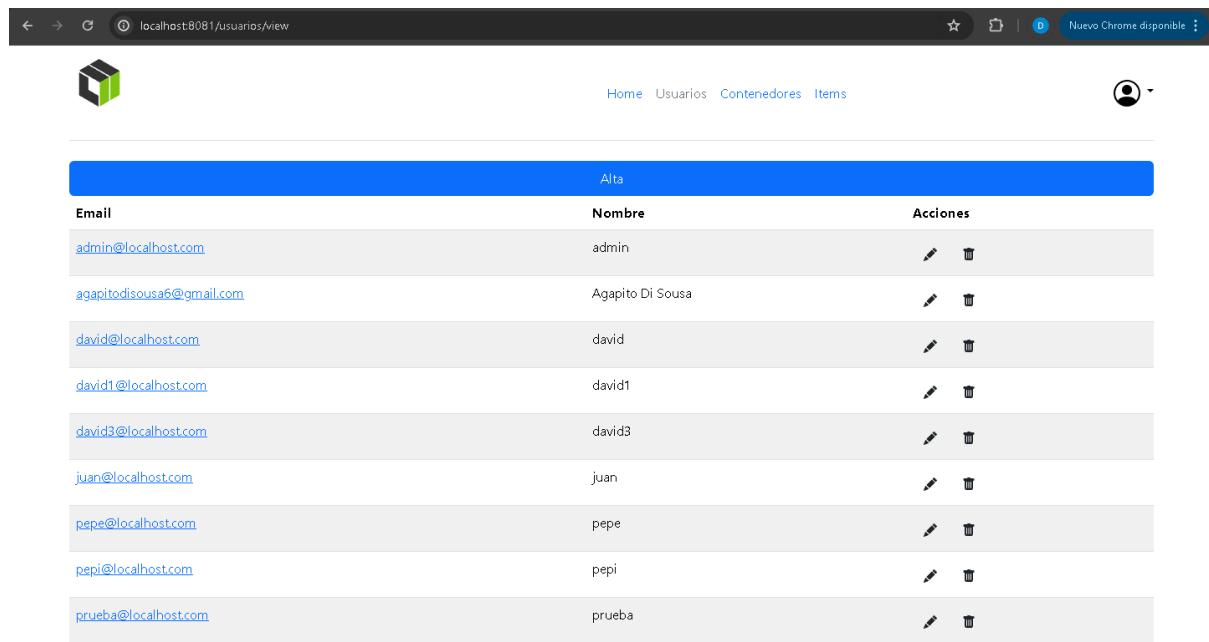
Nombre	pepi
Email	pepi@localhost.com
Password
Repite Password
Fecha alta	28-05-2024 17:33
Ultimo acceso	Ultimo acceso
Ultima modificación	Ultima modificación
<input type="checkbox"/> Activo	<input type="checkbox"/> Administrador
Lista	
Modificar	
Eliminar	

Dentro de la consulta el usuario podrá consultar el email, nombre, la fecha de alta, la modificación, la última conexión y los permisos que tiene dentro de la aplicación. Esta gestión es realizada por el controller de usuarios..

```
@GetMapping("/usuarios/select/{email}")
public String getUsuario(HttpServletRequest session, Model model, @PathVariable String email) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        model.addAttribute("usuario", apiService.getUsuario(email));
        return "usuarios/select";
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/usuarios/view";
}
```

Alta de usuario

Dentro de la aplicación el usuario podrá realizar un alta de otro usuario. El acceso a la pantalla de alta de usuario se realizará desde la búsqueda lista de usuario desde donde dispone de un botón que le dará acceso al formulario de alta.



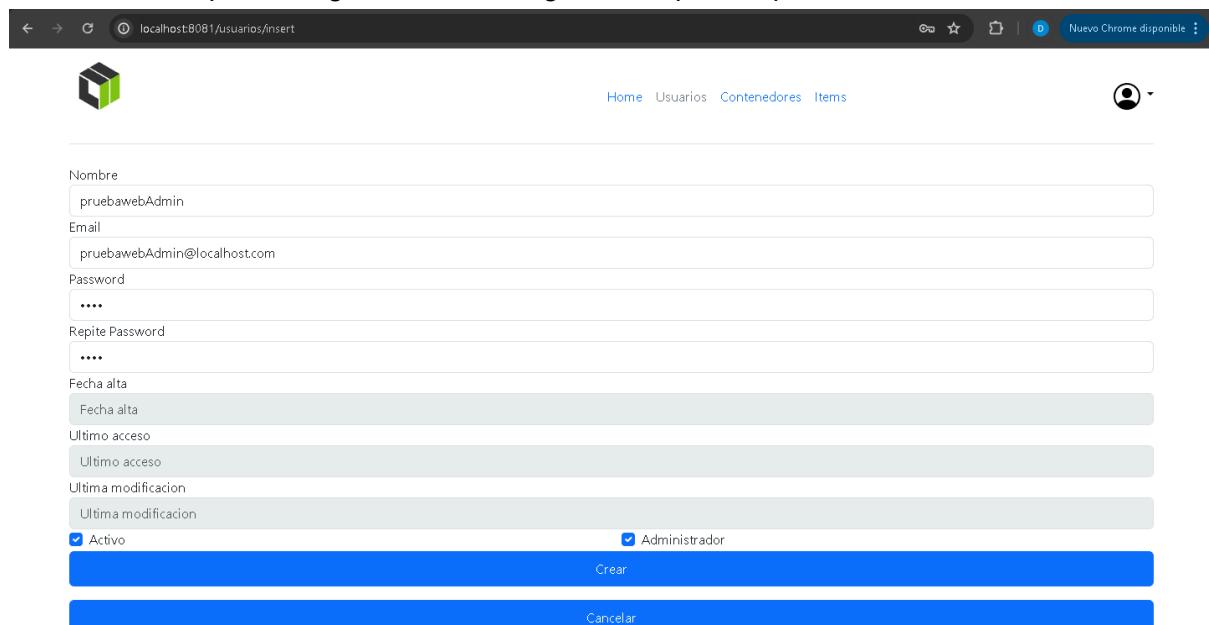
Email	Nombre	Acciones
admin@localhost.com	admin	
agapitodisousa6@gmail.com	Agapito Di Sousa	
david@localhost.com	david	
david1@localhost.com	david1	
david3@localhost.com	david3	
juan@localhost.com	juan	
pepe@localhost.com	pepe	
pepi@localhost.com	pepi	
prueba@localhost.com	prueba	

Dentro de la pantalla de alta el usuario podrá digitar el email, nombre, contraseñas y los permisos dentro de la aplicación.

El campo password y repeat password, se validará para que coincidan.

El email es único dentro de la aplicación.

El resto de campos, son gestionados íntegramente por la aplicación.



Nombre	pruebawebAdmin
Email	pruebawebAdmin@localhost.com
Password	****
Repite Password	****
Fecha alta	Fecha alta
Ultimo acceso	Ultimo acceso
Ultima modificación	Ultima modificación
<input checked="" type="checkbox"/> Activo	<input checked="" type="checkbox"/> Administrador
Crear	
Cancelar	

Una vez insertado nos retornará a la pantalla de consulta del usuario.

Nombre
pruebawebAdmin

Email
pruebawebAdmin@localhost.com

Password
.....

Repite Password
.....

Fecha alta
07-06-2024 16:20

Ultimo acceso

Ultima modificación

Active

Administrador

[Lista](#) [Modificar](#) [Eliminar](#)

Esta gestión es realizada por el controller de usuarios, donde se comprueba que el usuario logado tenga permisos de administrador.

```
@GetMapping("/usuarios/insert")
public String getInsertUsuario(HttpServletRequest session, Model model) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    return "usuarios/insert";
}

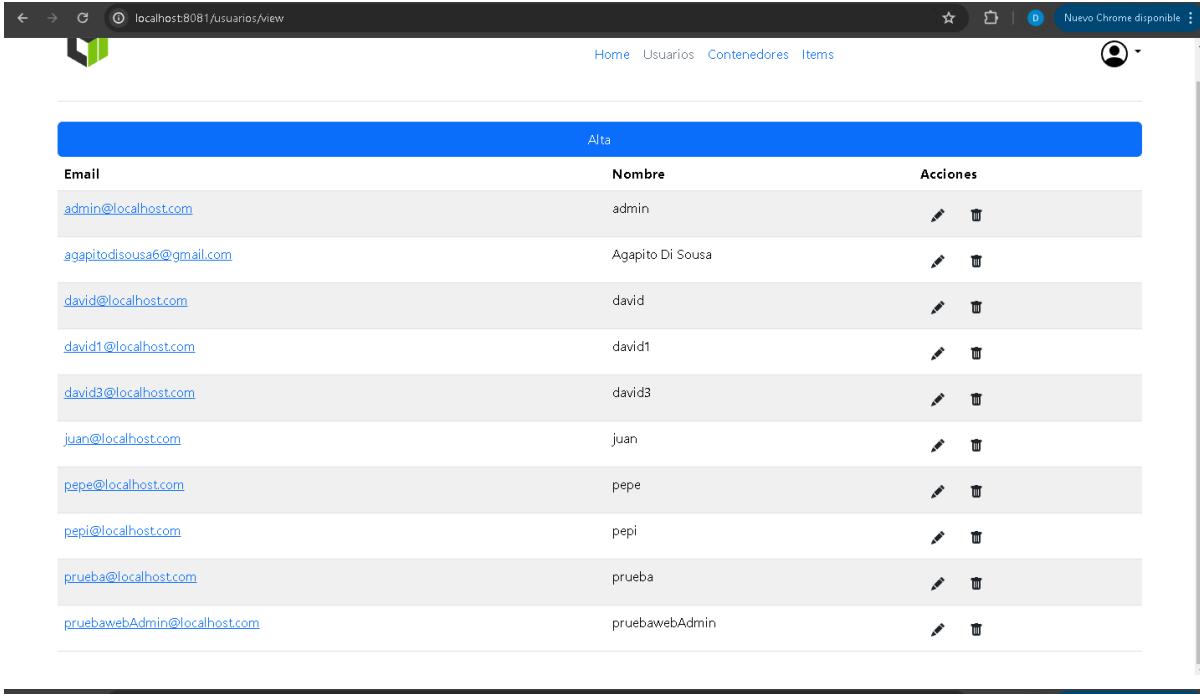
@PostMapping("/usuarios/insert")
public String postInsertUsuario(HttpServletRequest session, Model model, @RequestParam(value="nombre") String nombre, @RequestParam(value="email") String email, @RequestParam(value="password") String password, @RequestParam(value="repeatPassword") String repeatPassword, @RequestParam(value="active") boolean active, @RequestParam(value="admin") boolean admin) {
    try {
        Usuario usuario = insertUsuario(nombre, email, password, repeatPassword, active, admin);
        if (usuario != null) {
            return "redirect:/usuarios/select/" + usuario.getEmail();
        }
    } catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    } catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    model.addAttribute("usuario", new Usuario(email , nombre, password, active, admin));
    return "usuarios/insert";
}

private Usuario insertUsuario(String nombre, String email, String password, String repeatPassword, boolean active, boolean admin) throws Exception {
    if(validateInsertUsuarioForm(nombre, email, password, repeatPassword)){
        Usuario usuario = new Usuario(email, nombre, HashUtils.hashString(password), active, admin);
        return apiService.insertUsuario(usuario);
    } else {
        return null;
    }
}

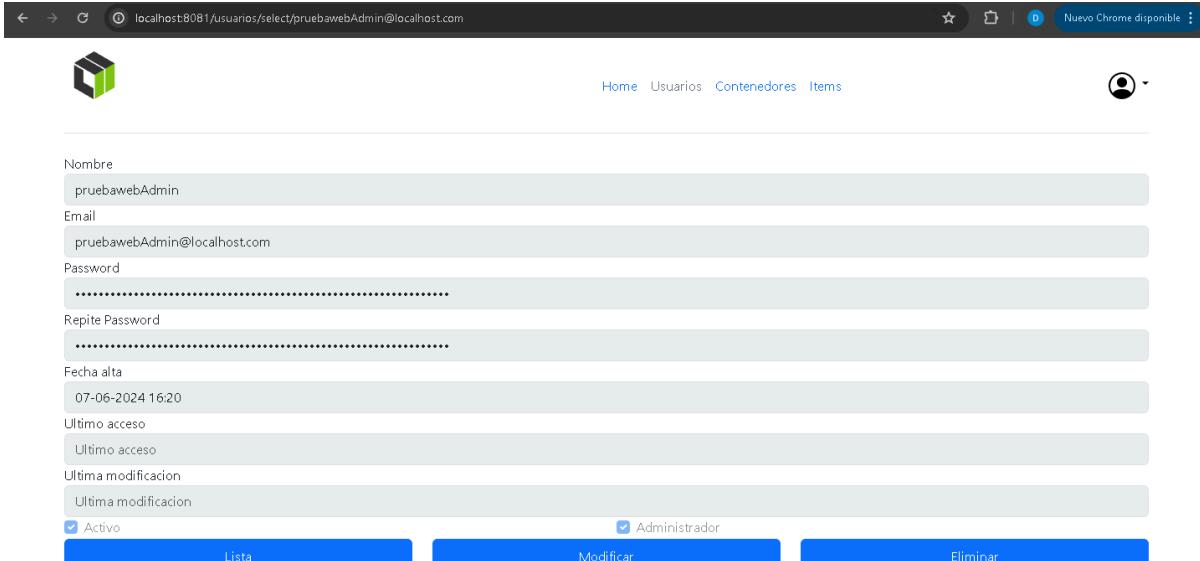
private boolean validateInsertUsuarioForm(String nombre, String email, String password, String repeatPassword) {
    boolean result = true;
    if(email.isEmpty()) {
        result = false;
    }
    if(nombre.isEmpty()) {
        result = false;
    }
    if(password.isEmpty()) {
        result = false;
    }
    if(repeatPassword.isEmpty()) {
        result = false;
    }
    if(result && !password.equals(repeatPassword)) {
        result = false;
    }
    return result;
}
```

Modificación de usuario

Dentro de la aplicación el usuario podrá realizar una modificación de otro usuario. El acceso a la pantalla de modificación del usuario se realizará desde la consulta del propio usuario a modificar desde donde dispone de un botón modificar o pulsando el botón lápiz del registro en el listado que le dará acceso al formulario.



Email	Nombre	Acciones
admin@localhost.com	admin	 
agapitodisousa6@gmail.com	Agapito Di Sousa	 
david@localhost.com	david	 
david1@localhost.com	david1	 
david3@localhost.com	david3	 
juan@localhost.com	juan	 
pepe@localhost.com	pepe	 
pepi@localhost.com	pepi	 
prueba@localhost.com	prueba	 
pruebawebAdmin@localhost.com	pruebawebAdmin	 



Nombre
pruebawebAdmin

Email
pruebawebAdmin@localhost.com

Password

Repite Password

Fecha alta
07-06-2024 16:20

Ultimo acceso
Ultimo acceso

Ultima modificacion
Ultima modificacion

Activo Administrador

Lista **Modificar** **Eliminar**

Dentro de la pantalla de modificación el usuario podrá digitar el nombre, cambiar la contraseña o modificar los permisos de la aplicación.
El campo email del usuario es único y no se puede modificar como tampoco los campos de fecha de alta, fecha de modificación y fecha de último acceso.

The screenshot shows a web browser window with the URL `localhost:8081/usuarios/update/pruebawebAdmin@localhost.com?`. The page title is "Prueba Web". The main content is a form for modifying a user. The fields are:

- Nombre: pruebawebAdmin modificado
- Email: pruebawebAdmin@localhost.com
- Password: (redacted)
- Repite Password: (redacted)
- Fecha alta: 07-06-2024 16:20
- Ultimo acceso: Último acceso
- Última modificación: Última modificación
- Activo
- Administrador

At the bottom are three buttons: "Cancelar" (Cancel), "Modificar" (Modify), and another "Modificar" button.

Como resultado de la operación, nos llevará a la consulta del usuario que hemos modificado.

The screenshot shows a web browser window with the URL `localhost:8081/usuarios/select/pruebawebAdmin@localhost.com`. The page title is "Prueba Web". The main content displays the user information:

Nombre: pruebawebAdmin modificado
Email: pruebawebAdmin@localhost.com
Password: (redacted)
Repite Password: (redacted)
Fecha alta: 07-06-2024 16:20
Ultimo acceso: Último acceso
Última modificación: 07-06-2024 16:23
 Activo
 Administrador

At the bottom are three buttons: "Lista" (List), "Modificar" (Modify), and "Eliminar" (Delete).

Cualquier usuario logado con el usuario al que se le modifique la contraseña o los permisos se le aplicará en el momento quedando inaccesible en función de las opciones modificadas.

Esta gestión es realizada por el controller de usuarios.

```
@GetMapping("/usuarios/update/{email}")
public String getUpdateUsuario(HttpServletRequest session, Model model, @PathVariable String email) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try{
        model.addAttribute("usuario", apiService.getUsuario(email));
        return "usuarios/update";
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/usuarios/view";
}

@PostMapping("/usuarios/update/{email}")
public String postUpdateUsuario(HttpServletRequest session, Model model, @RequestParam(value="nombre") String nombre, @PathVariable String email, @RequestParam(value="password") String password, @RequestParam(value="repeatPassword") String repeatPassword, @RequestParam(value="active") boolean active, @RequestParam(value="admin") boolean admin) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        Usuario usuario = updateUsuario(nombre, email, password, repeatPassword, active, admin);
        if (usuario != null) {
            return "redirect:/usuarios/select/" + usuario.getEmail();
        }
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    model.addAttribute("usuario", new Usuario(email , nombre, password, active, admin));
    return "usuarios/update";
}

private Usuario updateUsuario(String nombre, String email, String password, String repeatPassword, boolean active, boolean admin) throws Exception {
    if(validateUpdateUsuarioForm(nombre, email, password, repeatPassword)){
        Usuario usuario = apiService.getUsuario(email);

        //userModel=new User(email, name, password, isAdmin, isActive);
        usuario.setNombre(nombre);
        //se verifica si la contraseña ha cambiado, si no lo hizo no volver a lanzar el hash
        if(!password.equals(usuario.getPassword())){
            password = HashUtils.hashString(password);
        }
        usuario.setPassword(password);
        usuario.setActive(active);
        usuario.setAdmin(admin);

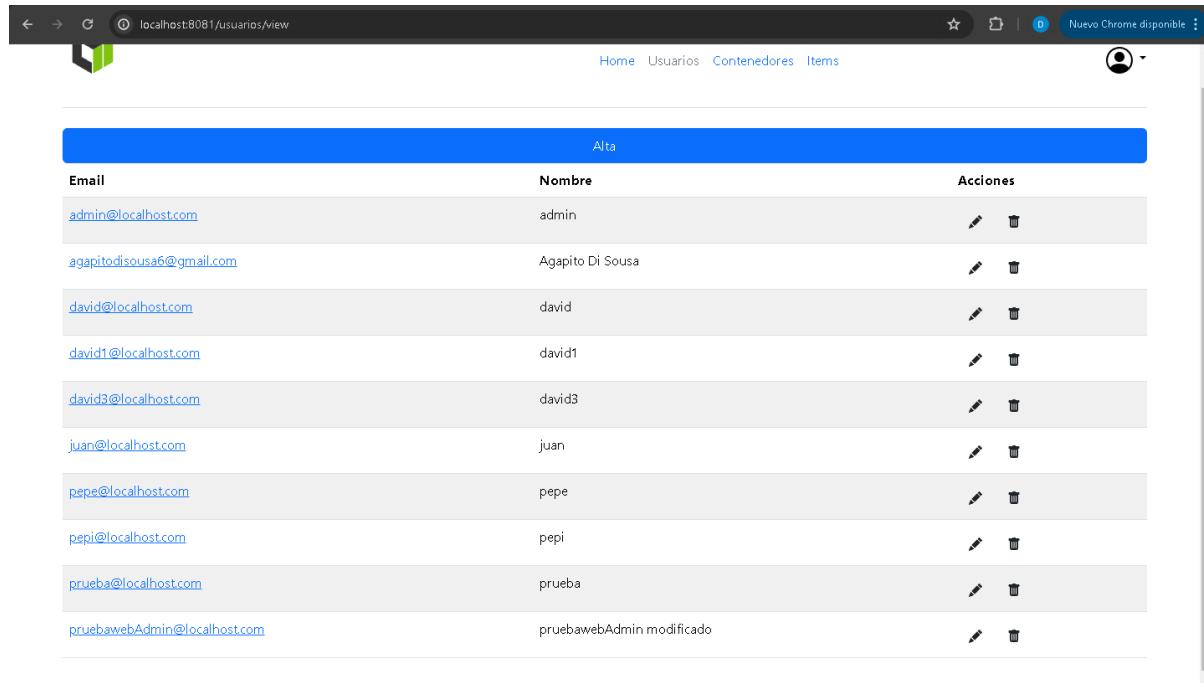
        return apiService.updateUsuario(email, usuario);
    }
    else {
        return null;
    }
}

private boolean validateUpdateUsuarioForm(String nombre, String email, String password, String repeatPassword) {
    boolean result = true;
    if(email.isEmpty()) {
        result = false;
    }
    if(nombre.isEmpty()) {
        result = false;
    }
    if(password.isEmpty()) {
        result = false;
    }
    if(repeatPassword.isEmpty()) {
        result = false;
    }
    if(result && !password.equals(repeatPassword)) {
        result = false;
    }
    return result;
}
```

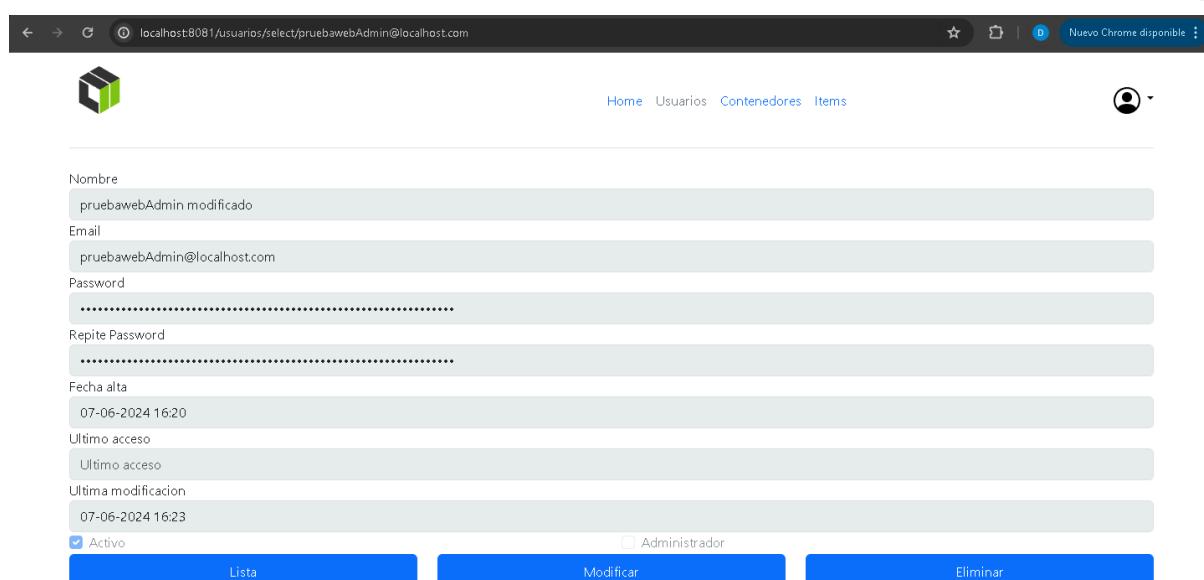
Borrado de usuario

Dentro de la aplicación el usuario podrá realizar el borrado de un usuario.

El acceso a la operativa de borrado del usuario se realizará desde la consulta del propio usuario, desde donde dispone de un botón borrar o pulsando el botón papelera del registro en el listado que le dará acceso a la acción directamente.



Email	Nombre	Acciones
admin@localhost.com	admin	 
agapitodisousa6@gmail.com	Agapito Di Sousa	 
david@localhost.com	david	 
david1@localhost.com	david1	 
david3@localhost.com	david3	 
juan@localhost.com	juan	 
pepe@localhost.com	pepe	 
pepi@localhost.com	pepi	 
prueba@localhost.com	prueba	 
pruebawebAdmin@localhost.com	pruebawebAdmin modificado	 



Nombre
pruebawebAdmin modificado

Email
pruebawebAdmin@localhost.com

Password
.....

Repite Password
.....

Fecha alta
07-06-2024 16:20

Ultimo acceso
Ultimo acceso

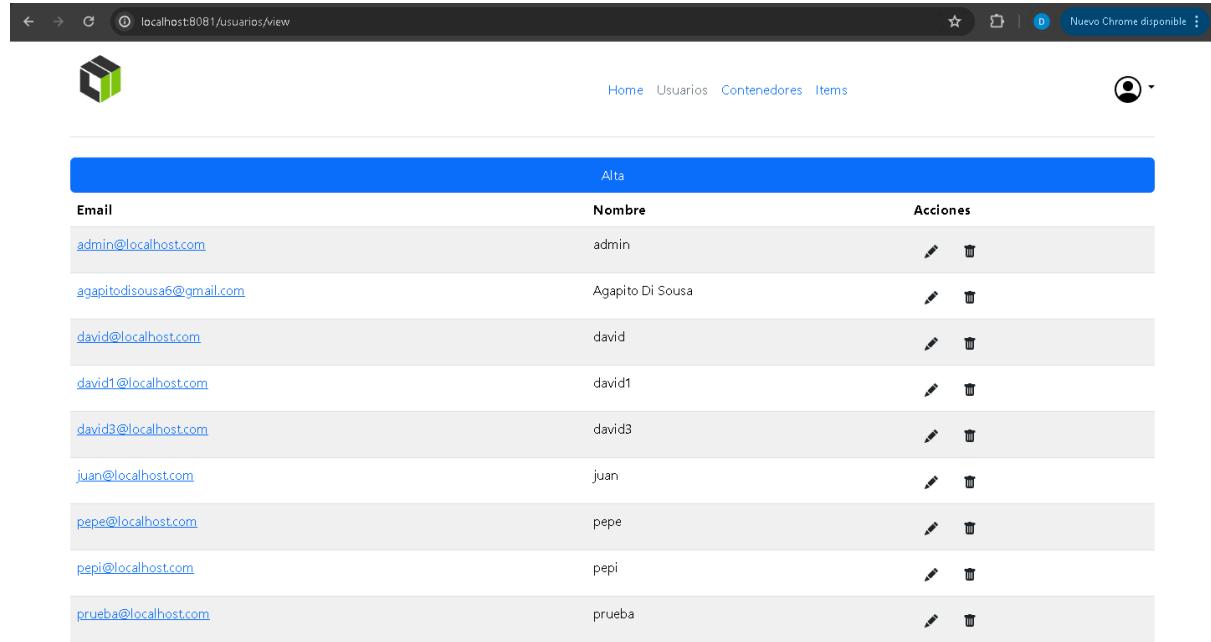
Ultima modificación
07-06-2024 16:23

Activo

Administrador

[Lista](#) [Modificar](#) [Eliminar](#)

Una vez ejecutado el borrado nos llevará a la pantalla de listado.



The screenshot shows a web application interface. At the top, there is a header bar with icons for back, forward, refresh, and search, followed by the URL 'localhost:8081/usuarios/view'. To the right of the URL are several browser-specific icons. Below the header is a navigation bar with links for 'Home', 'Usuarios', 'Contenedores', and 'Items'. On the far right of the navigation bar is a user profile icon. The main content area features a table titled 'Alta' (List). The table has three columns: 'Email', 'Nombre', and 'Acciones'. There are nine rows of data, each representing a user. The 'Acciones' column contains edit and delete icons for each row. The data is as follows:

Email	Nombre	Acciones
admin@localhost.com	admin	
agapitodisousa6@gmail.com	Agapito Di Sousa	
david@localhost.com	david	
david1@localhost.com	david1	
david3@localhost.com	david3	
juan@localhost.com	juan	
pepe@localhost.com	pepe	
pepi@localhost.com	pepi	
prueba@localhost.com	prueba	

Cualquier usuario logado con el usuario eliminado dejará de tener acceso a la aplicación. Esta gestión es realizada por el controller de usuarios.

```
@GetMapping("/usuarios/delete/{email}")
public String getDeleteUsuario(HttpSession session, Model model, @PathVariable String email) {
    if(!SecurityInterceptor.validateSession(session, model))return "redirect:/logout";
    try {
        apiService.deleteUsuario(email);
    }
    catch (HttpClientErrorException.Unauthorized e) {
        model.addAttribute("error", "error - " + e.getMessage());
        return "redirect:/logout";
    }
    catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", "error - " + e.getMessage());
    }
    return "redirect:/usuarios/view";
}
```

PROPUESTAS DE MEJORA

Como elementos a mejorar tendríamos varios:

Implementar un sistema de verificación de correos.

Implementar campos de auditoría en todas las acciones.

Implementar un sistema de perfilado.

Agregar un sistema de gestión de varios almacenes, esto para poder trabajar cada cliente en una sola aplicación.

Implementación de nuevos servidores y certificados para convertir el aplicativo en https

Mejorar el sistema de errores y logs.

Mejorar la robustez general.

CONCLUSIONES

Gracias a las tecnologías actuales y a los frameworks disponibles, el desarrollo de una aplicación se ha convertido en algo dinámico, ágil y escalable.

En conclusión, el desarrollo del proyecto de almacenaje ha sido un esfuerzo que ha abarcado diversas fases, desde la planificación y diseño hasta la implementación y evaluación.

El proyecto WareControl ha sido exitoso en su objetivo de proporcionar una herramienta útil y eficiente para la gestión de almacenamiento. Las tecnologías implementadas y las funcionalidades desarrolladas posicionan a WareControl como una solución práctica para la organización de espacios de almacenamiento en el ámbito doméstico..

Nosotros como desarrolladores tenemos claro que nuestra aspiración nos lleva a facilitar la vida cotidiana de las personas, sobre todo hoy en día que vivimos abrumados de interacciones.

BIBLIOGRAFÍA

- <https://spring.io/>
- <https://getbootstrap.com/>
- <https://www.thymeleaf.org/>
- <https://tomcat.apache.org/>
- <https://www.eclipse.org/>
- <https://developer.android.com/>
- <https://mariadb.org/>
- <https://www.mysql.com/>
- <https://www.java.com/es/>
- <https://github.com/>
- <https://www.postman.com/>