18-3-2018

Entregable 1

Sistemas distribuidos e internet

Iván González Mahagamage
UNIVERSISDAD DE OVIEDO

Autores: **Iván González Mahagamage**

Estudiante de Ingeniería Informática del Software en la Universidad de

Fecha: 18 de marzo de 2018

Versión: V1

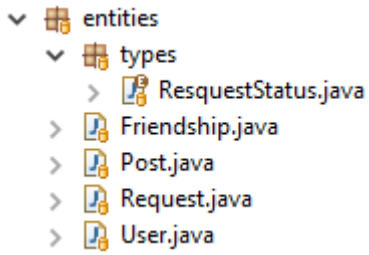
Contenido

Implementación

0. Aspectos comunes

0.1. Entidades de la aplicación.

Para la realización de la aplicación se ha diseñado una serie clases de objeto dentro del paquete “entities”.



Todos ellas están configuradas con la tecnología JPA para manejar la base de datos de manera automática. Las clases creadas son:

- User: simula los usuarios que están dentro de la aplicación.

```
@Entity
@Table(name = "TUSERS")
public class User {
    @Id
    @GeneratedValue
    private long id;

    @Column(unique = true)
    @NotNull
    private String email;

    private String name;

    @Column(name = "surname")
    private String surName;

    @NotNull
    private String password;

    @Transient
    private String passwordConfirm;

    private String role;

    @OneToMany(mappedBy = "sender", cascade = CascadeType.ALL)
    private Set<Request> sentRequests = new HashSet<>();

    @OneToMany(mappedBy = "receiver", cascade = CascadeType.ALL)
    private Set<Request> receiveRequests = new HashSet<>();

    @OneToMany(mappedBy = "friend")
    private Set<Friendship> friends = new HashSet<>();

    @OneToMany(mappedBy = "user")
    private Set<Friendship> iAmFriendOf = new HashSet<>();

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private Set<Post> posts = new HashSet<>();
```

La información del usuario que se guarda es su email, es único y se usara para su identificación, nombre completo y contraseña. Además de las referencias a los otros objetos de la aplicación.

- Request: simula las peticiones de amistad que se envían entre sí.

```
@Entity
@Table(name = "TREQUEST")
public class Request {
    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    private User sender;

    @ManyToOne
    private User receiver;

    @Enumerated(EnumType.STRING)
    private ResquestStatus status;
```

En este objeto además de guardar una referencia al usuario que manda la petición y el que la recibe, se guarda el estado de esta, que se indicara con enumerable. Este puede ser enviada, aceptada o bloqueada.

```
public enum ResquestStatus {
    SENT, ACCEPTED, BLOCKED
}
```

- Friendship: simula la amistad que existe entre los diferentes usuarios.

```
@Entity
@Table(name = "TFRIENDSHIP")
public class Friendship {

    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    private User user;

    @ManyToOne
    private User friend;
```

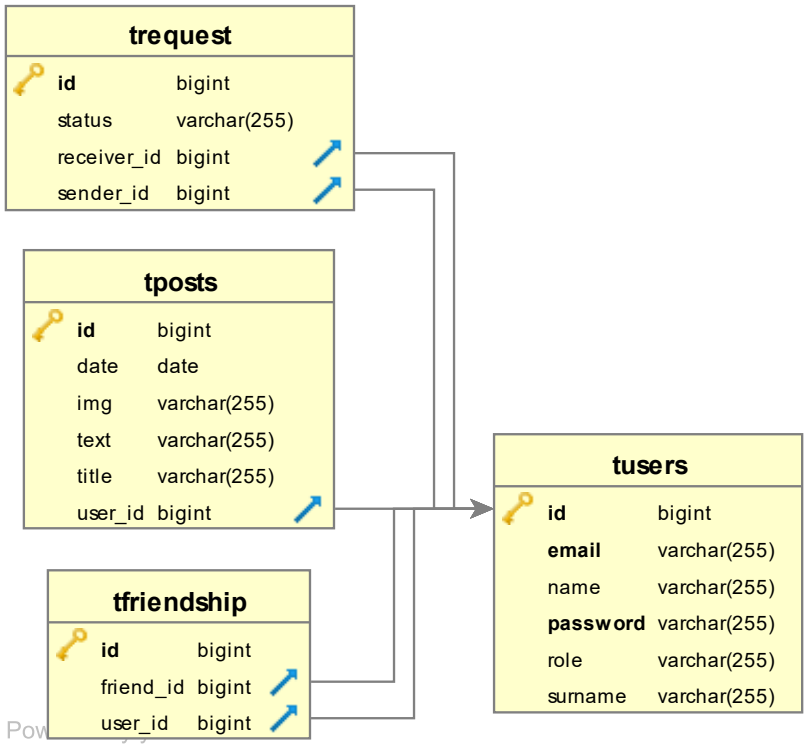
Esta clase ha sido creada debido a los problemas entre el mapeador y la paginación de lista, que se explicará más adelante en el documento. La manera elegante de crear las amistades hubiera sido en el atributo “friends” de la clase “User” haber puesto la etiqueta “ManyToMany” pero a la hora de hacer la paginación de resultados la aplicación fallaba, cuando había más de una página. Por este motivo se optó por crear esta clase

- Post: publicaciones que realizan los usuarios.

```
@Entity
@Table(name = "TPOSTS")
public class Post {
    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    private User user;

    private String title, text, img;

    private Date date;
```

En esta clase se guarda una referencia al usuario que ha creado la publicación, el título y el contenido se guardan directamente, pero se guarda la referencia a la imagen, no la misma en sí, pero esto ya se detallara más adelante.

La base de datos resultante es la siguiente:



También hay que decir que cada una de estas clases tiene un controlador, un servicio y repositorio.

0.2. Creación automática de datos.

Para tener datos para probar la aplicación se ha creado la clase “InsertSampleDataService” que tiene tres funciones:

- Crea un usuario Admin: ivangonzalezmahagamage@gmail.com.
- Crea un usuario User: igm1990@hotmail.com
- Genera el número de usuarios aleatorios que le indicamos.

A continuación, añado un fragmento de esta clase en la que se muestra como realiza estas opciones en el “@PostContruct”

```
@PostConstruct
public void init() {
    // inicializar(50);
}

protected void inicializar(int limite) {
    user1 = new User("ivangonzalezmahagamage@gmail.com", "Iván",
            "González Mahagamage");
    user1.setPassword("123456");
    user1.setRole(rolesService.getAdmin());
    Post post = new Post(user1, "Prueba1", "Contenido", "");
    user1.getPosts().add(post);
    usersService.add(user1);

    User user2 = new User("igm1990@hotmail.com", "Iván",
            "González Mahagamage");
    user2.setPassword("123456");
    user2.setRole(rolesService.getUser());
    usersService.add(user2);
    rellenarBaseDatos(limite);

}

protected void rellenarBaseDatos(int limite) {
    while (users.size() < limite) {
        createUser();
    }

    Iterator<User> a = users.iterator();
    while (a.hasNext()) {
        usersService.add(a.next());
    }
}
```

0.3. Creación del log de actividad.

Como se indica en el enunciado, se creado una clase denominada “LogService” para crear un log de todos los sucesos que ocurren en la aplicación. Para ello se añadido la siguiente dependencia al archivo “pom.xml”.

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
</dependency>
```

También en el fichero “aplicaction.properties” indicamos el fichero donde se guardará el log.

```
server.port=8090
spring.datasource.url=jdbc:hsqldb:hsql://localhost:9001
spring.datasource.username=SA
spring.datasource.password=
spring.datasource.driver-class-name=org.hsqldb.jdbcDriver
spring.jpa.hibernate.ddl-auto=validate
spring.messages.basename=messages/messages
logging.level.org.springframework.web=ERROR
logging.level.org.hibernate=ERROR
logging.file=log/log.log
spring.http.multipart.max-file-size=30MB
spring.http.multipart.max-request-size=30MB
```

La implementación de esta clase es la siguiente:

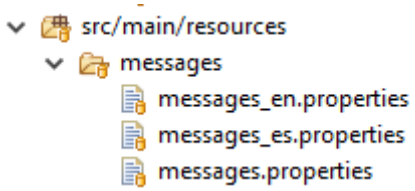
```
package com.uniovi.services;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.uniovi.services.util.CreateFolder;

public class LogService {

    private Logger log;

    public LogService(Object object) {
        CreateFolder.createFolder("/log");
        log = LoggerFactory.getLogger(object.getClass());
    }

    public void info(String message) {
        log.info(message);
    }

    public void error(String message) {
        log.error(message);
    }

    public void debug(String message) {
        log.debug(message);
    }

}
```

Se optado por esta manera para hacerla dinámica, únicamente tenemos que instanciarla en las clases que queremos registrar su actividad pasándole como parámetro “this” en su constructor. Un ejemplo:

```java
@Controller
public class HomeController {

    private LogService logService = new LogService(this);

    @RequestMapping("/")
    public String index() {
        logService.info("Usuario a entrado en la aplicación");
        return "index";
    }

}
```

0.4. Proceso de internacionalización.

Para el proceso de internacionalización se han creado los ficheros .messages como se indicó en clase únicamente con una ligera variación, su localización. En mi caso, los he colocado dentro de un directorio denominado “messages” dentro del directorio “resources”

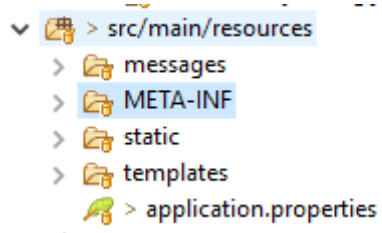


Para que la aplicación los reconozca, he tenido que hacer la siguiente modificación en el archivo “aplicaction.properties”.

```
server.port=8090
spring.datasource.url=jdbc:hsqldb:hsql://localhost:9001
spring.datasource.username=SA
spring.datasource.password=
spring.datasource.driver-class-name=org.hsqldb.jdbcDriver
spring.jpa.hibernate.ddl-auto=validate
spring.messages.basename=messages/messages
logging.level.org.springframework.web=ERROR
logging.level.org.hibernate=ERROR
logging.file=log/log.log
spring.http.multipart.max-file-size=30MB
spring.http.multipart.max-request-size=30MB
```

0.5. Externacionalización de las consultas.

Este apartado no se ha dado en clase pero yo he decido externalizar las consultas tal como nos enseñaron en la asignatura de “Repositorios de la Información”. Para ello he creado un directorio denominado “META-INF” dentro del directorio “resource”.



Dentro de este he creado un fichero denominado “jpa-named-queries.properties”, en el cual he añadido las diferente consultas que se usan en la aplicación. Para que esta funcione correctamente, se debe seguir la siguiente estructura:

- Nombre de la clase que devuelve la consulta.
- Nombre del método dentro del repositorio que referencia la consulta.
- Símbolo “=”
- Consulta

```
User.searchByEmailAndNameAndSurname=SELECT u FROM User u WHERE (LOWER(u.email) LIKE LOWER(?1) OR LOWER(u.name) L
User.findAllList=SELECT u FROM User u WHERE u.id != ?1 ORDER BY u.surName ASC
User.findAllByRequestReceiverId=SELECT r.sender FROM Request r WHERE r.status = 'SENT' AND r.receiver.id = ?1
User.findAllFriendsById=SELECT u.friend FROM Friendship u WHERE u.user.id = ?1
Request.findAllSentById=SELECT r FROM Request r WHERE r.sender.id = ?1
Request.findBySenderIdAndReceiverId=SELECT r FROM Request r WHERE r.sender.id = ?1 AND r.receiver.id = ?2
Post.findAll=SELECT p FROM Post where p.user.id = ?1
Friendship.searchByUsers=SELECT f FROM Friendship f WHERE f.user = ?1 AND f.friend = ?2
```

0.6. Roles de la aplicación

He establecido dos roles para la aplicación “ROLE_USER” para usuario sin privilegios especiales y “ROLE_ADMIN” para usuario con privilegios de administrador. Esto se realizado mediante la creación de la clase “RolesService”.

```java
@Service
public class RolesService {

    public String getUser() {
        return "ROLE_PUBLIC";
    }

    public String getAdmin() {
        return "ROLE_ADMIN";
    }
}
```

0.7. Definición de permisos

Para esta aplicación hemos definidos los siguientes permisos en la clase “WebSecurityConfig”.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable().authorizeRequests() // peticiones autorizadas
                .antMatchers("/css/**", "/img/**", "/script/**", "/",
                        "/signup", "/login", "/admin/login")
                .permitAll()
                // Permite a todos los usuarios
                .antMatchers("/admin/**").hasAuthority("ROLE_ADMIN")
                .anyRequest().authenticated()
                // pagina de autentificacion por defecto
                .and().formLogin().loginPage("/login").permitAll()
                // Si se loguea bien
                .defaultSuccessUrl("/home").and().logout().permitAll();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth)
            throws Exception {
        auth.userDetailsService(userDetailsService)
                .passwordEncoder(bCryptPasswordEncoder());
    }
}
```

Definimos que los css, imágenes, scripts, las pagina de inicio, registrase, login, y login del administrador pueda acceder cualquier usuario. También definimos que las url que sean de administrador solo pueden acceder los usuarios que tengan el rol “ADMIN_USER”

0.8. Utilidades

He reutilizado una clase denominada “createFolder” para crear directorios de manera dinámica para el log de la aplicación y el guardado de las imágenes subidas por los usuarios.

```java
/**
 * Clase encargada de crear las carpetas necesarias para nuestra aplicación.
 *
 * @author Adrián García Lumbreras
 * @author Iván González Mahagamage
 *
 */
public class CreateFolder {
    /**
     * Crea una carpeta por código si esta no existe previamente.
     *
     * @param url
     */
    public static void createFolder(String url) {
        File file = new File(url);
        if (!file.exists()) {
            file.mkdir();
        }
    }
}
```

1. Público: registrarse como usuario

Primero se ha creado una vista para este propósito denominada “signup.html”. De esta vista solo voy destacar ciertos aspectos ya que es demasiado código para ponerlo en este documento.

- Al formulario le insertamos un object para que, si el registro no es válido, guarde los datos que ha introducido el usuario a excepción de las contraseñas para evitar que los tenga que volver a introducir.

```html
<form method="post" action="/signup" th:object="${user}">
```

- A parte de introducir la internacionalización en todas las vistas, he creado un archivo en JavaScript para incluir validaciones por el lado del cliente además de incluir el atributo placeholder.

```html
<div class="form-group">
    <label th:text="#{signup.name}">Nombre:</label>
    <input class="form-control" type="text" name="name" placeholder="Iván" required="required"
           onkeyup="check.checkInputs()" th:attr="title=#{signup.name.title}, value=${user.name}"
           pattern="[a-zA-ZñÑáéíóúÁÉÍÓÚ]{2,}" />
    <div th:if="${#fields.hasErrors('name')}" class="alert alert-dismissible alert-danger">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong th:errors="*{name}">Oh snap!</strong>
    </div>
</div>
```

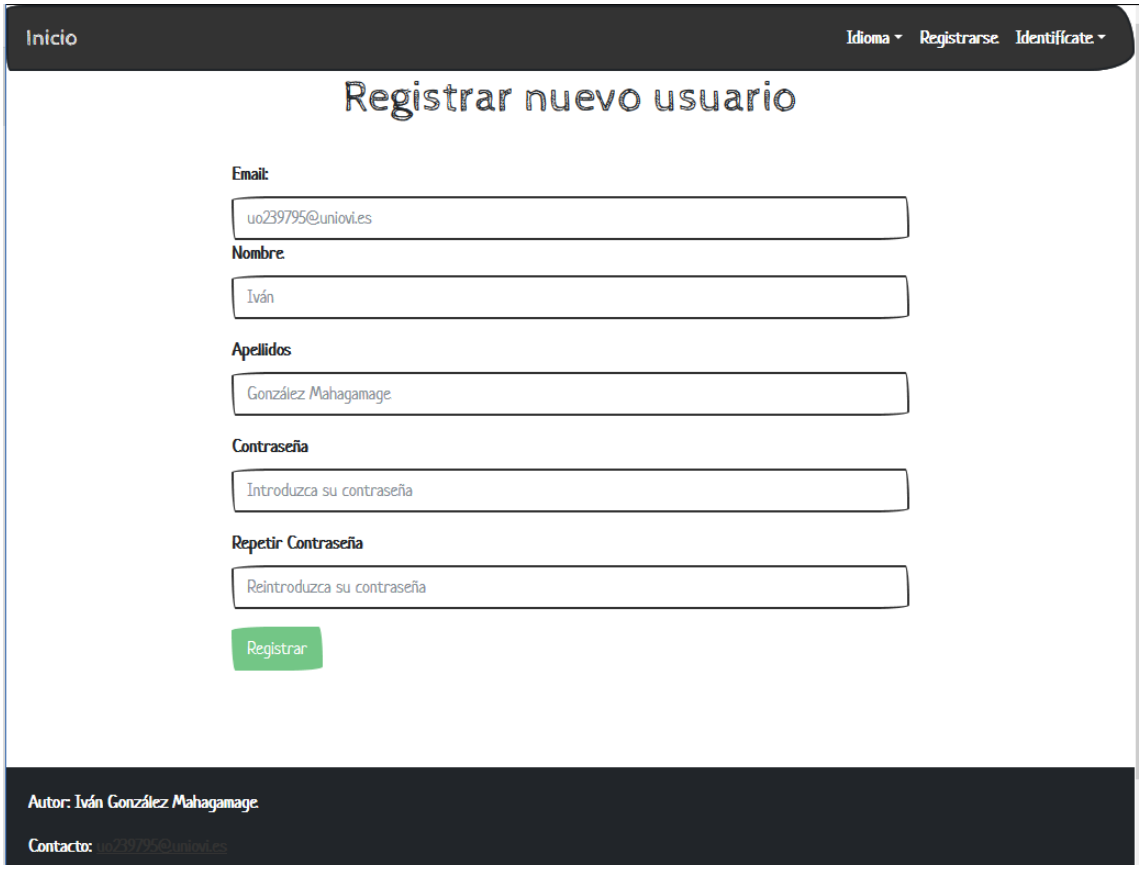
- El archivo JavaScript denominado anterior es “checkForm.js”.”

```html
<body>
    <script src="/script/checkForm.js"></script>
```

Comprueba que las dos contraseñas que introduce el usuario son iguales y hasta que el usuario no introduce todos los campos del formulario desactiva el botón de “Enviar”.

```
"use strict";

class Check {
    constructor() {}

     checkPasswords() {
        var password = $('input[name=password]').val();
        var repassword = $('input[name=passwordConfirm]').val();
        if (password !== repassword) {
            alert("Error: las contraseñas no coinciden");
            return false;
        }
        return true;
    }

    checkInputs() {
        $('input[type=submit]').prop('disabled', false);
        var inputs = $('input');
        for (var i = 0; i < inputs.length; i++) {
            var value = inputs[i].value;
            if (inputs[i].value === "") {
                $('input[type=submit]').prop('disabled', true);
                break;
            }
        }
    }
}

var check = new Check();
```

El resultado final de estos archivos es:



Para que la vista se muestre correctamente debemos incluir en “UserController” los siguientes métodos:

- Un método para recoja la petición Get para mostrar la vista.

```
@GetMapping("/signup")
public String signUp(Model model) {
    logService.info("Usuario se intenta registrar");
    model.addAttribute("user", new User());
    return "signup";
}
```

- Un método para recoja la petición Post y procese el formulario. Al hacer el autologin usamos el parámetro “passwordConfirm” ya que este método necesita la contraseña sin codificar para funcionar correctamente.

```
@PostMapping("/signup")
public String signupPost(@Validated User user, BindingResult result,
        Model model) {
    signUpFormValidator.validate(user, result);
    if (result.hasErrors()) {
        logService.error("Usuario introdujo mal los datos");
        return "signup";
    }
    logService.info("Usuario se ha registrado correctamente como "
            + user.getEmail());
    user.setRole(rolesService.getUser());
    usersService.add(user);
    securityService.autoLogin(user.getEmail(), user.getPasswordConfirm());
    return "redirect:home";
}
```

En este método incluimos una llamada a un validador para comprobarlo en lado del servidor.

```
@Component
public class SignUpFormValidator implements Validator {
    @Autowired
    private UsersService usersService;

    @Override
    public boolean supports(Class<?> aClass) {
        return User.class.equals(aClass);
    }
```

```
    @Override
    public void validate(Object target, Errors errors) {
        User user = (User) target;
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email",
                "Error.empty");
        if (usersService.getUserByEmail(user.getEmail()) != null) {
            errors.rejectValue("email", "Error.signup.email.duplicate");
        }
        if (user.getName().length() < 2) {
            errors.rejectValue("name", "Error.signup.name.length");
        }
        if (user.getSurName().length() < 2) {
            errors.rejectValue("surName", "Error.signup.lastName.length");
        }
        if (user.getPassword().length() < 5) {
            errors.rejectValue("password", "Error.signup.password.length");
        }
        if (user.getPasswordConfirm().length() < 5) {
            errors.rejectValue("passwordConfirm",
                    "Error.signup.password.length");
        }
        if (!user.getPasswordConfirm().equals(user.getPassword())) {
            errors.rejectValue("passwordConfirm",
                    "Error.signup.passwordConfirm.coincidence");
        }
    }
}
```

Si supera las comprobaciones se le asigna el rol “ROLE_USER” y se llama a “UserService” para que lo añada a la base de datos y encripte su contraseña.

```
public void add(User user) {
    user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
    usersRepository.save(user);
}
```

Una vez finalizado este proceso, la aplicación loguea automáticamente al usuario para que comience a usar la aplicación. Para ello también debemos implementar la clase “SecurityService”.

```java
@Service
public class SecurityService {
    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserDetailsService userDetailsService;
    private static final Logger logger = LoggerFactory
            .getLogger(SecurityService.class);

    public String findLoggedInEmail() {
        Object userDetails = SecurityContextHolder.getContext()
                .getAuthentication().getDetails();
        if (userDetails instanceof UserDetails) {
            return ((UserDetails) userDetails).getUsername();
        }
        return null;
    }

    public void autoLogin(String email, String password) {
        UserDetails userDetails = userDetailsService.loadUserByUsername(email);
        UsernamePasswordAuthenticationToken aToken;
        aToken = new UsernamePasswordAuthenticationToken(userDetails, password,
                userDetails.getAuthorities());
        authenticationManager.authenticate(aToken);
        if (aToken.isAuthenticated()) {
            SecurityContextHolder.getContext().setAuthentication(aToken);
            logger.debug(String.format("Auto login %s successfully!", email));
        }
    }
}
```

A su vez, esta clase llama a “UserDetailsServiceImpl”.

```java
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private UsersRepository usersRepository;

    @Override
    public UserDetails loadUserByUsername(String email)
            throws UsernameNotFoundException {
        com.uniovi.entities.User user = usersRepository.findByEmail(email);
        Set<GrantedAuthority> grantedAuthorities = new HashSet<>();
        grantedAuthorities.add(new SimpleGrantedAuthority(user.getRole()));
        return new User(user.getEmail(), user.getPassword(),
                grantedAuthorities);
    }

}
```

2. Público: iniciar sesión

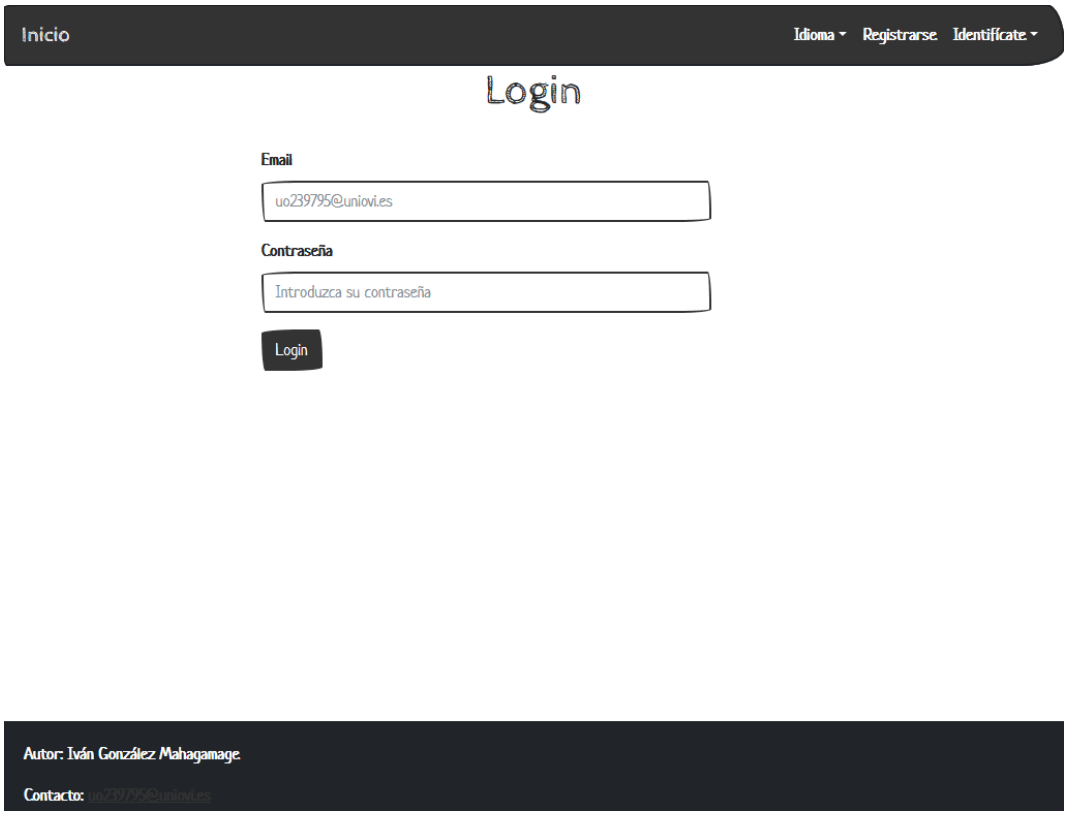
Se ha implementado la vista “login.html” para que el usuario pueda iniciar sesión.

Hay destacar un punto fundamental para que funcione correctamente esta vista.

- Los inputs que se usan deben tener como atributo “name”, “username” y “password” respectivamente para que los reconozca Spring Security sin problemas.

```
<!DOCTYPE html>
<html lang="es">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container">
        <h1 th:text="#{login.title}">Idéntificate</h1>
        <form class="form-horizontal" method="post" action="/login">
            <fieldset>
                <div class="form-group">
                    <label class="control-label col-sm-2" for="username">Email</label>
                    <div class="col-sm-10">
                        <input type="email" class="form-control" name="username"
                               placeholder="uo239795@uniovi.es" required="required" th:attr="title=#{signup.email.title}" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-sm-2" for="password" th:text="#{signup.password}">Password:</label>
                    <div class="col-sm-10">
                        <input type="password" class="form-control" name="password"
                               th:attr="placeholder=#{signup.password.placeholder}, title=#{signup.password.placeholder}"
                               pattern=".{2,}" required="required" />
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-sm-offset-2 col-sm-10">
                        <button type="submit" class="btn btn-primary">Login</button>
                    </div>
                </div>
                <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
            </fieldset>
            <div th:if="${param.error}" class="alert alert-dismissible alert-danger">
                <button type="button" class="close" data-dismiss="alert">&times;</button>
                <strong th:text="#{login.error}">Oh snap!</strong>
            </div>
            <div th:if="${param.logout}" class="alert alert-dismissible alert-success">
                <button type="button" class="close" data-dismiss="alert">&times;</button>
                <strong th:text="#{login.logout}">Oh snap!</strong>
            </div>
        </form>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

El resultado es el siguiente:



Se declarado un nuevo método “login()” en “UserController” para mostrar esta vista.

```
@GetMapping("/login")
public String login() {
    logService.info("Usuario se intenta loggear");
    return "login";
}
```

Cabe destacar que como se usa Spring Security no hay que implementar la petición post de la vista.

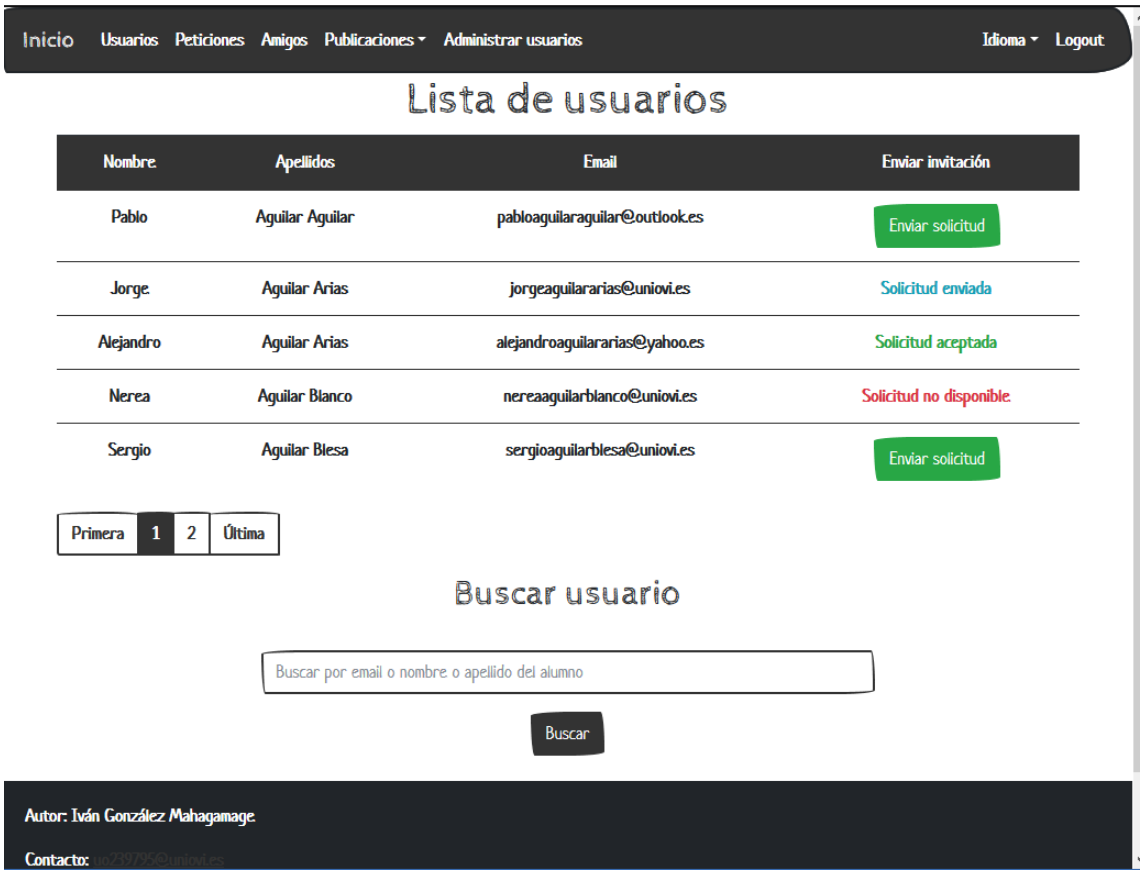
3. Usuario registrado: listar todos los usuarios de la aplicación

Se ha creado la vista “list.html” dentro del directorio “users”.

```
<!DOCTYPE html>
<html lang="es">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container" style="text-align: center">
        <h1 th:text="#{list.title}">Esta es una zona privada la web</h1>
        <table class="table table-hover">
            <thead class="table-dark">
                <tr>
                    <th scope="col" th:text="#{signup.name}">Column heading</th>
                    <th scope="col" th:text="#{signup.surname}">Column heading</th>
                    <th scope="col">Email</th>
                    <th scope="col" th:text="#{list.add}"></th>
                </tr>
            </thead>
            <tbody>
                <tr class="table-light" th:each="user:${usersList}">
                    <td th:text="${user.name}">Column content</td>
                    <td th:text="${user.surName}">Column content</td>
                    <td th:text="${user.email}">Column content</td>
                    <td>
                        <form th:if="${user.receiveRequest} == null" th:attr="action=${'/request/send/' + user.id}">
                            <input type="submit" class="btn btn-success" th:attr="value=#{list.send}" />
                        </form>
                        <span th:if="${user.receiveRequest} != null and ${#strings.toString(user.receiveRequest.status)} == 'ACCEPTED'"
                              class="text-success" th:text="#{list.send.accepted}"></span>
                        <span th:if="${user.receiveRequest} != null and ${#strings.toString(user.receiveRequest.status)} == 'SENT'"
                              type="button" class="text-info" th:text="#{list.send.info}" disabled="disable">Success</span>
                        <span th:if="${user.receiveRequest} != null and ${#strings.toString(user.receiveRequest.status)} == 'BLOCKED'"
                              type="button" class="text-danger" th:text="#{list.send.blocked}" disabled="disable">Success</span>
                    </td>
                </tr>
            </tbody>
        </table>
        <div th:replace="fragments/pagination"></div>
        <h2 th:text="#{list.search.title}">Buscar usuario</h2>
        <form class="navbar-form" action="/user/list">
            <div class="form-group">
                <input name="searchText" type="text" class="form-control" size="50" th:attr="placeholder=#{list.search}" />
            </div>
            <button type="submit" class="btn btn-primary" th:text="#{list.search.button}">Buscar</button>
        </form>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

En esta vista hay elementos pertenecientes a otros casos de usos que se explicarán más adelante.

El resultado es el siguiente:



Se declarado un nuevo método "`listUsers()`" en "UserController" para mostrar esta vista.

```
@GetMapping("/user/list")
public String listUsers(Model model, Pageable pageable,
        @RequestParam(value = "", required = false) String searchText,
        Principal principal) {
    logService.info(principal.getName() + " lista los usuarios");
    User user = usersService.getUserByEmail(principal.getName());
    Page<User> page = getUsers(pageable, searchText, user);
    List<User> list = page.getContent();
    for (User u : list) {
        u.setReceiveRequest(requestsService
                .findBySenderIdAndReceiverId(user.getId(), u.getId()));
    }
    model.addAttribute("usersList", list);
    model.addAttribute("page", page);
    return "/users/list";
}
```

El bucle for que hay en el método se explicara en el caso de uso 5.

También se ha creado el método "getUsers()" en esta clase para diferenciar cuando se muestran todos los usuarios o se desea filtrar, en este caso de uso e ejecutaría la opción del else.

```
private Page<User> getUsers(Pageable pageable, String searchText,
        User user) {
    Page<User> users = new PageImpl<User>(new LinkedList<User>());
    if (searchText != null && !searchText.isEmpty()) {
        users = usersService.searchByEmailAndNameAndSurname(pageable,
                searchText, user.getId());
    } else {
        users = usersService.getUsers(pageable, user.getId());
    }
    return users;
}
```

Para realizar estas búsquedas se han implementado los siguientes métodos en “UserServicies”

```
public User getUserByEmail(String email) {
    return usersRepository.findByEmail(email);
}

public Page<User> getUsers(Pageable pageable, Long id) {
    return usersRepository.findAllList(pageable, id);
}
```

Y en la clase “UsersRepository” también se han añadido nuevos métodos.

```
User findByEmail(String email);

Page<User> findAllList(Pageable pageable, Long id);
```

Cuyas consultas son:

- User.findAllList=SELECT u FROM User u WHERE u.id != ?1 ORDER BY u.surName ASC

4. Usuario registrado: buscar entre todos los usuarios de la aplicación

Para hacer este caso de uso se ha utilizado la vista “list.html” del directorio “users”, para ser más específicos el siguiente fragmento.

```
<h2 th:text="#{list.search.title}">Buscar usuario</h2>
<form class="navbar-form" action="/user/list">
    <div class="form-group">
        <input name="searchText" type="text" class="form-control" size="50" th:attr="placeholder=#{list.search}" />
    </div>
    <button type="submit" class="btn btn-primary" th:text="#{list.search.button}">Buscar</button>
</form>
```

Cuando se ejecuta este formulario llama al método “`listUsers()`” en “UserController” pero indicando el parámetro “searchText” para realizar el filtro en el método “getUsers()”.

En “UsersService” se ha implementado un nuevo método “searchByEmailAndNameAndSurname” para realizar la nueva búsqueda.

```
public Page<User> searchByEmailAndNameAndSurname(Pageable pageable,
        String searchText, Long id) {
    searchText = "%" + searchText + "%";
    return usersRepository.searchByEmailAndNameAndSurname(pageable,
            searchText, id);
}
```

Y en “UsersRepository” también se ha añadido un nuevo método “searchByEmailAndNameAndSurname”.

```
Page<User> searchByEmailAndNameAndSurname(Pageable pageable,
String seachtext, Long id);
```

Cuya consulta es:

- User.searchByEmailAndNameAndSurname=SELECT u FROM User u WHERE (LOWER(u.email) LIKE LOWER(?1) OR LOWER(u.name) LIKE LOWER(?1) OR LOWER(u.surName) LIKE LOWER(?1)) AND u.id != ?2 ORDER BY u.surName ASC

5. Usuario registrado: enviar una invitación de amistad a un usuario

Para hacer este caso de uso se ha utilizado la vista “list.html” del directorio “users”, para ser más específicos el siguiente fragmento.

```
<td>
    <form th:if="${user.receiveRequest} == null" th:attr="action=${'/request/send/' + user.id}">
        <input type="submit" class="btn btn-success" th:attr="value=#{list.send}" />
    </form>
    <span th:if="${user.receiveRequest} != null and ${#strings.toString(user.receiveRequest.status)} == 'ACCEPTED'"
          class="text-success" th:text="#{list.send.accepted}">Duis mollis, est non commodo luctus, nisi erat porttitor ligula.</span>
    <span th:if="${user.receiveRequest} != null and ${#strings.toString(user.receiveRequest.status)} == 'SENT'"
          type="button" class="text-info" th:text="#{list.send.info}" disabled="disable">Success</span>
    <span th:if="${user.receiveRequest} != null and ${#strings.toString(user.receiveRequest.status)} == 'BLOCKED'"
          type="button" class="text-danger" th:text="#{list.send.blocked}" disabled="disable">Success</span>
</td>
```

Como en thymleaf es difícil iterar en una lista de objetos contenida dentro de un objeto, he decidido hacer la comprobación en java. Para ello he añadido un atributo transient a los usuarios denominado “receiveRequest”.

```
// Esta variable se usa para mostrar el estado de la solicitud con el
// usuario logeado a la hora de listar los usuarios en la aplicación
@Transient
private Request receiveRequest;
```

En este objeto se carga la petición de amistad entre el usuario que está en sesión y el este usuario, si existe. Así en thymleaf solo cambiamos lo que se muestra dependiendo de este atributo.

Pablo	Aguilar Aguilar	pabloaguilaraguilar@outlook.es	Enviar solicitud
Jorge	Aguilar Arias	jorgeaguilararias@uniovi.es	Solicitud enviada
Alejandro	Aguilar Arias	alejandroaguilararias@yahoo.es	Solicitud aceptada
Nerea	Aguilar Blanco	nereaaguilarblanco@uniovi.es	Solicitud no disponible

Como este atributo es null cuando se crean los objetos, a la hora de llamar al controlador se inicializa.

```
@GetMapping("/user/list")
public String listUsers(Model model, Pageable pageable,
        @RequestParam(value = "", required = false) String searchText,
        Principal principal) {
    logService.info(principal.getName() + " lista los usuarios");
    User user = usersService.getUserByEmail(principal.getName());
    Page<User> page = getUsers(pageable, searchText, user);
    List<User> list = page.getContent();
    for (User u : list) {
        u.setReceiveRequest(requestsService
                .findBySenderIdAndReceiverId(user.getId(), u.getId()));
    }
    model.addAttribute("usersList", list);
    model.addAttribute("page", page);
    return "/users/list";
}
```

También hay que modificar “UsersServices” para añadir un nuevo método que devuleva las peticiones en “RequestServices”.

```
public Request findBySenderIdAndReceiverId(Long sender, Long receiver) {
    return requestRepository.findBySenderIdAndReceiverId(sender, receiver);
}
```

Y por consecuencia también hay que modificar “RequestRepository”.

```
Request findBySenderIdAndReceiverId(Long sender, Long receiver);
```

Cuya consulta es:

- Request.findBySenderIdAndReceiverId=SELECT r FROM Request r WHERE r.sender.id = ?1 AND r.receiver.id = ?2

Una vez configurada la lista, cuando el usuario envia el formulario para enviar una petición, es decir pulsa el botón “Enviar solicitud” se llama a “RequestController” con el siguiente método.

```
@GetMapping("/request/send/{id}")
public String sendRequest(@PathVariable Long id, Principal principal) {
    User user = usersService.getUserByEmail(principal.getName());
    logService.info(principal.getName() + " envia una petición al usuario"
            + user.getEmail());
    requestsService.add(new Request(user, usersService.getUser(id)));
    return "redirect:/user/list";
}
```

Y a su vez a “RequestService” con el siguiente método.

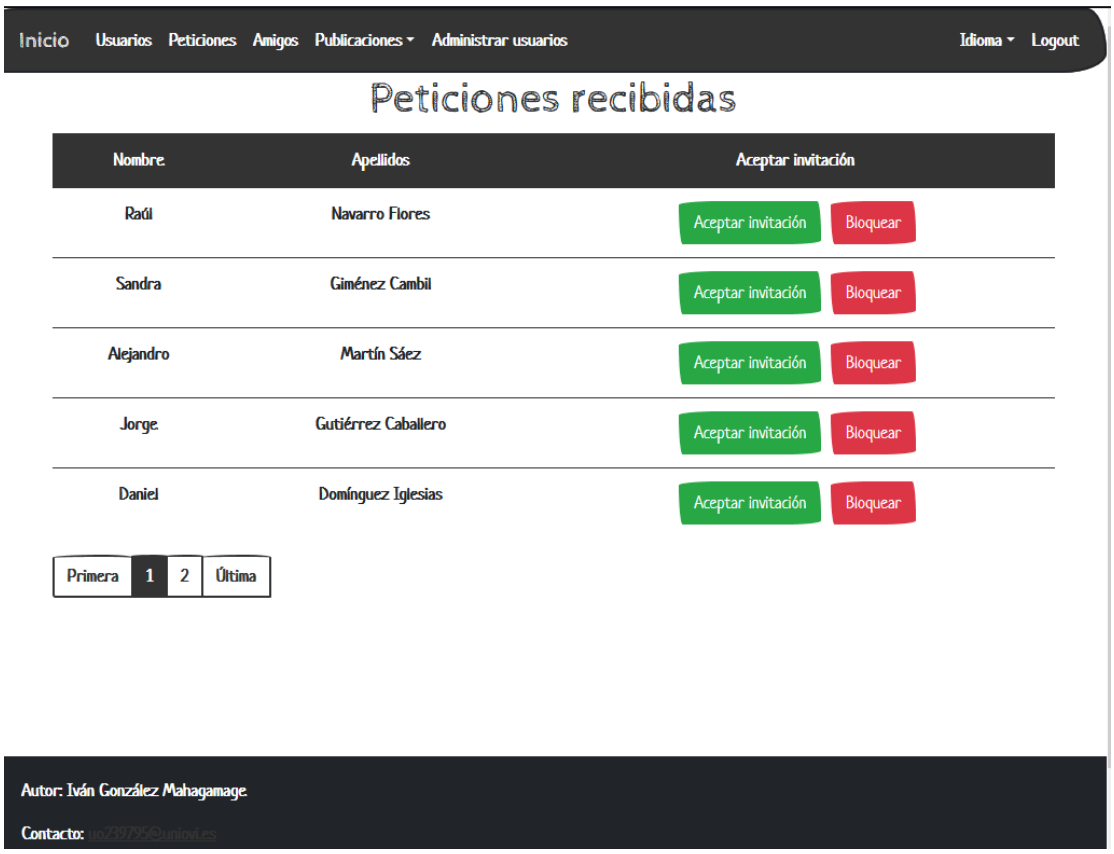
```
public void add(Request request) {
    requestRepository.save(request);
}
```

6. Usuario registrado: listar las invitaciones de amistad recibidas

Para listar todas las invitaciones se ha creado una nueva vista denominada “receiver.html” dentro del directorio “requests”.

```
<!DOCTYPE html>
<html lang="es">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container" style="text-align: center">
    <h1 th:text="#{nav.receive}">Esta es una zona privada la web</h1>
        <table class="table table-hover">
            <thead class="table-dark">
                <tr>
                    <th scope="col" th:text="#{signup.name}">Column heading</th>
                    <th scope="col" th:text="#{signup.surname}">Column heading</th>
                    <th scope="col" th:text="#{receiver.button}"></th>
                </tr>
            </thead>
            <tbody>
                <tr class="table-light" th:each="user:${usersList}">
                    <td th:text="${user.name}">Column content</td>
                    <td th:text="${user.surName}">Column content</td>
                    <td>
                        <form th:attr="action=${'/request/accepted/' + user.id}">
                            <input type="submit" class="btn btn-success" th:attr="value=#{receiver.button}" />
                        </form>
                        <form th:attr="action=${'/request/blocked/' + user.id}">
                            <input type="submit" class="btn btn-danger" th:attr="value=#{receiver.blocked}" />
                        </form>
                    </td>
                </tr>
            </tbody>
        </table>
        <div th:replace="fragments/pagination"></div>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

Dicha vista se mostraría de la siguiente forma.



Para mostrar esta vista, se ha creado un nuevo método dentro de “RequestReceiver”.

```
@GetMapping("/requests")
public String showReceiverRequests(Model model, Pageable pageable,
        Principal principal) {
    logService.info(principal.getName() + " lista sus peticiones");
    User user = usersService.getUserByEmail(principal.getName());
    Page<User> users = new PageImpl<User>(new LinkedList<User>());
    Long id = user.getId();
    users = usersService.findAllByRequestReceiverId(pageable, id);
    model.addAttribute("usersList", users.getContent());
    model.addAttribute("page", users);
    return "/requests/receiver";
}
```

En “RequestService” se creado un nuevo método para cargar estas peticiones.

```
public Page<User> findAllByRequestReceiverId(Pageable pageable, Long id) {
    return usersRepository.findAllByRequestReceiverId(pageable, id);
}
```

Y en “RequestRepository” también se ha añadido un nuevo método para realizar la nueva consulta.

```
Page<User> findAllByRequestReceiverId(Pageable pageable, Long id);
```

Cuya consulta es:

- User.findAllByRequestReceiverId=SELECT r.sender FROM Request r WHERE r.status = 'SENT' AND r.receiver.id = ?1

7. Usuario registrado: aceptar una invitación recibida

Para aceptar una invitación existe el siguiente formulario en la vista “receiver.html”.

```
<form th:attr="action=${'/request/accepted/' + user.id}">
    <input type="submit" class="btn btn-success" th:attr="value=#{receiver.button}" />
</form>
```

Este formulario llama a “RequestController”. Este controlador acepta la solicitud en varios pasos.

```
@GetMapping("/request/accepted/{id}")
public String acceptedRequest(@PathVariable Long id, Principal principal) {
    User receiver = usersService.getUserByEmail(principal.getName());
    User sender = usersService.getUser(id);
    logService.info(principal.getName() + " acepto la amistad de "
            + sender.getEmail());
    acceptBoth(receiver, sender);
    createFriendshipBoth(receiver, sender);
    return "redirect:/requests";
}
```

1. Comprueba las peticiones que existen entre los dos usuarios.
    a. Si existen en ambas direcciones, las marca como aceptadas.
    b. Si no existe una de ellas, la crea y la marca como aceptada. Esta solución no es la más optima, pero como no se especifica en el enunciado de la práctica he optado por esta vía ya que es la más rápida.

2. Actualiza/añade estas solicitudes.
3. Crea la nueva amistad en la base de datos en ambas direcciones.

Para realizar estas acciones se han creado los siguientes métodos en “RequestController”.

```
/**
 * Aceptar las solicitudes de ambos usarios, si las hay y además crea la
 * amistad entre los usuarios
 *
 * @param receiver
 * @param sender
 */
private void acceptBoth(User receiver, User sender) {
    acceptRequest(receiver, sender);
    acceptRequest(sender, receiver);
}
```

```
/**
 * Acepta una solicitud de amistad. Le cambia el estado a aceptada y la
 * guarda en la base de datos
 *
 * @param receiver
 * @param sender
 */
private void acceptRequest(User receiver, User sender) {
    Request request = requestsService
            .findBySenderIdAndReceiverId(sender.getId(), receiver.getId());
    if (request == null) {
        request = new Request(sender, receiver);
    }
    request.accept();
    requestsService.modify(request);
}
```

```
/**
 * Crea la amistad entre dos usuarios
 *
 * @param receiver
 * @param sender
 */
private void createFriendshipBoth(User receiver, User sender) {
    createFriendship(receiver, sender);
    createFriendship(sender, receiver);
    usersService.modify(sender);
    usersService.modify(receiver);
}

private void createFriendship(User receiver, User sender) {
    Friendship friendship = new Friendship(receiver, sender);
    receiver.getFriends().add(friendship);
    sender.getiAmFriendOf().add(friendship);
    friendshipService.add(friendship);
}
```

Dentro de la entidad “Request” creamos el método “accept”

```
public void accept() {
    setStatus(ResquestStatus.ACCEPTED);
}
```

En la clase “RequestService” creamos el método “modify()”:

```
public void modify(Request request) {
    requestRepository.save(request);
}
```

Y para finalizar creamos el método “modify()” en “UsersService”.

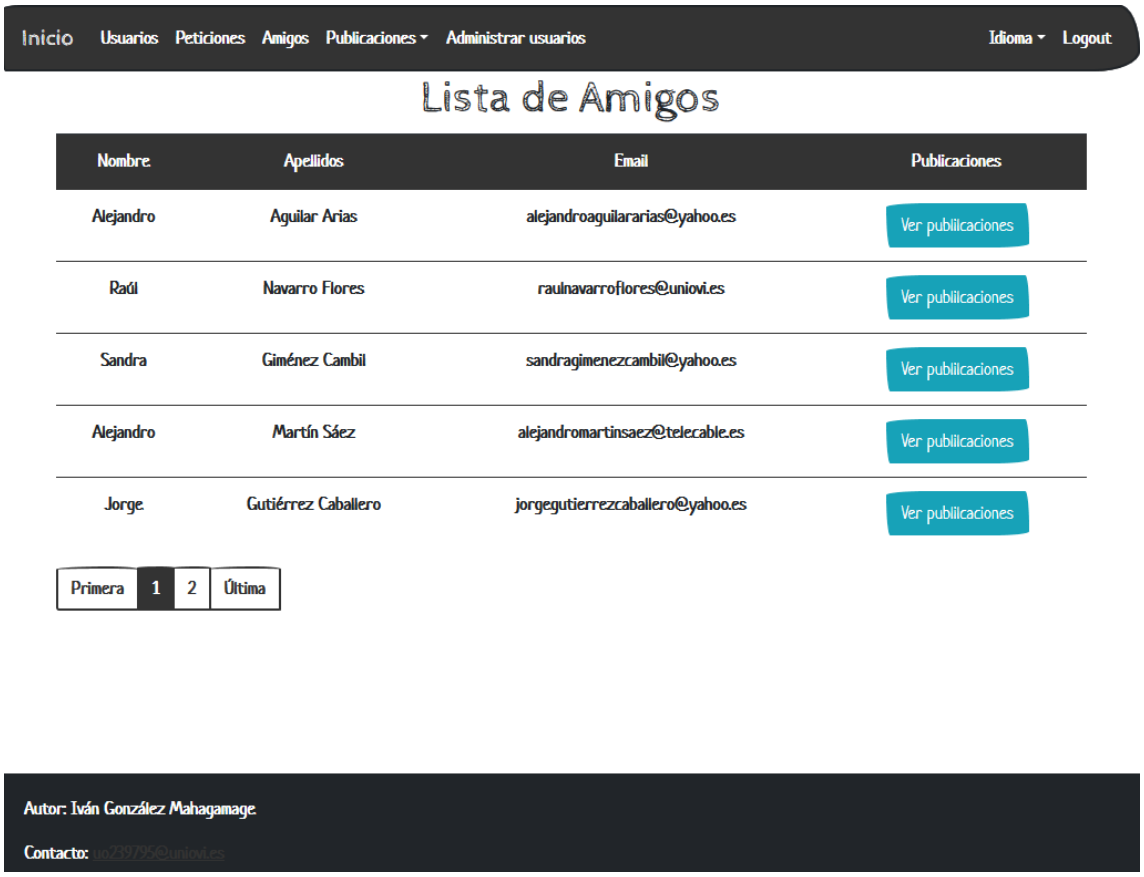
```
public void modify(User user) {
    usersRepository.save(user);
}
```

8. Usuario registrado: listar los usuarios amigos

Para ver los amigos de un usuario se ha creado la vista “friends.html” dentro del directorio “users”.

```
<!DOCTYPE html>
<html lang="es">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container" style="text-align: center">
        <h1 th:text="#{friends.title}">Esta es una zona privada la web</h1>
        <table class="table table-hover">
            <thead class="table-dark">
                <tr>
                    <th scope="col" th:text="#{signup.name}">Column heading</th>
                    <th scope="col" th:text="#{signup.surname}">Column heading</th>
                    <th scope="col">Email</th>
                    <th scope="col" th:text="#{nav.post}">Email</th>
                </tr>
            </thead>
            <tbody>
                <tr class="table-light" th:each="user:${usersList}">
                    <td th:text="${user.name}">Column content</td>
                    <td th:text="${user.surName}">Column content</td>
                    <td th:text="${user.email}">Column content</td>
                    <td>
                        <form th:attr="action=${'/post/friends/' + user.id}">
                            <input type="submit" class="btn btn-info" th:attr="value=#{post.button.friend}" />
                        </form>
                    </td>
                </tr>
            </tbody>
        </table>
        <div th:replace="fragments/pagination"></div>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

El resultado sería:



Para mostrar esta vista, se ha añadido el método “showFriends()” en “UserController”.

```
@GetMapping("/friends")
public String showFriends(Model model, Pageable pageable,
        Principal principal) {
    logService.info(principal.getName() + " lista sus amistades");
    User user = usersService.getUserByEmail(principal.getName());
    Page<User> users = usersService.findAllFriendsById(pageable,
            user.getId());
    model.addAttribute("usersList", users.getContent());
    model.addAttribute("page", users);
    return "/users/friends";
}
```

En “UsersServicies” se ha añadido el método “findAllFriendsById()”.

```
  public Page<User> findAllFriendsById(Pageable pageable, Long id) {
      return usersRepository.findAllFriendsById(pageable, id);
  }
```

Y en “UsersRepository” se ha añadido el método “findAllFriendsById()”.

```
Page<User> findAllFriendsById(Pageable pageable, Long id);
```

Cuya consulta es:

- User.findAllFriendsById=SELECT u.friend FROM Friendship u WHERE u.user.id = ?1

9. Usuario registrado: crear una nueva publicación

Para que el usuario cree nuevas publicaciones se ha creado la vista “add.html” dentro del directorio “posts”.

```
<!DOCTYPE html>
<html lang="es">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container">
        <h1 th:text="#{post.add.title}">Esta es una zona privada la web</h1>
        <form method="post" action="/post/add" enctype="multipart/form-data" th:object="${post}">
            <fieldset>
                <div class="form-group">
                    <label class="control-label col-sm-2" th:text="#{post.add.title.new}" for="title">Titulo</label>
                    <input class="form-control" type="text" name="title" required="required"
                           th:attr="title=#{post.title.title}, value=${post.title}" />
                    <div th:if="${#fields.hasErrors('title')}" class="alert alert-dismissible alert-danger">
                            <button type="button" class="close" data-dismiss="alert">&times;</button>
                            <strong th:errors="*{title}">Oh snap!</strong>
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-sm-2" th:text="#{post.add.text}" for="text">Contenido</label>
                    <textarea class="form-control" rows="3" cols="20" name="text" required="required"
                              th:attr="title=#{post.text.title}, value=${post.text}"></textarea>
                </div>
                <div class="form-group">
                    <input class="btn btn-outline-primary" type="file" name="imgn" accept=".jpeg, .png, .jpg" />
                </div>
                <div class="form-group">
                    <input class="btn btn-info" type="submit" th:attr="value=#{post.button}" />
                </div>
            </fieldset>
        </form>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

En esta vista ya están incluidos elementos del caso de uso 12 que se explicarán más adelante. El resultado de este html es el siguiente:

Para mostrar esta vista hay que implementar un nuevo método “formPost()” en “PostController”.

```
@GetMapping("/post/add")
public String formPost(Model model) {
    model.addAttribute("post", new Post());
    return "posts/add";
}
```

Y para procesar el formulario de creación hay que crear otro método en “PostController”.

```
@PostMapping("/post/add")
public String addPost(@ModelAttribute Post post,
        @RequestParam(value = "imgn", required = false) MultipartFile img,
        Principal principal, BindingResult result) {
    try {
        postValidator.validate(post, result);
        if (result.hasErrors()) {
            logService.error(
                    "Usuario introdujo mal los datos de la publicación");
            return "/posts/add";
        }
        User user = usersService.getUserByEmail(principal.getName());
        if (!img.getOriginalFilename().equals("")) {
            String fileName = postService.saveImg(img);
            logService.info(principal.getName()
                    + " ha realizado una nueva publicación");
            post.setImg("/imgUser/" + fileName);
        }
        post.setUser(user);
        post.setDate(new Date(System.currentTimeMillis()));
        postService.add(post);
        return "redirect:/post/list";
    } catch (IOException e) {
        return "redirect:/post/add";
    }
}
```

Obviando la parte de imágenes, este método llama primero un validador para comprobar los datos introducidos por el usuario. Esta clase es “PostValidator”.

```
@Component
public class PostValidator implements Validator {

    @Override
    public boolean supports(Class<?> aClass) {
        return User.class.equals(aClass);
    }

    @Override
    public void validate(Object target, Errors errors) {
        Post post = (Post) target;
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "title",
                "Error.empty");
        if (post.getTitle().length() < 2) {
            errors.rejectValue("title", "Error.post.title.length");
        }

    }
}
```

Una vez validado se guarda llamando a “PostService” con el método “add()”.

```
public void add(Post post) {
    postRepository.save(post);
}
```

10. Usuario registrado: listar mis publicaciones

Para listar las publicaciones del usuario se ha implementado la vista “list.html” en el directorio “posts”.

```
<!DOCTYPE html>
<html lang="es">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container" style="text-align: center">
        <h1 th:text="#{post.list.title}"></h1>
        <div class="jumbotron" th:each="post:${list}">
            <h1 class="display-3" th:text="${post.title}"></h1>
            <hr class="my-4" />
            <p th:text="${post.text}"></p>
            <img th:attr="src=${post.img}" width="100%" height="100%" />
            <p class="lead" th:text="${post.date}"></p>
        </div>
        <div th:replace="fragments/pagination"></div>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

Y se mostraría de la siguiente forma.

Para ello, se ha implementado el método “listPost” en “PostController”

```
@GetMapping("/post/list")
public String listPost(Model model, Pageable pageable,
        Principal principal) {
    User user = usersService.getUserByEmail(principal.getName());
    Page<Post> page = postService.findAll(pageable, user.getId());
    List<Post> posts = page.getContent();
    model.addAttribute("list", posts);
    model.addAttribute("page", page);
    return "posts/list";
}
```

En “PostService” se ha implementado el método “findAll”.

```
public Page<Post> findAll(Pageable pageable, Long id) {
    return postRepository.findAllByUserId(pageable, id);
}
```

Y en “PostRepository” el método “”.

```
Page<Post> findAllByUserId(Pageable pageable, Long id);
```

Cuya consulta es:

- Post.findAllByUserId=SELECT p FROM Post p where p.user.id = ?1 ORDER BY p.date ASC

11. Usuario registrado: listar las publicaciones de un usuario amigo

Para hacer este caso de uso, he creado unas vista denominada “friends.html” en la carpeta “posts”. Esta vista poco difiere a la de listar las propias publicaciones, únicamente se indica en el título de la página a quien pertenece estas publicaciones.

```
<!DOCTYPE html>
<html lang="es">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container" style="text-align: center">
        <h1 th:text="#{post.list.title}"></h1>
        <div class="jumbotron" th:each="post:${list}">
            <h1 class="display-3" th:text="${post.title}"></h1>
            <hr class="my-4" />
            <p th:text="${post.text}"></p>
            <img th:attr="src=${post.img}" width="100%" height="100%" />
            <p class="lead" th:text="${post.date}"></p>
        </div>
        <div th:replace="fragments/pagination"></div>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

El resultado es:

En “PostController” añadimos el siguiente método.

```
@GetMapping("/post/friends/{id}")
public String friendsPost(Model model, Pageable pageable,
        @PathVariable Long id, Principal principal) {
    User actual = usersService.getUserByEmail(principal.getName());
    User user = usersService.getUser(id);
    if (friendshipService.findByFriends(actual, user) == null) {
        return "redirect:/home";
    }
    Page<Post> page = postService.findAll(pageable, user.getId());
    List<Post> posts = page.getContent();
    model.addAttribute("friend", user.getName() + " " + user.getSurName());
    model.addAttribute("list", posts);
    model.addAttribute("page", page);
    return "posts/friends";

}
```

En este método comprobamos que los dos usuarios tienen amistad, en caso contrario devolvemos al usuario a la página principal.

12. Usuario registrado: crear una publicación con una foto adjunta

La vista usada para este caso es la misma que para el caso 9, voy a destacar el elemento que he añadido para que funcione.

```
<form method="post" action="/post/add" enctype="multipart/form-data" th:object="${post}">
    <fieldset>
        <div class="form-group">
            <label class="control-label col-sm-2" th:text="#{post.add.title.new}" for="title">Titulo</label>
            <input class="form-control" type="text" name="title" required="required"
                   th:attr="title=#{post.title.title}, value=${post.title}" />
            <div th:if="${#fields.hasErrors('title')}" class="alert alert-dismissible alert-danger">
                    <button type="button" class="close" data-dismiss="alert">&times;</button>
                    <strong th:errors="*{title}">Oh snap!</strong>
            </div>
        </div>
        <div class="form-group">
            <label class="control-label col-sm-2" th:text="#{post.add.text}" for="text">Contenido</label>
            <textarea class="form-control" rows="3" cols="20" name="text" required="required"
                    th:attr="title=#{post.text.title}, value=${post.text}"></textarea>
        </div>
        <div class="form-group">
            <input class="btn btn-outline-primary" type="file" name="imgn" accept=".jpeg, .png, .jpg" />
        </div>
        <div class="form-group">
            <input class="btn btn-info" type="submit" th:attr="value=#{post.button}" />
        </div>
    </fieldset>
```

En “PostController” comprobamos que la imagen es válida.

```
@PostMapping("/post/add")
public String addPost(@ModelAttribute Post post,
        @RequestParam(value = "imgn", required = false) MultipartFile img,
        Principal principal, BindingResult result) {
    try {
        postValidator.validate(post, result);
        if (result.hasErrors()) {
            logService.error(
                    "Usuario introdujo mal los datos de la publicación");
            return "/posts/add";
        }
        User user = usersService.getUserByEmail(principal.getName());
        if (!img.getOriginalFilename().equals("")) {
            String fileName = postService.saveImg(img);
            logService.info(principal.getName()
                    + " ha realizado una nueva publicación");
            post.setImg("/imgUser/" + fileName);
        }
        post.setUser(user);
        post.setDate(new Date(System.currentTimeMillis()));
        postService.add(post);
        return "redirect:/post/list";
    } catch (IOException e) {
        return "redirect:/post/add";
    }
}
```

Despues en “PostService” guardamos el post con la imagen.

```
/**
 * Guarda una imagen en la aplicación
 *
 * @param img
 * @return
 * @throws IOException
 */
public String saveImg(MultipartFile img) throws IOException {
    saveImgResource(img);
    saveImgTarget(img);
    return img.getOriginalFilename();
}


/**
 * Guarda una imagen en un determinado directorio
 *
 * @param img
 * @param path
 * @throws IOException
 */
private void saveImgByPath(MultipartFile img, String path)
        throws IOException {
    String fileName = img.getOriginalFilename();
    InputStream is = img.getInputStream();
    Files.copy(is, Paths.get(path + fileName),
            StandardCopyOption.REPLACE_EXISTING);
}
```

La imagen se guarda en dos sitios.

1. En la carpeta “resource” para que sea persistente pero no se cargara la imagen de este directorio hasta que no se reinicie o refresque el proyecto.

```
/**
 * Guarda en una imagen en la carpeta resource, aquí al guardarla, no se
 * ve hasta que no se recarga la aplicación pero si es persistente.
 * No se borra cuando se reinicia la aplicación
 *
 * @param img
 * @throws IOException
 */
private void saveImgResource(MultipartFile img) throws IOException {
    CreateFolder.createFolder("src/main/resources/static/imgUser");
    saveImgByPath(img, "src/main/resources/static/imgUser/");
}
```

2. En la carpeta “target” para que se muestre la imagen sin recargar la aplicación, pero esta carpeta se destruye cuando se recarga, no es persistente

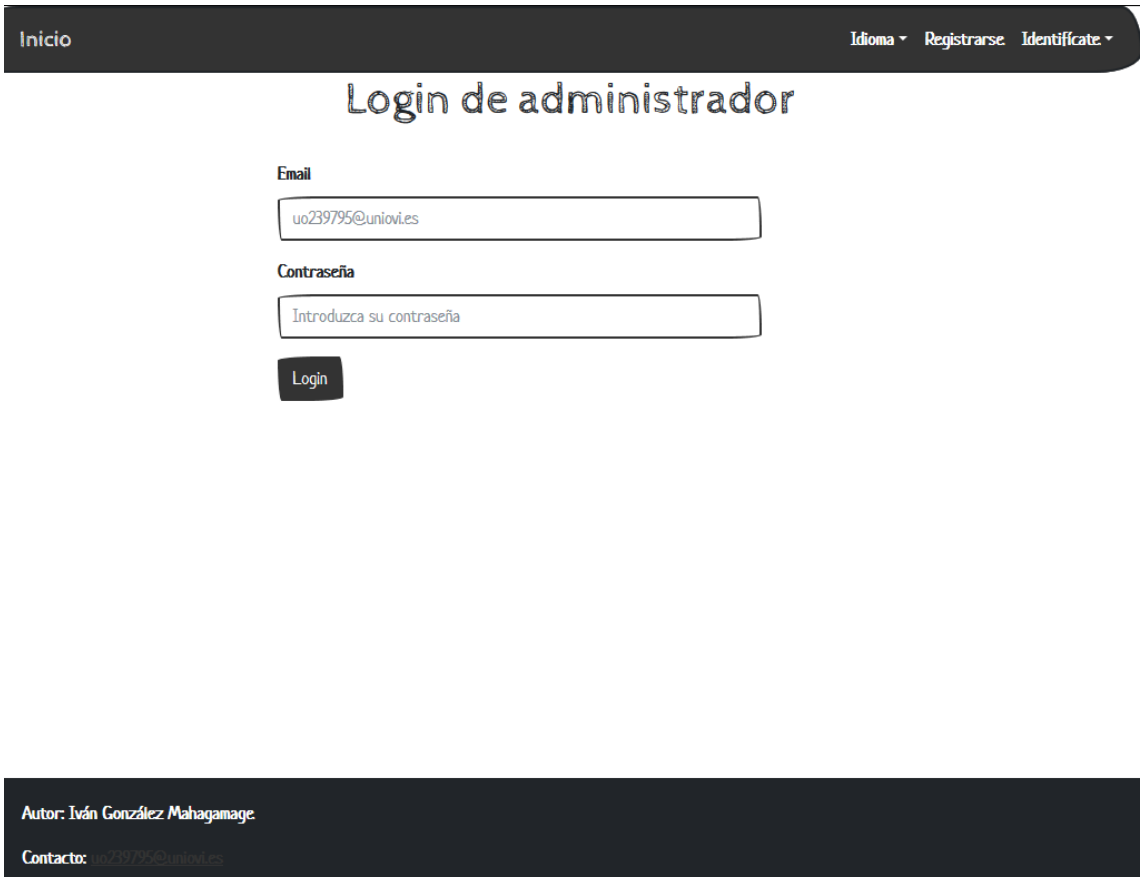
```java
/**
 * Guarda en una imagen en la carpeta target, para que así se visualice
 * sin recargar la aplicación. Pero es transient, no se guarda cuando se
 * acaba la ejecución.
 *
 * @param img
 * @throws IOException
 */
private void saveImgTarget(MultipartFile img) throws IOException {
    CreateFolder.createFolder("target/classes/static/imgUser/");
    saveImgByPath(img, "target/classes/static/imgUser/");
}
```

13. Público: iniciar sesión como administrador

Para hacer este caso de uso, se creado una vista denominada “adminLogin.html”.

```html
<!DOCTYPE html>
<html lang="es">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container">
        <h1 th:text="#{admin.login.title}">Idéntificate</h1>
        <form class="form-horizontal" method="post" action="/admin/login">
            <fieldset>
                <div class="form-group">
                    <label class="control-label col-sm-2" for="email">Email</label>
                    <div class="col-sm-10">
                        <input type="email" class="form-control" name="email"
                               placeholder="uo239795@uniovi.es" required="required"
                               th:attr="title=#{signup.email.title}" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-sm-2" for="password" th:text="#{signup.password}">Password:</label>
                    <div class="col-sm-10">
                        <input type="password" class="form-control" name="password"
                               th:attr="placeholder=#{signup.password.placeholder}, title=#{signup.password.placeholder}"
                               pattern=".{2,}" required="required" />
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-sm-offset-2 col-sm-10">
                        <button type="submit" class="btn btn-primary">Login</button>
                    </div>
                </div>
                <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
            </fieldset>
            <div th:if="${noAdmin} != null" class="alert alert-dismissible alert-danger">
                <button type="button" class="close" data-dismiss="alert">&times;</button>
                <strong th:text="#{admin.noadmin.error}">Oh snap!</strong>
            </div>
            <div th:if="${password} != null" class="alert alert-dismissible alert-danger">
                <button type="button" class="close" data-dismiss="alert">&times;</button>
                <strong th:text="#{admin.password.error}">Oh snap!</strong>
            </div>
            <div th:if="${noExist} != null" class="alert alert-dismissible alert-danger">
                <button type="button" class="close" data-dismiss="alert">&times;</button>
                <strong th:text="#{admin.noexist.error}">Oh snap!</strong>
            </div>
        </form>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

Cuya visualización es.



En “UserController” se han creado dos métodos para que funcione esta vista.

- Un método para mostrar la vista.

```
@GetMapping("/admin/login")
public String adminLoginGet() {
    logService.info("Usuario se intenta loggear como admin");
    return "/adminLogin";
}
```

- Un método para comprobar el formulario.
  1. Comprueba si el usuario existe.
  2. Comprueba que el usuario es un administrador.
  3. Comprueba la contraseña sea correcta.

  Si todas las comprobaciones son correcta logue al usuario, usando el autologin de Spring Security y lo lleva a la pantalla de “home”

```
@PostMapping("/admin/login")
public String adminLoginPost(@Validated User user, Model model) {
    User user1 = usersService.getUserByEmail(user.getEmail());
    if (user1 == null) {
        logService.info("Usuario " + user.getEmail() + " no existe");
        model.addAttribute("noExist", "noExist");
        return "/adminLogin";
    }
    if (!user1.getRole().equals(rolesService.getAdmin())) {
        logService.info("Usuario " + user.getEmail() + " no es un Admin");
        model.addAttribute("noAdmin", "noAdmin");
        return "/adminLogin";
    }
    if (!bCryptPasswordEncoder.matches(user.getPassword(),
            user1.getPassword())) {
        logService.info("Usuario " + user.getEmail() + " no es un Admin");
        model.addAttribute("password", "password");
        return "/adminLogin";
    }
    securityService.autoLogin(user.getEmail(), user.getPassword());
    logService.info(
            "Usuario " + user.getEmail() + " se ha logueado como Admin");
    return "redirect:/home";
}
```

14. Consola de administración: listar todos los usuarios de la aplicación

Para mostrar esta lista se ha creado una nueva opción en el nav solo visible para los administradores.

- Usuario.

- Administrador.

Después se ha creado la vista “adminlist.html”.

```
<!DOCTYPE html>
<html lang="en">
<head th:replace="fragments/head" />

<body>
    <nav th:replace="fragments/nav" />
    <main class="container" style="text-align: center">
        <h1 th:text="#{admin.list.title}">Esta es una zona privada la web</h1>
        <table class="table table-hover">
            <thead class="table-dark">
                <tr>
                    <th scope="col" th:text="#{signup.name}">Column heading</th>
                    <th scope="col" th:text="#{signup.surname}">Column heading</th>
                    <th scope="col">Email</th>
                    <th scope="col" th:text="#{list.add}"></th>
                </tr>
            </thead>
            <tbody>
                <tr class="table-light" th:each="user:${usersList}">
                    <td th:text="${user.name}">Column content</td>
                    <td th:text="${user.surName}">Column content</td>
                    <td th:text="${user.email}">Column content</td>
                    <td>
                        <form method="post" th:attr="action=${'/user/delete/' + user.id}">
                            <input type="submit" class="btn btn-danger" th:attr="value=#{delete.button}" />
                        </form>
                    </td>
                </tr>
            </tbody>
        </table>
        <div th:replace="fragments/pagination"></div>
        <h2 th:text="#{list.search.title}">Buscar usuario</h2>
        <form class="navbar-form" action="/admin/list">
            <div class="form-group">
                <input name="searchText" type="text" class="form-control"
                       size="50" th:attr="placeholder=#{list.search}" />
            </div>
            <button type="submit" class="btn btn-primary" th:text="#{list.search.button}">Buscar</button>
        </form>
    </main>
    <footer th:replace="fragments/footer" />
</body>

</html>
```

La cual se muestra de la siguiente forma.

En la clase “UserController” se añadido el método “”.

```
@GetMapping("/admin/list")
public String listUserAdmin(Model model, Principal principal,
        Pageable pageable,
        @RequestParam(value = "", required = false) String searchText) {
    logService.info(
            "Administrador " + principal.getName() + " "
                    + "lista los usuarios");
    User user = usersService.getUserByEmail(principal.getName());
    Page<User> page = getUsers(pageable, searchText, user);
    List<User> list = page.getContent();
    model.addAttribute("usersList", list);
    model.addAttribute("page", page);
    return "/users/adminlist";
}
```

15. Consola de administración: Consola de administración: eliminar usuario

Para este caso se ha usado la misma vista que en el caso 14. Voy a hacer hincapié en el elemento de esta vista usado para este caso.

```
<form method="post" th:attr="action=${'/user/delete/' + user.id}">
    <input type="submit" class="btn btn-danger" th:attr="value=#{delete.button}" />
</form>
```

Este formulario llama al método “deleteUser()” de la clase “UserController”.

```
@PostMapping("/user/delete/{id}")
public String deleteUser(Principal principal, @PathVariable Long id) {
    usersService.delete(id);
    logService.info("Administrador " + principal.getName()
            + " elimino al usuario con id " + id);
    return "redirect:/admin/list";
}
```

Este a su vez llama a “delete()” de la clase “UserService”

```
public void delete(Long id) {
    usersRepository.delete(id);
}
```

Prueba Unitarias

1.1. Registro de Usuario con datos válidos.

```
/**
 * Registro de Usuario con datos válidos.
 */
@Test
public void Test01_1_RegVal() {
    email = Random.email();
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Registrarse")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("email")).click();
    driver.findElement(By.name("email")).clear();
    driver.findElement(By.name("email")).sendKeys(email);
    driver.findElement(By.name("name")).click();
    driver.findElement(By.name("name")).clear();
    driver.findElement(By.name("name")).sendKeys("Test");
    driver.findElement(By.name("surName")).clear();
    driver.findElement(By.name("surName")).sendKeys("Selenium");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.name("passwordConfirm")).clear();
    driver.findElement(By.name("passwordConfirm")).sendKeys("123456");
    driver.findElement(By.xpath("//input[@value='Registrar']")).click();
    test.textoPresentePagina("Usuario Autenticado como");
    test.textoPresentePagina(email);
}
```

1.2. Registro de Usuario con datos inválidos (repetición de contraseña invalida).

```
/**
 * Registro de Usuario con datos inválidos
 * (repetición de contraseña invalida).
 */
@Test
public void Test01_2_RegInval() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Registrarse")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("email")).click();
    driver.findElement(By.name("email")).clear();
    driver.findElement(By.name("email")).sendKeys(Random.email());
    driver.findElement(By.name("name")).click();
    driver.findElement(By.name("name")).clear();
    driver.findElement(By.name("name")).sendKeys("Test");
    driver.findElement(By.name("surName")).clear();
    driver.findElement(By.name("surName")).sendKeys("Selenium");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.name("passwordConfirm")).clear();
    driver.findElement(By.name("passwordConfirm")).sendKeys("1234567");
    driver.findElement(By.xpath("//input[@value='Registrar']")).click();
    assertEquals("Error: las contraseñas no coinciden",
            test.closeAlertAndGetItsText());
}
```

2.1. Inicio de sesión con datos válidos.

```
/**
 * Inicio de sesión con datos válidos.
 */
@Test
public void Test02_1_Inval() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username"))
            .sendKeys("ivangonzalezmahagamage@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.textoPresentePagina("Usuario Autenticado como");
    test.textoPresentePagina("ivangonzalezmahagamage@gmail.com");
    test.textoPresentePagina("Logout");
    test.textoNoPresentePagina("Login");
}
```

2.2. Inicio de sesión con datos inválidos.

```
/**
 * Inicio de sesión con datos inválidos
 * (usuario no existente en la aplicación).
 */
@Test
public void Test02_2_InInVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys("noExisto@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.textoPresentePagina("Login");
}
```

3.1. Acceso al listado de usuarios desde un usuario en sesión.

```
/**
 * Acceso al listado de usuarios desde un usuario en sesión
 */
@Test
public void Test03_1_LisUsrVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username"))
            .sendKeys("ivangonzalezmahagamage@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Usuario Autenticado como");
    driver.findElement(By.linkText("Usuarios")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Buscar usuario");
    test.textoPresentePagina("Lista de usuarios");
}

/**
```

3.2. Intento de acceso con URL desde un usuario no identificado al listado de usuarios desde un usuario en sesión.

```
/**
 * Intento de acceso con URL desde un usuario no identificado al
 * listado de usuarios desde un usuario en sesión.
 * Debe producirse un acceso no permitido a
 * vistas privadas.
 */
@Test
public void Test03_2_LisUsrInVal() {
    driver.get(URL + "?lang=es");
    driver.get(URL + "user/list");
    test.waitChangeWeb();
    test.textoNoPresentePagina("Lista de usuarios");
    test.textoPresentePagina("Login");
}
```

4.1. Realizar una búsqueda valida en el listado de usuarios desde un usuario en sesión.

```
/**
 * Realizar una búsqueda valida en el listado de usuarios desde
 * un usuario en sesión.
 */
@Test
public void Test04_1_BusUsrVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username"))
            .sendKeys("ivangonzalezmahagamage@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Usuario Autenticado como");
    driver.findElement(By.linkText("Usuarios")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("searchText")).click();
    driver.findElement(By.name("searchText")).clear();
    driver.findElement(By.name("searchText")).sendKeys(email);
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina(email);
}
```

4.2. Intento de acceso con URL a la búsqueda de usuarios desde un usuario no identificado.

```
/**
 * Intento de acceso con URL a la búsqueda de usuarios desde un usuario no
 * identificado. Debe producirse un acceso no permitido a vistas privadas.
 */
@Test
public void Test04_2_BusUsrInVal() {
    driver.get(URL + "?lang=es");
    driver.get(URL + "user/list");
    test.waitChangeWeb();
    test.textoNoPresentePagina("Lista de usuarios");
    test.textoNoPresentePagina("Buscar usuario");
    test.textoPresentePagina("Login");
}
```

5.1. Enviar una invitación de amistad a un usuario de forma valida.

```
/**
 * Enviar una invitación de amistad a un usuario de forma valida.
 */
@Test
public void Test05_1_InvVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username"))
            .sendKeys("ivangonzalezmahagamage@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Usuario Autenticado como");
    driver.findElement(By.linkText("Usuarios")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("searchText")).click();
    driver.findElement(By.name("searchText")).clear();
    driver.findElement(By.name("searchText")).sendKeys(email);
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("searchText")).sendKeys(email);
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    driver.findElement(By.xpath("//input[@value='Enviar solicitud']"))
            .click();
    test.textoNoPresentePagina(email);
}
```

5.2. Enviar una invitación de amistad a un usuario al que ya le habíamos invitado la invitación previamente.

```
/**
 * Enviar una invitación de amistad a un usuario al que ya le habíamos
 * invitado la invitación previamente. No debería dejarnos enviar
 * la invitación, se podría ocultar el botón de enviar invitación o
 * notificar que ya había sido enviada previamente.
 */
@Test
public void Test05_2_InvInVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username"))
            .sendKeys("ivangonzalezmahagamage@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Usuario Autenticado como");
    driver.findElement(By.linkText("Usuarios")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("searchText")).click();
    driver.findElement(By.name("searchText")).clear();
    driver.findElement(By.name("searchText")).sendKeys(email);
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("searchText")).sendKeys(email);
    test.textoPresentePagina(email);
    test.textoPresentePagina("Solicitud enviada");
}
```

6.1. Listar las invitaciones recibidas por un usuario.

```
/**
 * Listar las invitaciones recibidas por un usuario, realizar
 * la comprobación con una lista que al menos tenga una
 * invitación recibida.
 */
@Test
public void Test06_1_LisInvVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Peticiones")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Iván");
    test.textoPresentePagina("González Mahagamage");
}
```

7.1. Aceptar una invitación recibida.

```
/**
 * Aceptar una invitación recibida.
 */
@Test
public void Test07_1_AcepInvVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Peticiones")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Iván");
    test.textoPresentePagina("González Mahagamage");
    driver.findElement(By.xpath("//input[@value='Aceptar invitación']"))
            .click();
    test.waitChangeWeb();
    test.textoNoPresentePagina("Aceptar Petición");
    test.textoNoPresentePagina("Bloquear");
}
```

8.1. Listar los amigos de un usuario, realizar la comprobación con una lista que al menos tenga un amigo.

```
/**
 * Listar los amigos de un usuario, realizar la comprobación con una
 * lista que al menos tenga un amigo.
 */
@Test
public void Test08_1_ListAmiVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Amigos")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Iván");
    test.textoPresentePagina("González Mahagamage");
}
```

9.1. Crear una publicación con datos válidos.

```
/**
 * Crear una publicación con datos válidos.
 */
@Test
public void Test09_1_PubVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Publicaciones")).click();
    driver.findElement(By.linkText("Añadir publicación")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("title")).click();
    driver.findElement(By.name("title")).clear();
    driver.findElement(By.name("title")).sendKeys("Prueba Post");
    driver.findElement(By.name("text")).click();
    driver.findElement(By.name("text")).clear();
    driver.findElement(By.name("text"))
            .sendKeys("Prueba de contenido de post");
    driver.findElement(By.xpath("//input[@value='Enviar']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Lista de publicaciones");
    test.textoPresentePagina("Prueba Post");
    test.textoPresentePagina("Prueba de contenido de post");
}
```

10.1. Acceso al listado de publicaciones desde un usuario en sesión.

```
/**
 * Acceso al listado de publicaciones desde un usuario en sesión.
 */
@Test
public void Test10_1_LisPubVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Publicaciones")).click();
    driver.findElement(By.linkText("Listar mis publicaciones")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Lista de publicaciones de");
    test.textoPresentePagina("Prueba Post");
    test.textoPresentePagina("Prueba de contenido de post");
}
```

11.1. Listar las publicaciones de un usuario amigo.

```
/**
 * Listar las publicaciones de un usuario amigo
 */
@Test
public void Test11_1_LisPubAmiVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Amigos")).click();
    test.waitChangeWeb();
    driver.findElement(By.xpath("//input[@value='Ver publilcaciones']"))
            .click();
    test.waitChangeWeb();
    test.textoPresentePagina("Lista de publicaciones de");
    test.textoPresentePagina("Iván González Mahagamage");
}
```

11.2. Utilizando un acceso vía URL tratar de listar las publicaciones de un usuario que no sea amigo del usuario identificado en sesión.

```
/**
 * Utilizando un acceso vía URL tratar de listar
 * las publicaciones de un usuario
 * que no sea amigo del usuario identificado en sesión.
 */
@Test
public void Test11_2_LisPubAmiVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.get(URL + "post/friends/3");
    test.textoPresentePagina("Esta es una zona privada la web");
    test.textoPresentePagina("Usuario Autenticado como");
    test.textoPresentePagina(email);

}
```

12.1. Crear una publicación con datos válidos y una foto adjunta.

```
/**
 * Crear una publicación con datos válidos y una foto adjunta.
 */
@Test
public void Test12_1_PubFot1Val() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Publicaciones")).click();
    driver.findElement(By.linkText("Añadir publicación")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("title")).click();
    driver.findElement(By.name("title")).clear();
    driver.findElement(By.name("title")).sendKeys("Prueba Post con foto");
    driver.findElement(By.name("text")).click();
    driver.findElement(By.name("text")).clear();
    String imagenDePrueba = "file:///";
    imagenDePrueba += System.getProperty("user.dir");
    imagenDePrueba += "\\imagenDePrueba.jpg";
    imagenDePrueba = imagenDePrueba.replace("\\", "/");
    driver.findElement(By.name("text")).sendKeys(imagenDePrueba);
    driver.findElement(By.name("imgn")).sendKeys(imagenDePrueba);
    driver.findElement(By.xpath("//input[@value='Enviar']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Lista de publicaciones");
    test.textoPresentePagina("Prueba Post con foto");
    test.textoPresentePagina(imagenDePrueba);
}
```

12.2. Crear una publicación con datos válidos y sin una foto adjunta.

```
/**
 * Crear una publicación con datos válidos y sin una foto adjunta.
 */
@Test
public void Test12_2_PubFot2Val() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username")).sendKeys(email);
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Publicaciones")).click();
    driver.findElement(By.linkText("Añadir publicación")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("title")).click();
    driver.findElement(By.name("title")).clear();
    driver.findElement(By.name("title")).sendKeys("Prueba Post sin foto");
    driver.findElement(By.name("text")).click();
    driver.findElement(By.name("text")).clear();
    driver.findElement(By.name("text"))
            .sendKeys("Prueba de contenido de post sin foto");
    driver.findElement(By.xpath("//input[@value='Enviar']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Lista de publicaciones");
    test.textoPresentePagina("Prueba Post sin foto");
    test.textoPresentePagina("Prueba de contenido de post sin foto");
}
```

13.1. Inicio de sesión como administrador con datos válidos.

```
/**
 * Inicio de sesión como administrador con datos válidos.
 */
@Test
public void Test13_1_AdInVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Administrador")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Login de administrador");
    driver.findElement(By.name("email")).click();
    driver.findElement(By.name("email")).clear();
    driver.findElement(By.name("email"))
            .sendKeys("ivangonzalezmahagamage@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.textoPresentePagina("Usuario Autenticado como");
    test.textoPresentePagina("ivangonzalezmahagamage@gmail.com");
    test.textoPresentePagina("Logout");
    test.textoNoPresentePagina("Login de administrador");
}
```

13.2. Inicio de sesión como administrador con datos inválidos.

```
/**
 * Inicio de sesión como administrador con datos inválidos (usar los
 * datos de un usuario que no tenga perfil administrador).
 */
@Test
public void Test13_2_AdInInVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Administrador")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("email")).click();
    driver.findElement(By.name("email")).clear();
    driver.findElement(By.name("email")).sendKeys("igm1990@hotmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Login de administrador");
    driver.findElement(By.xpath("(//button[@type='button'])[2]")).click();
}
```

14.1. Desde un usuario identificado en sesión como administrador listar a todos los usuarios de la aplicación.

```
/**
 * Desde un usuario identificado en sesión como administrador
 * listar a todos los usuarios de la aplicación.
 */
@Test
public void Test14_1_AdLisUsrVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Administrador")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Login de administrador");
    driver.findElement(By.name("email")).click();
    driver.findElement(By.name("email")).clear();
    driver.findElement(By.name("email"))
            .sendKeys("ivangonzalezmahagamage@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Administrar usuarios")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Administrador: Listado de usuarios");
}
```

15.1. Desde un usuario identificado en sesión como administrador eliminar un usuario existente en la aplicación.

```
/**
 * Desde un usuario identificado en sesión como administrador eliminar un
 * usuario existente en la aplicación.
 */
@Test
public void Test15_1_AdBorUsrVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Administrador")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Login de administrador");
    driver.findElement(By.name("email")).click();
    driver.findElement(By.name("email")).clear();
    driver.findElement(By.name("email"))
            .sendKeys("ivangonzalezmahagamage@gmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.linkText("Administrar usuarios")).click();
    test.waitChangeWeb();
    test.textoPresentePagina("Administrador: Listado de usuarios");
    driver.findElement(By.name("searchText")).click();
    driver.findElement(By.name("searchText")).clear();
    driver.findElement(By.name("searchText")).sendKeys(email);
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    test.waitChangeWeb();
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    driver.findElement(By.xpath("//input[@value='Eliminar']"));
}
```

15.2. Intento de acceso vía URL al borrado de un usuario existente en la aplicación.

```
/**
 * Intento de acceso vía URL al borrado de un usuario existente en la
 * aplicación. Debe utilizarse un usuario identificado en sesión
 * pero que no tenga perfil de administrador.
 */
@Test
public void Test15_2_AdBorUsrInVal() {
    driver.get(URL + "?lang=es");
    driver.findElement(By.linkText("Identifícate")).click();
    driver.findElement(By.linkText("Usuario")).click();
    test.waitChangeWeb();
    driver.findElement(By.name("username")).click();
    driver.findElement(By.name("username")).clear();
    driver.findElement(By.name("username"))
            .sendKeys("igm1990@hotmail.com");
    driver.findElement(By.name("password")).clear();
    driver.findElement(By.name("password")).sendKeys("123456");
    driver.findElement(By.xpath("//button[@type='submit']")).click();
    driver.get(URL + "admin/list");
    test.textoPresentePagina("Access is denied");
}
```

Otros aspectos

1. Usuario registrado: bloquear invitaciones de amistad de un usuario

Este caso de uso no está dentro de la especificación, pero lo implemente después de lo comentado en la tutoría grupal. Para ello, en la vista usada en el caso 7 se añadió un formulario adicional.

```
<form th:attr="action=${'/request/blocked/' + user.id}">
    <input type="submit" class="btn btn-danger" th:attr="value=#{receiver.blocked}" />
</form>
```

En “PostController” creamos un método que ejecute el formulario.

```
@GetMapping("/request/blocked/{id}")
public String blockedRequest(@PathVariable Long id, Principal principal) {
    User receiver = usersService.getUserByEmail(principal.getName());
    User sender = usersService.getUser(id);
    blockedBoth(receiver, sender);
    return "redirect:/requests";
}
```

Después dentro de “PostService” añadimos los siguiente métodos.

```
/**
 * Aceptar las solicitudes de ambos usarios, si las hay y además crea la
 * amistad entre los usuarios
 *
 * @param receiver
 * @param sender
 */
private void blockedBoth(User receiver, User sender) {
    blockRequest(receiver, sender);
    blockRequest(sender, receiver);
}

/**
 * Acepta una solicitud de amistad. Le cambia el estado a aceptada y la
 * guarda en la base de datos
 *
 * @param receiver
 * @param sender
 */
private void blockRequest(User receiver, User sender) {
    Request request = requestsService
            .findBySenderIdAndReceiverId(sender.getId(), receiver.getId());
    if (request == null) {
        request = new Request(sender, receiver);
    }
    request.block();
    requestsService.modify(request);
}
```