RED SOCIAL

SDI

Pelayo Díaz Soto – UO251000
Saúl Castillo Valdés – UO251370

Índice

Por modificaciones de última hora, puede haber pequeñas variaciones en los archivos html y clases.

Implementación de casos de uso

1. Público: registrarse como usuario

Antes de implementar esta opción debíamos implementar la clase usuario, su controlador, su servicio y su repositorio (entre otras clases necesarias para conseguir seguridad y algunos ficheros html y css para poder visualizar la aplicación).

La clase usuario tiene el siguiente aspecto:

```
@Entity
public class User {
	@Id
	@GeneratedValue
	private long id;

	@Column(unique=true)
	private String email;

	private String name;
	private String lastName;

	private String password;
	@Transient //propiedad que no se almacena e la tabla.
	private String passwordConfirm;

	private String role;

	private Boolean isAddFriend = true;

	@OneToMany(mappedBy = "transmitter", cascade = CascadeType.ALL)
	private Set<Request> sent /*= new HashSet<Request>()*/;

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

	// Amistades actuales
	@OneToMany(mappedBy = "friend", cascade = CascadeType.ALL)
	private Set<Friendship> friends ;

	// Publicaciones realizadas
	@OneToMany(mappedBy = "author", cascade = CascadeType.ALL)
	private Set<Post> posts ;

	public User() { }

	. . .
```

Como se puede observar en el código, esta clase esta mapeada y relacionada con otras clases cuya implementación se mostrará más adelante en este documento.

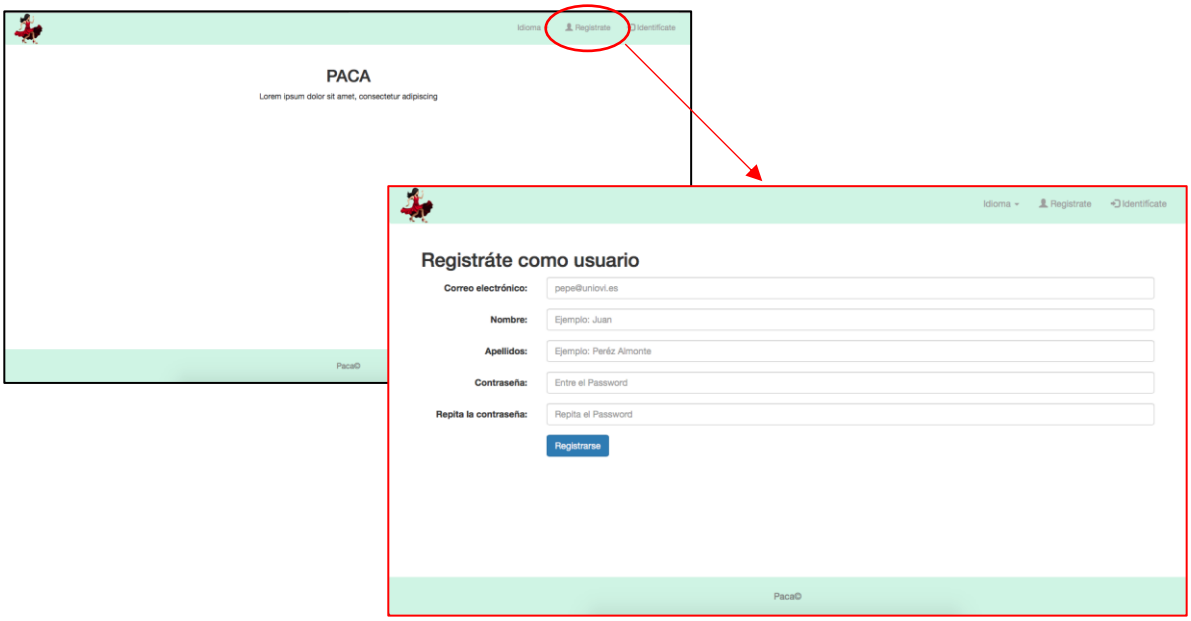
En lo que respecta al caso de uso concreto, para implementarlo hemos tenido que crear un fichero .html con el siguiente aspecto:

```
<!DOCTYPE html>
<html lang="en">
<head th:replace="fragments/head" />
<body>
		<!-- Barra de Navegación superior -->
		<nav th:replace="fragments/nav" />
		<div class="container">
			<h2>Regístráte como usuario</h2>
			<form class="form-horizontal" method="post" action="/signup"
				th:object="${user}">
				<div class="form-group">
					<label class="control-label col-sm-2" for="email" th:text="#{signup.email}">Email:</label>
					<div class="col-sm-10">
						<input type="text" class="form-control" name="email"
							placeholder="pepe@uniovi.es" required="true" />
						<span class="text-danger" th:if="${#fields.hasErrors('email')}"
							th:errors="*{email}">
					</div>
				</div>
				<div class="form-group">
					<label class="control-label col-sm-2" for="name" th:text="#{signup.name}">Nombre:</label>
					<div class="col-sm-10">
						<input type="text" class="form-control" name="name"
							placeholder="Ejemplo: Juan" required="true" /> <span
							class="text-danger" th:if="${#fields.hasErrors('name')}"
							th:errors="*{name}" />
					</div>
				</div>
				<div class="form-group">
					<label class="control-label col-sm-2" for="lastName" th:text="#{signup.lastname}">Apellidos:</label>
					<div class="col-sm-10">
						<input type="text" class="form-control" name="lastName"
							placeholder="Ejemplo: Peréz Almonte" required="true" /> <span
							class="text-danger" th:if="${#fields.hasErrors('lastName')}"
							th:errors="*{lastName}" />
					</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"
							placeholder="Entre el Password" /> <span class="text-danger"
							th:if="${#fields.hasErrors('password')}" th:errors="*{password}" />
					</div>
				</div>
				<div class="form-group">

					<label class="control-label col-sm-2" for="passwordConfirm" th:text="#{signup.passwordConfirm}">
						Repita el Password:</label>
					<div class="col-sm-10">
						<input type="password" class="form-control" name="passwordConfirm"
							placeholder="Repita el Password" />
						<span class="text-danger"
						th:if="${#fields.hasErrors('passwordConfirm')}"
						th:errors="*{passwordConfirm}" />
					</div>
				</div>
				<div class="form-group">
					<div class="col-sm-offset-2 col-sm-10">
						<button type="submit" class="btn btn-primary" th:text="#{signup.signup}">Registrarse</button>
					</div>
				</div>
			</form>
		</div>
		<footer th:replace="fragments/footer" />
</body>
</html>
```

Una vez realizado este .html se deben implementar una petición GET y POST en UsersController:

```
@RequestMapping(value = "/signup", method = RequestMethod.GET)
        public String signup(Model model) {
                model.addAttribute("user", new User());
                return "signup";
        }

        @RequestMapping(value = "/signup", method= RequestMethod.POST)
        public String signup(@Validated User user, BindingResult result ,Model model) {
                signUpFormValidator.validate(user, result);
                if (result.hasErrors()) {
                        return "signup";
                }
                user.setRole(rolesService.getRoles()[0]);
                usersService.addUser(user);
                securityService.autoLogin(user.getEmail(), user.getPasswordConfirm());
                return "home";
        }
```

En la aplicación se visualiza de la siguiente manera:



2. Pública: iniciar sesión

Para este punto hemos tenido que añadir un nuevo .html, y recoger la petición GET en el UsersController (la petición POST se la dejamos a WebSecurityConfig).

El html cuyo nombre es login.html (de la carpeta templates) tiene el siguiente aspecto:

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

	<nav th:replace="fragments/nav" />

	<div class="container">

		<h2 th:text="#{login.ident}">Identifícate</h2>
		<form class="form-horizontal" method="post" action="/login"
			th:object="${user}">
			<div class="form-group">
				<label class="control-label col-sm-2" for="username"
					th:text="#{login.email}">Email:</label>
				<div class="col-sm-10">
					<input type="text" class="form-control" name="username"
						placeholder="Ejemplo: pacaSalitasLindas@yahoo.es" required="true" />
				</div>
				<div class="form-group">
					<label class="control-label col-sm-2" for="username"
						th:text="#{login.password}">Contraseña:</label>
					<div class="col-sm-10">
						<input type="password" class="form-control" name="password"
							placeholder="Introducir Password" required="true" />
					</div>
				</div>
				<div class="form-group">

				<div style="color: red;" th:if="${error != null}">
					<span th:text="login.failed">Inicio de sesión fallido</span>
				</div>
				<div class="col-sm-offset-2 col-sm-10">
					<button type="submit" class="btn btn-primary"
						th:text="#{login.login}">Login</button>
				</div>
			</div>
			<input type="hidden" name="${_csrf.parameterName}"
				value="${_csrf.token}" />
		</form>

	</div>

	<footer th:replace="fragments/footer" />

</body>
</html>
```

A continuación, se muestra parte del código de WebSecurityConfig y el código de la petición GET:

En UsersController:

```
@RequestMapping(value = "/login", method = RequestMethod.GET)
	public String login(Model model, @RequestParam(required=false) String error) {

		if(error !=null) {
			model.addAttribute("error", "Inicio de sesión fallido");
		}
		model.addAttribute("user", new User());
		httpSession.setAttribute("origen", rolesService.getRoles()[0]);
		return "login";
	}
```

En WebSecurityConfig:

```
@Override
	protected void configure(HttpSecurity http) throws Exception { http
		.csrf().disable()
		.authorizeRequests()
			.antMatchers("/css/**", "/img/**", "/script/**", "/", "/signup", "/admin/login").permitAll()
			.antMatchers("/user/delete/*").hasAnyAuthority("ROLE_ADMIN")
			.anyRequest().authenticated()
				.and()
		.formLogin()
			.loginPage("/login")
			.permitAll()
			.defaultSuccessUrl("/user/list")
			.failureUrl("/login?error=aka")
				.and()
		.logout()
			.permitAll();
	}
```

Para que funcionará la petición POST hemos tenido que poner los nombres correspondientes a los inputs en el .html, es decir username y password.

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

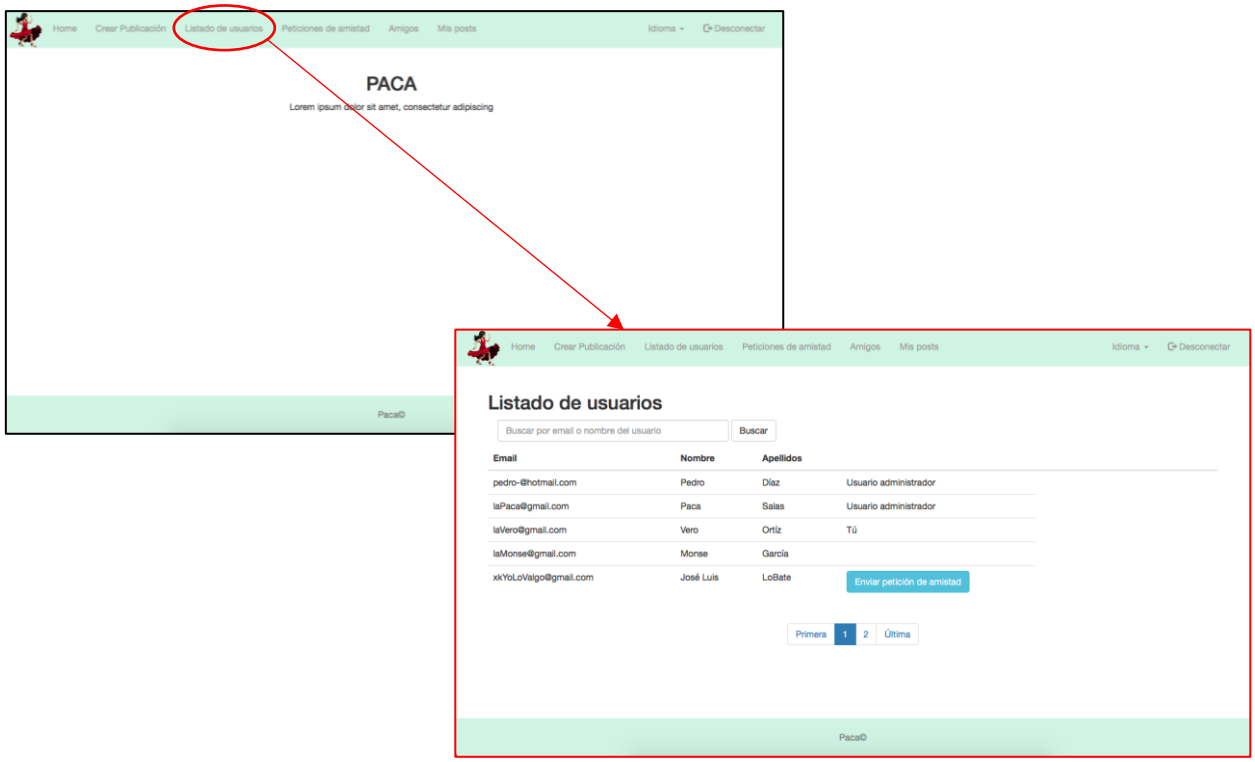
En un primer enfoque de este caso de uso, la lista mostraba todos los usuarios menos al que estaba conectado. Pero el enunciado dice claramente “todos los usuarios” y eso entendemos incluye también al que está conectado.
Para solucionar este problema sin que en futuros casos de uso, se permitiera al usuario mandarse a sí mismo una petición o incluso eliminarse, en lugar de esas opciones aparece la palabra “Tú” indicando al usuario conectado que esa fila hace referencia a su usuario.

Hecha esta aclaración, para implementar esta lista, aparte obviamente de incluir la opción de menú en el nav, hemos creado un nuevo .html (este html corresponde a todos los casos de uso realizados, por esta razón aparecen las opciones de enviar petición o eliminar usuario). Por la extensión de este fichero .html no se mostrará en este documento, el .html en concreto se encuentra en la carpeta templates/user, y tiene como nombre list.html.

Además de este nuevo html hemos tenido que recoger la petición GET en el controlador de User:

```
@RequestMapping("/user/list" )
public String getListado(Model model, Pageable pageable,
                        @RequestParam(value = "", required=false) String searchText,
                                        Principal principal){


        if(httpSession.getAttribute("origen").equals(rolesService.getRoles()[1]))
                if(usersService.getUserEmail(principal.getName()).getRole().equals(
                                        rolesService.getRoles()[0])) {
                        return "redirect:/logout";
                }


        String email = principal.getName();

        Page<User> users = new PageImpl<User>(new LinkedList<User>());

        if (searchText != null && !searchText.isEmpty()) {
                users=usersService.searchUsersByDNIAndName (pageable, searchText, email);
        }else {
                users = usersService.getUsers(pageable, email);
        }
        model.addAttribute("usersList", users.getContent() );
        model.addAttribute("page", users);
        model.addAttribute("conectado",
                        usersService.getUserEmail(principal.getName()).getEmail());

        return "user/list";
}
```

El aspecto en la interface sería el siguiente:



No se ha destacado antes, pero para hacer la paginación hemos tenido que implentar un fragmento nuevo he incluir este en el html, así como añadir el paramentro Pageable a la consulta en el UsersRepository.

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

En el mismo .html del apartado anterior, introducimos la búsqueda con las siguientes líneas:

```
<form class="navbar-form" action="/user/list">

                        <div class="form-group">

                                <input name="searchText" type="text" class="form-control" size="50"
                                        placeholder="Buscar por email o nombre del usuario">
                        </div>
                        <button id="search" type="submit" class="btn btn-default" th:text="#{user.list.search}">Buscar</button>

                </form>
```

En el UsersController que mostramos en el apartado anterior ya se mostraba la implementación necesaria para este caso de uso. A continuación, se muestran las líneas de código concretas:

```
@RequestMapping("/user/list" )
public String getListado(Model model, Pageable pageable,
                        @RequestParam(value = "", required=false) String searchText,
                                        Principal principal){


        if(httpSession.getAttribute("origen").equals(rolesService.getRoles()[1]))
                if(usersService.getUserEmail(principal.getName()).getRole().equals(
                                        rolesService.getRoles()[0])) {
                        return "redirect:/logout";
                }


        String email = principal.getName();

        Page<User> users = new PageImpl<User>(new LinkedList<User>());

        if (searchText != null && !searchText.isEmpty()) {
                        users=usersService.searchUsersByDNIAndName (pageable, searchText, email);
        }else {
                        users = usersService.getUsers(pageable, email);
        }
        model.addAttribute("usersList", users.getContent() );
        model.addAttribute("page", users);
        model.addAttribute("conectado",
                        usersService.getUserEmail(principal.getName()).getEmail());

        return "user/list";
}
```

En UsersService hemos implementado el método que llama a la consulta de la interface UsersRepository de la siguiente manera:

```
public Page<User> searchUsersByDNIAndName(Pageable pageable,
                                String searchText, String email){
        Page<User> users = new PageImpl<User>(new LinkedList<User>());
        searchText = "%"+searchText+"%";
        users = usersRepository.searchByEmailAndNameWithOutCurrentUser(
                                pageable, searchText, email);
        return users;
}
```

Y en UsersRepository, la consulta es:

```
@Query("SELECT u FROM User u WHERE u.email != ?2 and (LOWER(u.email) LIKE LOWER(?1) OR LOWER(u.name) LIKE LOWER(?1))")
    Page<User> searchByEmailAndNameWithOutCurrentUser(Pageable pageable,String seachtext, String email);
```

(*) Antes de realizar lo descrito anteriormente en este apartado, se tuvo que crear una nueva clase Request y relacionarla con la clase User:

```
@Entity
public class Request {

        @Id
        @GeneratedValue
        private Long id;
        private String descripcion;

        @ManyToOne
        private User transmitter; //emite

        @ManyToOne
        private User receiver; //recibe

        public Request() {}
```

5. Usuario registrado: listar las invitaciones de amistad recibidas

En primer lugar, introducimos esta opción en el list.html mencionado en el apartado número 3. Para ello añadimos a la tabla las siguientes líneas:

```
<td sec:authorize="hasRole('ROLE_USER')"><div>
        <div th:if="${user.isAddFriend && user.role=='ROLE_USER'
                        && !conectado.equals(user.email)}">

                <button type="button" th:id="${'addButton' + user.id}"
                        class="btn btn-info">Enviar petición de amistad</button>
                        <script th:inline="javascript">
                        /*<![CDATA[*/
                                $("#addButton[[${user.id}]]").click(function() {
                                        $.get("/user/[[${user.id}]]/isAddFriend", function(data){

                                                var numberPage = [[${param.page}]];
                                                var urlUpdate = '/user/list/update';
                                                if( numberPage != null){
                                                        urlUpdate += "?page="+numberPage[0];
                                                        }

                                                $("#tableUsers").load(urlUpdate);
                                        })
                                                });
                                                /*]]>*/
                        </script>
                </div>
        <div th:if="${user.role!='ROLE_USER' && !conectado.equals(user.email)}">
        Usuario administrador
        </div>
        <div th:if="${conectado.equals(user.email)}">
        Tú
        </div>

        </div></td>
```

En este punto no habría nada relacionado con los roles, pero esta instantánea pertenece al proyecto ya acabado.

Como se ve en el html, se hace refencia a la url "/user/[[${user.id}]]/isAddFriend" y a la url '/user/list/update'. Estas url mandarán peticiones que se han de recoger en el controller de usuarios.

Las siguientes líneas de código corresponde a la clase UsersController:

```
@RequestMapping("/user/list/update")
	public String updateList(Model model, Pageable pageable, Principal principal){
		Page<User> users = usersService.getUsers(pageable,  principal.getName());
		model.addAttribute("usersList", users.getContent() );
		model.addAttribute("page", users);
		model.addAttribute("conectado",
			usersService.getUserEmail(principal.getName()).getEmail());
		return "user/list :: tableUsers";
	}

	@RequestMapping(value="/user/{id}/isAddFriend", method=RequestMethod.GET)
	public String setResendTrue(Model model, @PathVariable Long id,
		Principal principal){
		requestsService.sendRequest(id,
			usersService.getUserEmail(principal.getName()).getId());
		return "redirect:/user/list";
	}
```

En el método sendRequest vemos que llamamos a un método de RequestsService que tiene la siguiente implementación:

```
public void sendRequest(Long id_to,Long id_from ) {
			User transmitter= usersService.getUser(id_from);
			User receiver = usersService.getUser(id_to);
			usersService.setUserIsAddFriend(false, id_to);
			Request request = new Request(transmitter.getFullName()
				+" quiere ser tu amig@", transmitter, receiver);
			requestsRepository.save(request);
		}
```

Creamos el objeto Request y lo guardamos en la base de datos.

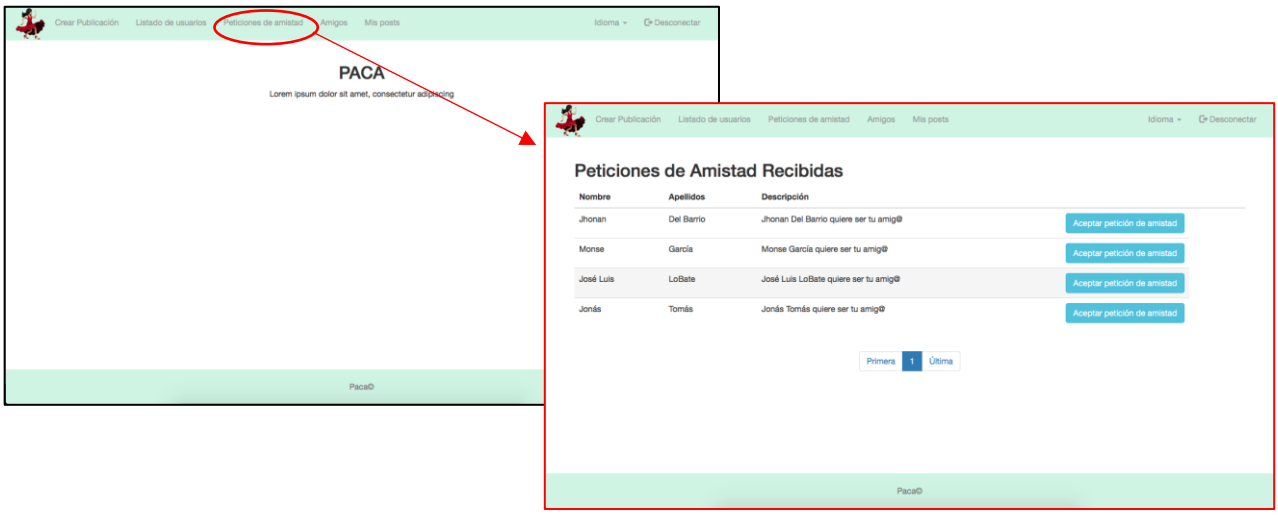
6. Usuario registrado: listar las invitaciones de amistad recibidas

Creamos un nuevo html llamado list.html en la carpeta templates/request, muy similar a al list.html mencionado anteriormente. Además, añadimos un acceso directo en el nav.

Además añadimos las peticiones al controller de Request:

```
@RequestMapping("/request/list")
        public String getList(Model model, Pageable pageable, Principal principal){

                String email = principal.getName(); // email es el name de la autenticación
                User user = usersService.getUserEmail(email);
                Page<Request> request = new PageImpl<Request>(new LinkedList<Request>());

                request = requestService.searchRequestReceived(pageable, user) ;

                model.addAttribute("requestList", request.getContent());
                model.addAttribute("page", request);
                return "/request/list";
        }
```

A esta vista se accede así:



7. Usuario registrado: aceptar una invitación recibida

Añadimos funcionalidad al botón de cada línea de la tabla de la vista que se mostró en el apartado anterior.

```
<td><div>
		<button type="button" th:id="${'addButton' + request.id}"
                                                                                class="btn btn-info">Aceptar
petición de amistad</button>
		<script th:inline="javascript">
			/*<![CDATA[*/
                                                $("#addButton[[${request.id}]]").click(function() {

		$.get("/request/[[${request.id}]]/accept", function(data){

			var numberPage = [[${param.page}]]
			var urlUpdate = '/request/list/update';
			if( numberPage != null){
				urlUpdate += "?page="+numberPage[0];
                                                                                                }


		$("#tableRequests").load(urlUpdate);


		})
		});
			/*]]>*/
		</script>
</div></td>
```

En el controller de Request implementamos las peticiones:

```
@RequestMapping("/request/list/update")
	public String updateList(Model model, Pageable pageable, Principal principal){
		String email = principal.getName(); // email es el name de la autenticación
		User user = usersService.getUserEmail(email);
		Page<Request> request = new PageImpl<Request>(new LinkedList<Request>());

		request = requestService.searchRequestReceived(pageable, user) ;

		model.addAttribute("requestList", request.getContent());
		return "/request/list :: tableRequests";
	}


	@RequestMapping(value="/request/{id}/accept", method=RequestMethod.GET)
	public String setResendTrue(Model model, @PathVariable Long id, Principal principal){
		requestService.acceptRequest( id, usersService.getUserEmail(principal.getName()).getId());
		return "redirect:/user/list";
	}
```

Al pulsar sobre este botón desaparecerá la fila de la tabla y desde ese momento los usuarios serán amigos.
Este código pertenece a RequestsService:

```
public void acceptRequest(Long id_request, Long idUser) {
                friendshipService.addFriend(requestsRepository.
                        findOtherUserIdFromRequest(id_request), idUser);
                requestsRepository.delete(id_request);
        }
```

(*) No se ha mencionado al principio de este punto, pero este caso de uso conlleva la creación de una nueva clase Friendship (además, por supuesto de relacionarla con la clase User) que tiene el siguiente aspecto:

```
@Entity
public class Friendship {

        @Id
        @GeneratedValue
        private Long id;

        @ManyToOne
        private User friend;

        @ManyToOne
        private User user;

        public Friendship() {}


        . . .
```

8. Usuario registrado: listar los usuarios amigos

Como cada vez que hemos listado algo, creamos un nuevo html de nombre friendList.html, esta vez en la carpeta templates/user.

Además tenemos que recoger la petición en el controller especifico, en este caso FriendshipController:

```
@RequestMapping("/user/friendsList" )
	public String getListadoAmigos(Model model, Pageable pageable, Principal principal){

		String email = principal.getName();

		Page<User> users = new PageImpl<User>(new LinkedList<User>());

		users = friendshipService.getFriends(pageable, email);

		model.addAttribute("usersList", users.getContent() );
		model.addAttribute("page", users);

		return "user/friendsList";
	}
```

9. Usuario registrado: crear una nueva publicación

Para implementar este caso de uso, lo primero que se debe hacer es crear la clase que representará a una publicación. Decimos llamarla Post y la relacionamos con la clase User.

```
@Entity
public class Post {

	@Id
	@GeneratedValue
	private Long id;

	@ManyToOne
	private User author;

	private String date; // con el formato YYYY/MM/DD
	private String text;
	private String title;
	private String imageURL;

	public Post() {}
```

En esta implementación ya incluimos la posibilidad de que tengan imagen, con el atributo imageURL (que indica el lugar en el que está la imagen dentro del proyecto).

El siguiente paso que seguir es crear un nuevo fichero .html para crear estos posts. Se llamará create.html y se encontrará en la carpeta templates/post:

```
. . .
<div class="container">
		<h2 id="crearPubli">Crea una nueva publicación</h2>

		<form class="form-horizontal" method="post" action="/post/create"
			th:object="${post}"  enctype="multipart/form-data">
			<div class="form-group">
				<label class="control-label col-sm-2" for="title" th:text="#{post.create.title}">Título</label>
				<div class="col-sm-10">
					<input type="text" class="form-control" name="title"
						placeholder="La novela de Paca" required="true" />
					<span class="text-danger" th:if="${#fields.hasErrors('title')}"
						th:errors="*{title}">
				</div>
			</div>
			<div class="form-group">
				<label class="control-label col-sm-2" for="text" th:text="#{post.create.text}">Texto</label>
				<div class="col-sm-10">
					<input style="height:250px; align-content:bottom;" type="text" class="form-control" name="text"
						placeholder="Las mejores tres páginas de mi vida" required="true" /> <span
						class="text-danger" th:if="${#fields.hasErrors('text')}"
						th:errors="*{text}" />
				</div>
			</div>
			<div class="form-group">
				<label class="control-label col-sm-2" for="image" th:text="#{post.create.image}">Adjuntar imagen</label>
				<div class="col-sm-10">
					<input type="file" class="form-control" name="image"
					  accept=".png, .jpg"/>
				</div>
			</div>


			<div class="form-group">
				<div class="col-sm-offset-2 col-sm-10">
					<button type="submit" class="btn btn-primary" th:text="#{post.create.post}">Publicar</button>
				</div>
			</div>
		</form>
	</div>


	<footer th:replace="fragments/footer" />
</body>
</html>
```

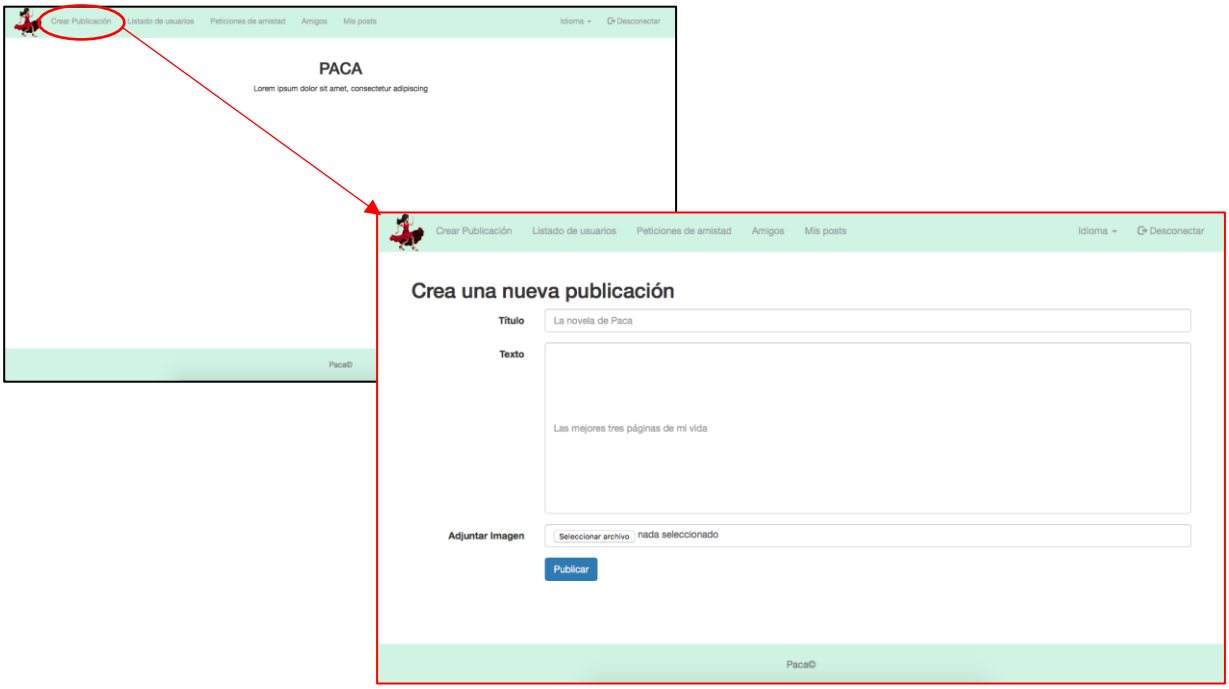
En la clase PostController recogemos las peticiones GET y POST, en el código que se muestra a continuación ya está implementado para admitir imágenes, por esto en vez de poner @RequestMaping usamos @PostMapping :

```
@PostMapping(value = "/post/create")
	public String signup(@Validated Post post, BindingResult result,Model model, Principal principal,
				@RequestParam("image") MultipartFile image) {

			createPostFormValidator.validate(post, result);
			if (result.hasErrors()) {
				return "/post/create";
			}

			try {
				String fileName = image.getOriginalFilename();
				InputStream is = image.getInputStream();
				Files.copy(is, Paths.get("src/main/resources/static/img/post/" + fileName),
						StandardCopyOption.REPLACE_EXISTING);
				post.setImageURL("/img/post/" + fileName);
			} catch (IOException e) {
//				e.printStackTrace();
			}

			post.setAuthor(usersService.getUserEmail(principal.getName()));
			post.setDateToday();
			postService.createPost(post);
			return "home";
	}

@RequestMapping(value = "/post/create", method = RequestMethod.GET)
	public String signup(Model model) {
			model.addAttribute("post", new Post());
			return "post/create";
	}
```

En el método createPost de PostService se implementa así:

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

Esta vista se podrá visualizar de la siguiente manera:



10. Usuario registrado: listar mis publicaciones

Como en el caso de las otras listas, el primer paso para listar las publicaciones es hacer un list.html, que esta vez estará en templates/post:

```
<!DOCTYPE html>
<html lang="en">
<head th:replace="fragments/head" />
<body>
	<!-- Barra de Navegación superior -->
	<nav th:replace="fragments/nav" />

	<div class="container">
		<h2 id="listarPubli" th:text="${nameAuthor}">Publicaciones</h2>

		<div class="table-responsive">
			<table class="table table-hover" th:fragment="tablePosts"
				id="tablePosts">
				<thead>
					<tr>
						<th >Imagen</th>
						<th>Título</th>
						<th>Fecha</th>
						<th class="col-md-1"></th>
						<th class="col-md-1"></th>
					</tr>
				</thead>
				<tbody>
					<tr th:each="post : ${postsList}">
						<td style="width:250px; vertical-align:middle"><img th:if="${post.imageURL != null}" th:src="${post.imageURL}" style="width:250px"" /></td>
						<td style="width:250px; vertical-align:middle" th:text="${post.title}">Título post</td>
						<td  style="width:250px; vertical-align:middle" th:text="${post.date}">Fecha post</td>


					</tr>
				</tbody>
			</table>
		</div>
		<footer th:replace="fragments/pagination" />
	</div>
	<footer th:replace="fragments/footer" />
</body>
</html>
```

Ahora deberemos recoger la petición de /post/list en el PostController:

```
@RequestMapping("/post/list" )
	public String getListado(Model model, Pageable pageable, Principal principal){

		String email = principal.getName();

		Page<Post> post = new PageImpl<Post>(new LinkedList<Post>());

		post = postService.getPosts(pageable, email);

		model.addAttribute("postsList", post.getContent() );
		model.addAttribute("nameAuthor", "Mis Publicaciones");
		model.addAttribute("page", post);

		return "post/list";
	}
```

Lo que hacemos aquí es buscar los posts del usuario que esté conectado en ese momento.

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

En el friendList.html añadimos un enlace en el nombre del usuario que nos redirigirá a la lista de publicaciones de ese usuario:

```
. . .
<tbody>
	<tr th:each="user : ${usersList}">
		<td th:text="${user.email}">pepe@uniovi.es</td>
		<td><a th:href="${'/post/' + user.id + '/list'}"  th:text="${user.name}">Nombre
del alumno</a></td>
		<td th:text="${user.lastName}">Apellidos del alumno</td>

			</tr>
		</tbody>
. . .
```

Ahora en la clase PostController recogemos la petición del href del código anterior:

```
@RequestMapping(value="/post/{id}/list", method=RequestMethod.GET)
	public String setResendTrue(Model model, @PathVariable Long id , Pageable pageable, Principal principal){

		if(!friendshipService.areFriends(id, usersService.getUserEmail(principal.getName()).getId())) {
			return "redirect:/user/friendsList";
		}
		Page<Post> post = new PageImpl<Post>(new LinkedList<Post>());

		post = postService.getPosts(pageable, usersService.getUser(id).getEmail());

		model.addAttribute("postsList", post.getContent() );
		model.addAttribute("nameAuthor", "Publicaciones de " + usersService.getUser(id).getFullName());
		model.addAttribute("page", post);
		return "/post/list";
	}
```

Lo se remarca en el código anterior es para que si un usuario introduce esta url y el id que introduce no pertenece a uno de sus amigos, le redirigimos a la página de posts del usuario que está intentando acceder a la url.

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

El código necesario para este caso de uso ya se ha mostrado en el apartado 9, pero ahora resaltaremos el código que hace refencia a este caso de uso:

En el create.html:

```
. . .
<div class="form-group">
		<label class="control-label col-sm-2" for="image" th:text="#{post.create.image}">Adjuntar
imagen</label>
		<div class="col-sm-10">
			<input type="file" class="form-control" name="image"
					accept=".png, .jpg"/>
			</div>
		</div>
. . .
```

En el PostController:

```
@PostMapping(value = "/post/create")
	public String signup(@Validated Post post, BindingResult result,Model model, Principal principal,
				@RequestParam("image") MultipartFile image) {

		createPostFormValidator.validate(post, result);
		if (result.hasErrors()) {
			return "/post/create";
		}

		try {
			String fileName = image.getOriginalFilename();
			InputStream is = image.getInputStream();
			Files.copy(is, Paths.get("src/main/resources/static/img/post/" + fileName),
					StandardCopyOption.REPLACE_EXISTING);
			post.setImageURL("/img/post/" + fileName);
		} catch (IOException e) {
		}

		post.setAuthor(usersService.getUserEmail(principal.getName()));
		post.setDateToday();
		postService.createPost(post);
		return "home";
	}
```

Como se puede ver seguimos uno de los enfoques que se nos recomendaba en el pdf del campus virtual y hemos guardado las imágenes en una carpeta del proyecto, concretamente scr/main/resources/static/img/post.

13. Público: iniciar sesión como administrador

Para la realización de este apartado hemos llevado a cabo el primer enfoque que se nos recomienda en el pdf de posibles enfoques…. La única diferencia es que en este enfoque se indica que al iniciar sesión con un usuario normal se redirigiría a éste a /home, nosotros seguiremos redirigiéndolos a /user/list como indicaba el caso de uso de inicio de sesión.

```
@RequestMapping("/user/list" )
public String getListado(Model model, Pageable pageable,
                        @RequestParam(value = "", required=false) String searchText,
                                        Principal principal){


        if(httpSession.getAttribute("origen").equals(rolesService.getRoles()[1]))
                if(usersService.getUserEmail(principal.getName()).getRole().equals(
                                        rolesService.getRoles()[0])) {
                        return "redirect:/logout";
                }


        String email = principal.getName();

        Page<User> users = new PageImpl<User>(new LinkedList<User>());

        if (searchText != null && !searchText.isEmpty()) {
                users=usersService.searchUsersByDNIAndName (pageable, searchText, email);
        }else {
                users = usersService.getUsers(pageable, email);
        }
        model.addAttribute("usersList", users.getContent() );
        model.addAttribute("page", users);
        model.addAttribute("conectado",
                        usersService.getUserEmail(principal.getName()).getEmail());

        return "user/list";
}
```

Al httpSession que vemos en este código tiene un atributo origen que adquiere el valor de un rol dependiendo de donde se inicie sesión:

```
@RequestMapping(value="/admin/login", method = RequestMethod.GET)
        public String loginAdmin(Model model, @RequestParam(required=false) String error) {

                        if(error !=null) {
                                model.addAttribute("error", "Inicio de sesión fallido");
                        }
                        model.addAttribute("user", new User());
                        httpSession.setAttribute("origen", rolesService.getRoles()[1]);

                        return "admin/login";
        }
```

```
@RequestMapping(value = "/login", method = RequestMethod.GET)
	public String login(Model model, @RequestParam(required=false) String error) {

		if(error !=null) {
			model.addAttribute("error", "Inicio de sesión fallido");
		}
		model.addAttribute("user", new User());
		httpSession.setAttribute("origen", rolesService.getRoles()[0]);
		return "login";
	}
```

Como se ve no es que se guarde el rol del usuario que inicia sesión, si no que se guarda el rol de usuario si se accede desde /login y el de administrador si se accede /admin/login.

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

En este caso de uso utilizaremos la misma petición que para los usuarios normales. Como no indica si el administrador podrá mandar peticiones, crear posts, etc… anulamos todas estas opciones para él.

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

En primer lugar, incluimos esta opción en el list.html que se encuentra en templates/user:

```
. . .
<td><a th:href="${'/user/delete/' + user.id}"
                                                    sec:authorize="hasRole('ROLE_ADMIN')"

        th:text="#{user.list.removeUser}">Eliminar</a></td>
. . .
```

Ahora desde UsersController recogemos la petición que lanzará el código anterior:

```
@RequestMapping("/user/delete/{id}")
        public String deleteMark(@PathVariable Long id) {
                usersService.deleteUser(id);
                return "redirect:/user/list";
        }
```

El método de deleteUser(Long id) ira eliminando primero al usuario de todos las tablas de BDD, es decir Request, Post y Friendship y después de la tabla User, para que no se produzcan fallos al eliminarlo directamente desde la tabla User.

Implementación de pruebas

Para el desarrollo de las pruebas se ha utilizado Selenium, para simular la interacción que realizaría un usuario con la web.

Se han desarrollado un total de 21 casos de prueba que han conseguido ejecutarse de manera satisfactoria.

El estilo de la implementación de las pruebas es el mismo que el visto en las prácticas de la asignatura.

Deben ejecutarse en este orden, pues algunas dependen de información introducida por casos anteriores.

Cabe destacar que, para evitar problemas provocados por caracteres especiales en los tests a la hora de ejecutarlos, hemos optado por realizarlos en inglés. Para ello se ha modificado el método setUp() para que antes de que se ejecute cada test, se cambie el idioma a inglés. Esto además verifica la correcta internacionalización del proyecto.

```
@Before
	public void setUp() {
		driver.navigate().to(URL);
		PO_NavView.changeIdiom(driver, "btnEnglish");
	}
```

Prueba 1: Registro de un nuevo usuario con datos correctos.

```
@Test
	public void P01_RegVal() {
		// Vamos al formulario de registro
		PO_HomeView.clickOption(driver, "signup", "class", "btn btn-
primary");
		//Comprobamos que estamos en la página correcta
		PO_View.checkElement(driver, "id", "RegistroTitulo");
		// Rellenamos el formulario.
		PO_RegisterView.fillForm(driver, "laLoles@gmail.com",
"Loles", "Fuentes", "123456", "123456");
		// Comprobamos que entramos en la sección privada
		PO_View.checkElement(driver, "text", "PACA");
	}
```

Esta prueba se basa en ir hasta el formulario de registro y rellenarlo con datos válidos para incorporar un nuevo usuario a la aplicación.
Para ello presionamos la opción de menú correspondiente con el registro para a continuación comprobar que estamos en la página adecuada.

Siguiendo con la arquitectura Page Object dejamos que sea PO_RegisterView la encargada de rellenar el formulario.
Una vez rellenado el formulario e iniciada la sesión se comprueba que, tras esto, se ha redirigido a la página correcta.

Prueba 2: Registro incorrecto

En esta prueba trataremos de comprobar que el formulario de registro produce un error si no se cumplen todas las condiciones.

```
// Vamos al formulario de registro
            PO_HomeView.clickOption(driver, "signup", "class", "btn btn-
primary");
            //Comprobamos que estamos en la página correcta
            PO_View.checkElement(driver, "id", "RegistroTitulo");
```

Para ello presionamos la opción de menú de registro y comprobamos que estamos en la página correcta. Una vez ahí, procederemos a rellenar el formulario con diferentes casos erróneos, estos son: email repetido, corto y con formato incorrecto, nombre y apellidos cortos, contraseña corta o no coincidente al volver a introducirla.
Y se comprueban los mensajes de error con:

PO_RegisterView.checkKey(driver, "Error.signup.email.duplicate",
PO_Properties.getENGLISH());

Prueba 3: Inicio de Sesión Fallido

```
@Test
	public void P03_InInVal() {
            // Vamos al formulario de incio de sesion
            PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
            //Comprobamos que estamos en la página correcta
            PO_ListUsersView.checkElement(driver, "id", "tituloLogin");
            //Rellenamos el formulario: Usuario no existe
            PO_LogInView.fillForm(driver, "noExisto@gmail.com",
"123456");
            //Comprobamos el fallo comporbando que seguimos en la página
            PO_ListUsersView.checkElement(driver, "id", "tituloLogin");
            //Rellenamos el formulario: Contraseña inválida
            PO_LogInView.fillForm(driver, "laPaca@gmail.com", "1234567");
            //Comprobamos el fallo
            PO_ListUsersView.checkElement(driver, "id", "tituloLogin");
	}
```

Se siguen los pasos anteriores, pero esta vez nos dirigimos al botón de inicio de sesión y es esa la página que comprobamos. Rellenemos el formulario a través de su Page Object correspondiente y probamos dos casos: el usuario no existe y contraseña incorrecta.

Prueba 4: Inicio de sesión válido y acceso a listado de usuarios

```
@Test
	public void P04_InVal_LisUsrVal(){
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Comprobamos que estamos en la página correcta
		PO_ListUsersView.checkElement(driver, "id", "tituloLogin");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laPaca@gmail.com", "123456");
		// Comprobamos que entramos en la sección privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
	}
```

En esta prueba hemos juntado dos: el inicio de sesión correcto y el acceso al listado de amigos. Para ello, seguimos los pasos del test anterior pero esta vez rellenando el formulario de forma correcta. A continuación, comprobamos que la página a la que se nos redirige es la del listado de usuarios de la aplicación.

Prueba 5: Acceso inválido a listado de usuarios desde URL

```
@Test
	public void P05_LisUsrInVal(){
		// Intentamos acceder al listado de usuarios sin logearnos
/user/list
		driver.navigate().to(URL+"/user/list");
		//Comprobamos que se nos redirige a la pagina de inicio de
sesion
		PO_LogInView.checkElement(driver, "free", "//h2[contains(@id,
'tituloLogin')]");
	}
```

Para realizar esta prueba, tratamos de navegar a la página de listado de usuarios y vemos como se nos redirige automáticamente a la página de Log In. Al no estar iniciados de sesión, no tenemos acceso.

Prueba 6: Búsqueda válida

Para esta prueba en primer lugar deberemos iniciar sesión en la aplicación, y mediante el PO_ListUsersView rellenamos el formulario de búsqueda para a continuación, comprobar que nos ha encontrado el usuario solicitado. Para ello se busca por texto en la página si existe el correo del usuario que debería dar como resultado.

```
@Test
	public void P06_BusUsrVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Comprobamos que estamos en la página correcta
		PO_ListUsersView.checkElement(driver, "id", "tituloLogin");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laPaca@gmail.com", "123456");
		//Comprobamos que estamos en la pagina de listado de usuarios
		PO_LogInView.checkElement(driver, "id", "tituloUsuarios");
		//Realizamos la busqueda
		PO_ListUsersView.fillForm(driver, "co");
		//Comprobamos que aparece el usuario en la pagina y solo el
		List<WebElement> elementos = PO_View.checkElement(driver,
"text", "elPaco-@hotmail.com");
		assertTrue(elementos.size()==1);
	}
```

Prueba 7: Búsqueda inválida

```
@Test
	public void P07_BusUsrInVal(){
		// Intentamos acceder al listado de usuarios sin logearnos
/user/list
		driver.navigate().to(URL+"/user/list?searchText=pe");
		//Comprobamos que se nos redirige a la pagina de inicio de
sesion
		PO_LogInView.checkElement(driver, "free", "//h2[contains(@id,
'tituloLogin')]");

	}
```

Metemos en la url la búsqueda deseada y vemos como, sin embargo, la aplicación nos redirige a la página de log in para que iniciemos sesión pues no estamos autenticados.

Prueba 8: Enviar solicitud de amistad válido.

En este caso de prueba en primer lugar deberemos seguir los pasos seguidos en las otras pruebas, presionar el login en el menú, iniciar sesión y comprobar que estamos en la página de listado de usuarios.

Una vez ahí, nos vamos hasta la última página y buscamos los botones de *"Enviar petición de Amistad"*. La id de estos botones siguen el patrón: *addButton + número*. Por lo que mediante una búsqueda podremos obtener todos los elementos de la página cuya id tenga de base "addButton". Una vez los tenemos en una lista presionamos el primero de ellos y guardamos la cantidad de botones que había.

Se ha dormido la aplicación durante un segundo antes de volver a buscar los botones para darle tiempo al navegador a recargar la página. Tras esto contamos el número de botones actuales y nos cercioramos de que, en efecto, ha bajado en una unidad.

```
public void P08_InvVal(){
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laPaca@gmail.com", "123456");
		// Comprobamos que entramos en la sección privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		// Esperamos a que se muestren los enlaces de paginación la
lista de usuarios
		List<WebElement> elementos = PO_View.checkElement(driver,
"free", "//a[contains(@class, 'page-link')]");
		// Nos vamos a la última página
		elementos.get(3).click();
		//Presionar sobre boton "Enviar peticion de amistad"
		elementos = PO_View.checkElement(driver, "id","addButton");
		elementos.get(0).click();
		int tamaño1 = elementos.size();
		//Comprobar que se han reducido el numero de botones
		try {
			Thread.sleep(1000); //Para darle tiempo a recargar la
pagina
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		elementos = PO_View.checkElement(driver, "id", "addButton");
		int tamaño2 = elementos.size();
		assertTrue(tamaño1==tamaño2+1);
	}
```

Prueba 9: Invitación de amistad inválida

```
@Test(expected=org.openqa.selenium.TimeoutException.class)
	public void P09_InvInVal() throws TimeoutException {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		// Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laPaca@gmail.com", "123456");
		// Comprobamos que entramos en la sección privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		// Esperamos a que se muestren los enlaces de paginación la
lista de notas
		List<WebElement> elementos = PO_View.checkElement(driver,
"free", "//a[contains(@class, 'page-link')]");
		// Nos vamos a la última página
		elementos.get(3).click();
		// Comprobamos que no existe el boton
		PO_View.checkElement(driver, "id", "addButton6");
	}
```

Para esta prueba seguiremos los pasos anteriores e intentaremos buscar el botón que se ha eliminado, de id addButton6. Como no existe, saltará una excepción que controlaremos.

Prueba 10: Listar invitaciones de un usuario

```
public void P10_LisInvVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laVero@gmail.com", "123456");
		//Comprobamos que estamos en la página privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		//Vamos a la vista peticiones de amistad -
href="/request/list"
		List<WebElement> elementos = PO_View.checkElement(driver,
"free", "//a[contains(@href,'request/list')]");
		elementos.get(0).click();
		//Comprobamos que estamos en la vista de peticiones de
amistad
		PO_ListUsersView.checkElement(driver, "id",
"tituloPeticiones");
		//Comprobar tamaño de la tabla para ver que hay peticiones
		List<WebElement> peticiones =
SeleniumUtils.EsperaCargaPagina(driver, "free", "//tbody/tr",
				PO_View.getTimeout());
		assertTrue(peticiones.size() > 0);
	}
```

Para realizar este test, seguimos los pasos habituales para iniciar sesión, a continuación, vamos y comprobamos que estamos en la vista de peticiones de amistad, y que su tabla tiene contenido.

Prueba 11: Aceptar una petición

Seguimos todos los pasos del caso anterior, para a continuación buscar por id un botón para aceptar la petición. Todo ello habiendo guardado previamente el tamaño de la tabla.

```
//Comprobar tamaño de la tabla
		List<WebElement> peticiones =
SeleniumUtils.EsperaCargaPagina(driver, "free", "//tbody/tr",
				PO_View.getTimeout());
		assertTrue(peticiones.size() > 0);
		int tamaño = peticiones.size();
		//Presionar sobre boton "Aceptar peticion de amistad"
		elementos = PO_View.checkElement(driver, "id","addButton1");
		elementos.get(0).click();
		//Ver que el tamaño de la tabla ha disminuido en uno
		try {
			Thread.sleep(1000); //Para darle tiempo a recargar la
pagina
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		peticiones = SeleniumUtils.EsperaCargaPagina(driver, "free",
"//tbody/tr", PO_View.getTimeout());
		assertTrue(peticiones.size()==tamaño-1);
```

A continuación, dormimos nuevamente el hilo un segundo para darle tiempo al navegador, y comprobamos que el nuevo tamaño de la tabla ha descendido en una unidad.

Prueba 12: Listar los amigos de un usuario.

```
@Test
	public void P12_ListAmiVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laVero@gmail.com", "123456");
		//Comprobamos que estamos en la página privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		//Vamos a la vista peticiones de amistad -
href="/request/list"
		List<WebElement> elementos = PO_View.checkElement(driver,
"free", "//a[contains(@href,'user/friendsList')]");
		elementos.get(0).click();
		//Comprobamos que estamos en la vista de listas de amigos
		PO_ListUsersView.checkElement(driver, "id", "tituloAmigos");
		//Comprobar tamaño de la tabla
		List<WebElement> peticiones =
SeleniumUtils.EsperaCargaPagina(driver, "free", "//tbody/tr",
				PO_View.getTimeout());
		assertTrue(peticiones.size() > 0);
	}
```

Procedemos iniciando sesión para a continuación dirigirnos a la vista de amigos y comprobamos que en efecto hemos llegado a la página adecuada. A continuación, comprobamos que se listan los amigos.

Prueba 13: Crear una publicación con datos válidos sin foto

```
@Test
	public void P13_PubVal_PubFot2Val() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laPaca@gmail.com", "123456");
		//Comprobamos que estamos en la página privada
		PO_ListUsersView.checkElement(driver, "id", "tituloUsuarios");
		//Vamos a la vista crear publicaciones
		List<WebElement> elementos = PO_View.checkElement(driver, "free",
"//a[contains(@href,'post/create')]");
		elementos.get(0).click();
		//Comprobamos que estamos en la vista de crear publicaciones
		PO_ListUsersView.checkElement(driver, "id", "crearPubli");
		//Rellenamos el formulario
		PO_CreatePostView.fillForm(driver, "La publi de la Paca", "Hola
mundo");
		//Comprobamos que volvemos a home
		PO_View.checkElement(driver, "text", "PACA");
	}
```

De nuevo hemos juntado dos casos de prueba: creación de una publicación válida y sin foto. Para ello iniciamos sesión y vamos a la vista de creación de peticiones. Dejamos que sea su PO quien rellene el formulario y comprobamos que tras crear la publicación con éxito se nos redirecciona a home.

Prueba 14: Listado de publicaciones

```
@Test
	public void P14_LisPubVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laPaca@gmail.com", "123456");
		//Comprobamos que estamos en la página privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		//Vamos a la vista ver publicaciones
		List<WebElement> elementos = PO_View.checkElement(driver,
"free", "//a[contains(@href,'post/list')]");
		elementos.get(0).click();
		//Comprobamos que estamos en la vista de ver publicaciones
		PO_ListUsersView.checkElement(driver, "id", "listarPubli");
		//Y comprobamos que se ha añadido la publicacion que acabamos
de introducir
		elementos = PO_View.checkElement(driver,
"free","//td[contains(text(), 'La publi de la Paca')]");
		assertTrue(elementos.size()==1);
	}
```

Iniciamos sesión y vamos y comprobamos que estamos en la vista de publicaciones. Una vez ahí comprobamos que se ha añadido la publicación que acabamos de crear.

Prueba 15: Listar publicaciones de un amigo

Para esta prueba iniciamos sesión y vamos a la vista de amigos. Una vez ahí, comprobamos que estamos en la vista adecuada y que existen amigos. Mediante una búsqueda, buscamos todos los enlaces de los amigos y presionamos el primero de ellos. A continuación, comprobamos que estamos en la vista adecuada.

```
@Test
	public void P15_ListAmiVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laVero@gmail.com", "123456");
		//Comprobamos que estamos en la página privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		//Vamos a la vista ver amigos - href="/user/friendsList"
		List<WebElement> elementos = PO_View.checkElement(driver,
"free", "//a[contains(@href,'user/friendsList')]");
		elementos.get(0).click();
		//Comprobamos que estamos en la vista de listas de amigos
		PO_ListUsersView.checkElement(driver, "id", "tituloAmigos");
		//Comprobamos que hay amigos
		List<WebElement> peticiones =
SeleniumUtils.EsperaCargaPagina(driver, "free", "//tbody/tr",
				PO_View.getTimeout());
		assertTrue(peticiones.size() > 0);
		//Presionamos el enlace del amigo
		elementos = PO_View.checkElement(driver, "id",
"enlaceAmigo");
		elementos.get(0).click();
		//Comprobamos que estamos en publicaciones publicaciones
listarPubli
		elementos = PO_View.checkElement(driver, "id",
"listarPubli");
	}
```

Prueba 16: Listar publicaciones Inválido.

```
@Test
	public void P16_LisPubAmiInVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laPaca@gmail.com", "123456");
		//Intentamos acceder a la url de un usuario que no es amigo
/post/7/list
		driver.navigate().to(URL+"/post/7/list");
		//Vemos que redirige a la pagina de listado de amigos
		PO_ListUsersView.checkElement(driver, "id", "tituloAmigos");
//Redirige a tus amigos
	}
```

Al intentar acceder por URL vemos que se nos redirige a la página de los amigos del usuario en sesión al no pertenecer a su lista de amigos.

Prueba 17: Publicación con foto

Seguimos los pasos realizados a la hora de crear publicaciones, pero esta vez rellenamos de otra manera el formulario.

```
static public void fillForm(WebDriver driver, String titulop, String
textop, String imagenp) {
		String path =
System.getProperty("user.dir")+"\\src\\"+imagenp;
		WebElement image = driver.findElement(By.name("image"));
		image.sendKeys(path);
		fillForm(driver, titulop, textop);
	}
```

En esta ocasión, en el PO se rellena también con una imagen cuyo nombre se pasa por parámetro.

Una vez rellenado el formulario, comprobaremos que se ha creado la publicación yendo a su vista y comprobando su existencia.

Prueba 18: Administrador y listar usuarios

De nuevo, juntamos dos pruebas accediendo como administrador y listando los usuarios de la aplicación.

```
@Test
	public void P18_AdInVal_AdLisUsrVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "elPaco-@hotmail.com",
"123456");
		// Comprobamos que entramos en la sección privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		//Y que sus opciones son eliminar usuario
		List<WebElement> elementos = PO_View.checkElement(driver,
"free", "//a[contains(@href,'/user/delete/')]");
		assertTrue(elementos.size()>0);
	}
```

Una vez iniciada la sesión como administrador, comprobamos que estamos en la página de listar usuarios. Para diferenciarlo de un usuario normal comprobamos que en esta página aparecen las opciones de eliminar usuario.

Prueba 19: Administrador datos inválidos

En esta ocasión comprobamos que tras meter datos de usuario normal, no aparecen botones de eliminar usuario si no de enviar peticiones de amistad.

```
	@Test
	public void P19_AdInInVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laMonse@gmail.com", "123456");
		// Comprobamos que entramos en la sección privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		//Y que sus opciones son enviar peticiones
		List<WebElement> elementos = PO_View.checkElement(driver,
"id", "addButton");
		assertTrue(elementos.size()>0);
	}
```

Prueba 20: Administrador elimina usuario

```
@Test
	public void P20_AdLisUsrVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "elPaco-@hotmail.com",
"123456");
		// Comprobamos que entramos en la sección privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		// Esperamos a que se muestren los enlaces de paginación la
lista de usuarios
		List<WebElement> elementos = PO_View.checkElement(driver,
"free", "//a[contains(@class, 'page-link')]");
		// Nos vamos a la última página
		elementos.get(3).click();
		//Almacenamos el tamaño de la tabla
		int tamaño = PO_View.checkElement(driver, "free",
"//tbody/tr").size();
		//Y eliminamos uno de los usuarios
		elementos = PO_View.checkElement(driver, "free",
"//a[contains(@href,'/user/delete/')]");
		elementos.get(0).click();
		//Nos vamos a la ultima pagina
		elementos = PO_View.checkElement(driver, "free",
"//a[contains(@class, 'page-link')]");
		elementos.get(3).click();
		//Y comprobamos que no existe ya pues el numero de filas ha
disminuido
		int tamaño2 = PO_View.checkElement(driver, "free",
"//tbody/tr").size();
		assertTrue(tamaño==tamaño2+1);
	}
```

Iniciamos sesión cómo administrador y vamos a la última página de usuarios. Tras ello, listamos todos los componentes que referencien a la url *'/user/delete/'*, que se corresponderá con los botones. Pulsamos el primero de ellos, habiendo guardado previamente el tamaño de la tabla.

Vamos de nuevo a la última página y comprobamos que el tamaño ha disminuido en una unidad.

Prueba 21: Eliminado vía URL sin permiso

```
	@Test
	public void P21_AdBorUsrInVal() {
		// Vamos al formulario de incio de sesion
		PO_HomeView.clickOption(driver, "login", "class", "btn btn-
primary");
		//Rellenamos el formulario
		PO_LogInView.fillForm(driver, "laMonse@gmail.com", "123456");
		// Comprobamos que entramos en la sección privada
		PO_ListUsersView.checkElement(driver, "id",
"tituloUsuarios");
		// Intentamos borrar un usuario
		driver.navigate().to(URL+"/user/delete/7");
		//Comprobamos que no tenemos acceso
		PO_View.checkElement(driver, "text", "Access is denied");
	}
```

Introducimos una url de eliminado con un usuario que no sea un administrador y comprobamos que salta la página de error.