



Universitat d'Alacant  
Universidad de Alicante

# **Metodologías Ágiles de Desarrollo de Software**

Sergio Baeza Carrasco

Álvaro Lario Sánchez

Eugenio Benito López

Ingeniería Informática Universidad de  
Alicante

# **Índice**

1. Sprint Backlog
2. Funcionalidades implementadas
3. Puesta en producción
4. Informe sobre la evolución del desarrollo
5. Informe sobre las sesiones de pair programming
6. Resultado de la retrospectiva

## 1. Sprint Backlog

### 014 Tarea con fecha y vista de calendario

#### *Historia de usuario*

Como usuario quiero crear las tareas con una fecha límite para realizarla y visualizarlas en un calendario

#### *Detalles*

- Las tareas contarán con un atributo que represente la fecha límite de cumplimiento
- El usuario tendrá una forma extra de ver sus tareas (a parte del modo 'lista' ya implementado), un calendario con las tareas puestas en la fecha límite de cumplimiento

#### *COS (Condiciones de satisfacción)*

- El usuario crea una tarea con una fecha límite de realización
- Poder ver la tarea creada en una vista de calendario

#### *Mockups*

No se desarrollaron mockups ya que no sabíamos si íbamos a ser capaces de implementar un calendario tal y como lo imaginábamos. Posteriormente entendimos que el mockup previo se trata la representación a mano alzada de lo que se desea implementar, por lo que sí deberíamos haber realizado el mockup.

### 015 Editar perfil de usuario

#### *Historia de usuario*

Como usuario quiero que se pueda editar el perfil

#### *Detalles*

Existirá un botón en el navegador al apartado 'Mi cuenta'

En esta vista se mostrará un formulario para cambiar los datos de la cuenta:

- Contraseña
- Nombre
- Fecha de nacimiento

#### *COS (Condiciones de satisfacción)*

Se comprueba que tras llenar de forma correcta el formulario de 'Mi cuenta' se actualizan los detalles del usuario.

## *Mockups*

<b>Usuario 1 Tareas Equipos Mis Equipos</b>
<b>Usuario</b> <input type="text"/>
<b>Correo</b> <input type="text"/>
<b>Contraseña</b> <input type="text"/>
<b>Guardar</b>

## **016 Administrador del equipo**

### *Historia de usuario*

Como usuario quiero que se cree la figura de propietario del equipo para que sea este el que maneje los ajustes del equipo, invite miembros, cree proyectos...

### *Detalles*

Cada equipo tendrá un administrador que será el creador del equipo y no podrá cambiarse

El administrador podrá:

- Editar el equipo
- Gestionar los miembros
- A la hora de crear un equipo, el usuario que lo crea se convertirá en el administrador y no podrá dejar de serlo.
- Será el encargado de gestionar el equipo, sus miembros, nombre... y podrá eliminarlo
- Un equipo solo podrá tener un administrador

### *COS (Condiciones de satisfacción)*

Cada equipo tiene su administrador y desde la vista de un administrador puede acceder a la pestaña de edición del equipo. Esta vista solo es visible para los usuarios

## *Mockups*

- Vista administrador de un equipo

usuario1	Tareas	Equipos
Detalles Equipo1		
Administrador: usuario1 Nombre: Equipo1 Integrantes: - usuario1 <a href="#">Eliminar</a> - usuario2 <a href="#">Eliminar</a>		
<a href="#">Ajustes</a> <a href="#">Eliminar equipo</a>		

- Vista sin ser administrador de un equipo

usuario1	Tareas	Equipos
Detalles Equipo1		
Administrador: usuario1 Nombre: Equipo1 Integrantes: - usuario1 - usuario2		

## 017 Añadir comentarios a los equipos

### *Historia de usuario*

Como usuario quiero en la pestaña del equipo poder ver los comentarios del equipo, donde sólo podrán comentar los usuarios del equipo.

### *Detalles*

- Será un comentario con un texto, un autor y una fecha
- Se situará en la vista de los detalles del equipo
- Los usuarios que sean administradores del equipo podrán eliminar comentarios
- Sólo podrán comentar los usuarios del equipo

### *COS (Condiciones de satisfacción)*

Cuando el usuario entre en la pestaña de un equipo, si está en el equipo podrá introducir un nuevo comentario, sí no es así, sólo verá los comentarios si el equipo es público. Además si es el dueño del equipo podrá eliminar cualquier comentario.

### *Mockups*

- Vista de un usuario que está unido al grupo y puede comentar

usuario1	Tareas	Equipos
Detalles Equipo1		
<p>Administrador: usuario1 Nombre: Equipo1 Integrantes: - usuario1 - usuario2</p>		
<p>Nuevo comentario:</p> <input type="text"/> <p>Enviar</p>		
<p>Comentarios:</p> <p>usuario1 (ADMIN) dijo: Tenemos que acabar antes de mañana - 10/12/2022</p> <p>usuario2 dijo: Me encanta - 12/12/2022</p>		

- Vista de un usuario que no es miembro del grupo y no puede comentar:

usuario1 Tareas Equipos
<p><b>Detalles Equipo1</b></p> <p>Administrador: usuario1 Nombre: Equipo1 Integrantes: - usuario1 - usuario2</p> <p>Comentarios:</p> <p>usuario1 (ADMIN) dijo: Tenemos que acabar antes de mañana - 10/12/2022</p> <p>usuario2 dijo: Me encanta - 12/12/2022</p>

- Vista del administrador, que puede eliminar los comentarios

usuario1 Tareas Equipos
<p><b>Detalles Equipo1</b></p> <p>Administrador: usuario1 Nombre: Equipo1 Integrantes: - usuario1      <a href="#">Eliminar</a> - usuario2      <a href="#">Eliminar</a></p> <p>Nuevo comentario:</p> <p><input type="text"/> <a href="#">Enviar</a></p> <p>Comentarios:</p> <p>usuario1 (ADMIN) dijo: Tenemos que acabar antes de mañana - 10/12/2022      <a href="#">Eliminar comentario</a></p> <p>usuario2 dijo: Me encanta - 12/12/2022      <a href="#">Eliminar comentario</a></p>

## 018 Tareas con estado de progreso y vista en tablero

### *Historia de usuario*

Como usuario quiero que las tareas se puedan representar en un tablero estilo 'Kanban' con diferentes etiquetas o columnas que muestren el estado en el que se encuentra dicha tarea. Por ejemplo, si no ha comenzado o si ha finalizado.

### *Detalles*

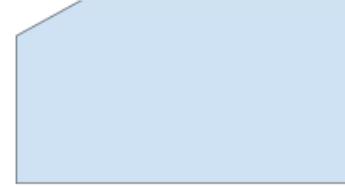
- Las tareas contarán con un atributo que represente el estado en el que se encuentra la tarea. Por ejemplo, pueden haber tareas que estén en progreso, tareas completadas y tareas pendientes. Este atributo permitirá al usuario visualizar de manera rápida el estado de cada tarea dentro del tablero Kanban.
- La vista del tablero Kanban se mostrará en una pantalla o interfaz gráfica, permitiendo al usuario ver todas las tareas en una sola vista. Las tareas se representarán mediante tarjetas, que se organizan en diferentes columnas según su estado. Por ejemplo, las tareas en progreso se mostrarán en una columna, las tareas completadas en otra y las tareas pendientes en otra.
- El usuario podrá modificar las tareas dentro del tablero Kanban mediante una interfaz de arrastrar y soltar. Esto significa que, para cambiar el estado de una tarea, el usuario solo tendrá que pulsar en la tarjeta que representa la tarea y arrastrarla a la columna que corresponda. De esta manera, el usuario podrá controlar y organizar las tareas de manera sencilla y visualmente atractiva.

### *COS (Condiciones de satisfacción)*

- Cuando un usuario crea una tarea no tendrá posibilidad de definir el estado. Por defecto será "To Do"
- El estado de la tarea puede ser "To Do", "Work In Progress" o "Done".
- El usuario puede modificar el estado de la tarea desde la vista en lista de tareas o desde la página de detalles de la tarea.
- Cuando una tarea se marque como terminada, si tiene fecha límite esta desaparecerá.
- Los atributos de una tarea que se encuentre en estado "Done" son inmutables a excepción del propio atributo "estado"

En resumen, el proceso de creación de una tarea se basa en la definición del estado de la tarea por parte del usuario, y en la posibilidad de modificar este estado en cualquier momento a través del tablero Kanban o de la vista total de tareas.

### *Mockups*

To Do	Work In Progress	Done
		
		

## **019 Pestaña de 'Mis equipos'**

### *Historia de usuario*

Como usuario quiero tener una vista en la que poder consultar los equipos a los que pertenezco.

### *Detalles*

- En la barra de menú aparecerá una opción 'Mis equipos'

### *COS (Condiciones de satisfacción)*

- Cuando el usuario pinche en la opción descrita anteriormente le llevará a un listado de los equipos a los que pertenece.

### *Mockups*

**Usuario 1 Tareas Equipos Mis Equipos**

**Mis Equipos:**

- |           |          |
|-----------|----------|
| - Equipo1 | Detalles |
| - Equipo2 | Detalles |
| - Equipo3 | Detalles |

**019 Añadir equipos a los que administras a la pestaña de 'Mis equipos'**

*Historia de usuario*

Como usuario quiero que se muestre un listado de los equipos que administro.

*Detalles*

En el apartado 'Mis equipos' se mostrará un listado de los equipos a los que pertenezco.

En el apartado 'Mis equipos' se mostrará un listado de los equipos que administro.

*COS (Condiciones de satisfacción)*

Cuando el usuario pinche en la pestaña 'Mis equipos' ubicada en la barra de navegación se mostrarán dos listados, uno de ellos con los equipos a los que pertenece si existen, en el caso contrario no mostrará ninguno. Y otro listado con los equipos que administro si existen, en el caso contrario no mostrará ninguno.

*Mockups*

<b>Usuario 1 Tareas Equipos Mis Equipos</b>	
<b>Mis Equipos:</b>	
- Equipo1	Detalles
- Equipo2	Detalles
- Equipo3	Detalles
<b>Administrados:</b>	
- Equipo1	Detalles
- Equipo2	Detalles
- Equipo3	Detalles

## 020 Recuperar contraseña por correo electrónico

### *Historia de usuario*

Como usuario quiero poder recuperar mi contraseña en caso de olvido

### *Detalles*

Existirá un botón de '¿Has olvidado la contraseña?' en la página de login

Se solicitará la contraseña al usuario y, en caso de ser un usuario registrado, se le enviará un correo con un enlace para restablecer la contraseña.

### *COS (Condiciones de satisfacción)*

El usuario recibe el correo, accede al link que le llega y es capaz de restablecer su contraseña

### *Mockups*

[Login](#) [Registro](#) [About](#)

**Registro:**

Usuario:

Email:

Contraseña:

Fecha\_Nac:

[¿Ha olvidado la contraseña?](#)

[Atras](#)

[Registrar](#)

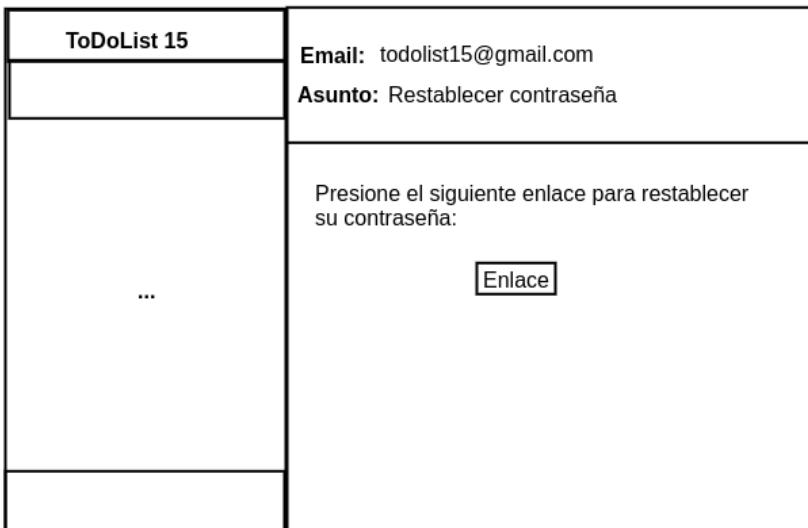
[Login](#) [Registro](#) [About](#)

**Restablecer contraseña:**

Correo Electrónico:

[Enviar](#)

## GMAIL



The image shows a password reset form. At the top, there are navigation links: 'Login', 'Registro', and 'About'. Below them, the heading 'Restablecer contraseña:' is displayed. A text input field is labeled 'Nueva contraseña:' followed by a 'CAMBIAR' button.

Login Registro About

**Restablecer contraseña:**

Nueva contraseña:

**CAMBIAR**

## 021 Equipos públicos o privados

### *Historia de usuario*

Como usuario quiero que los equipos de la aplicación tengan un estado público o privado. Los privados no son visibles para los usuarios comunes, y funcionan mediante invitaciones.

## *Detalles*

- Un equipo con estado público es aquel que puede ser visto y unido por cualquier usuario de la aplicación.
- Un equipo con estado privado es aquel que sólo es visible para los miembros del equipo y no para el resto de usuarios. El creador del equipo tiene la opción de invitar a otros usuarios a unirse al equipo, pero estos usuarios sólo podrán unirse si aceptan la invitación.
- El estado de un equipo puede cambiar en cualquier momento, dependiendo de la decisión del creador del equipo. Si el creador decide cambiar el estado de un equipo de privado a público, el equipo se volverá visible para todos los usuarios de la plataforma y estos podrán unirse al equipo. Si el creador decide cambiar el estado de un equipo de público a privado, el equipo se volverá invisible para los usuarios de la plataforma y sólo podrán unirse a él si son invitados por el creador.
- Los usuarios que han sido invitados a unirse a un equipo privado podrán acceder desde la barra de menú

## *COS (Condiciones de satisfacción)*

- En la pestaña de Equipos se mostrarán solo los públicos, en los privados existirá una pestaña para enviar invitaciones.
- Como usuario logeado podré ver las invitaciones que tengo para unirme a equipos privados.

## *Mockups*

En esta historia de usuario no se realizaron mockups previos porque no se tenía una idea clara de cómo se podría implementar la el front de esta funcionalidad (pese a tener muy claro cómo funcionará el sistema de invitaciones) Las decisiones de contar con una campana de notificaciones en la barra de navegación se tomaron durante el desarrollo de la funcionalidad.

## 022 El equipo puede contener proyectos que a su vez contienen tareas

### *Historia de usuario*

Como usuario administrador de un equipo puedo crear proyectos que han de desempeñar los usuarios miembros del equipo, a su vez estos usuarios podrán crear tareas. Como usuario de un equipo, puedo crear tareas en los proyectos de los que formo parte, así como desempeñar algunas.

### *Detalles*

- Al crear un equipo dispondremos de un botón para crear proyectos.
- En un menú saldrán la totalidad de los proyectos existentes en el equipo.
- Podremos acceder a la lista de tareas de un proyecto pulsando en él.
- Podremos crear tareas, así como eliminarlas, modificarlas y asignarlas.
- Tendremos una vista con un tablero Kanban para visualizar el estado de las tareas.
- También tendremos una pestaña donde podremos visualizar un calendario y si le corresponden tareas estarán representadas.

### *COS (Condiciones de satisfacción)*

Cuando el usuario entre en la pestaña de un equipo, si está en el equipo podrá introducir un nuevo comentario, en cambio si no está unido sólo verá los comentarios si el equipo es público. Además si es el dueño del equipo podrá eliminar cualquier comentario.

### *Mockups*

- Vista de un administrador que puede crear proyectos

usuario1 Tareas Equipos

Detalles Equipo1

Administrador: usuario1  
Nombre: Equipo1  
Integrantes:  
- usuario1    **Eliminar**  
- usuario2    **Eliminar**

**Crear proyecto**

**Ajustes**    **Eliminar equipo**

- Vista de un equipo con proyectos creados

usuario1 Tareas Equipos

Detalles Equipo1

Administrador: usuario1  
Nombre: Equipo1  
Integrantes:  
- usuario1  
- usuario2

Proyecto  
pagina web    **ver más**

- Página de crear proyectos

usuario1 Tareas Equipos

Crear proyecto para el equipo 1

Nombre del proyecto

**Crear**

- Vista de proyecto con posibilidad de añadir tareas (sólo para usuarios del equipo)

The mockup displays a user interface for managing tasks within a team project. At the top, it says "usuario1 Tareas Equipos". Below that is a form with a text input field labeled "Nombre de la tarea" and a "Crear" button. Underneath is a Kanban board with three columns: "To do", "In progress", and "Done". The "To do" column has one card labeled "esquiar". The "In progress" column has two cards, both labeled "esquiar". The "Done" column is empty.

## 023 Las tareas de un proyecto de equipo pueden ser asociadas a usuarios

### *Historia de usuario*

Como usuario podrá asociarse una tarea creada dentro de un proyecto de equipo y podrá visualizar las tareas que están asociadas a los diferentes miembros de equipo que pertenezcan al proyecto.

### *Detalles*

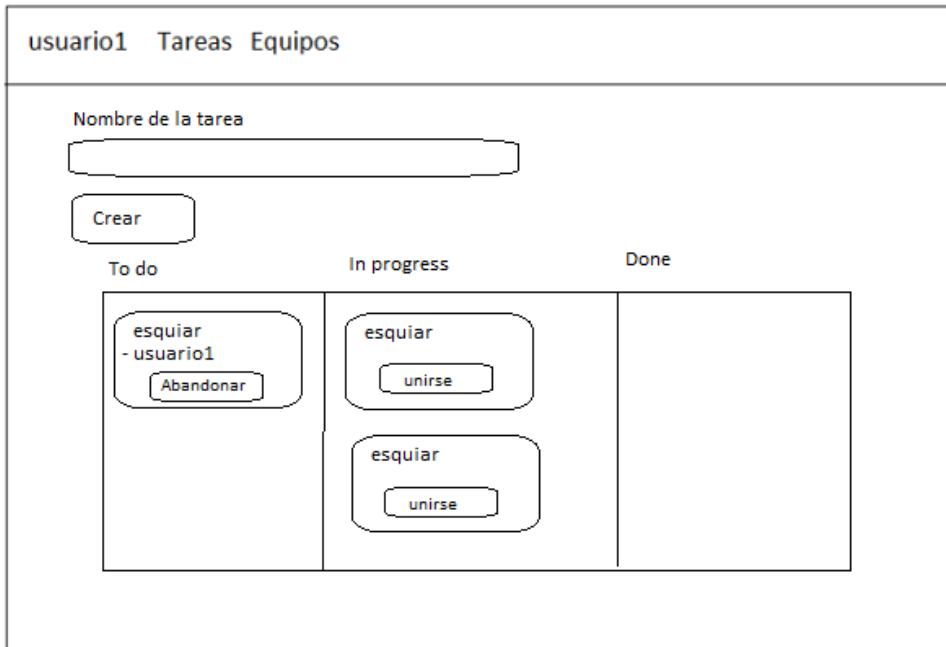
- Las tareas contarán con un atributo que representen los usuarios que la han de desempeñar.
- El usuario mediante un botón podrá solicitar el desempeño de la misma.
- El usuario mediante un botón podrá solicitar que se le desvincule de la misma.
- El usuario podrá consultar si una tarea está vinculada a un usuario y en tal caso ver el usuario a la que se le ha vinculado.

### *COS (Condiciones de satisfacción)*

- El usuario al hacer clic en el botón asignar, se asocia a tal tarea.
- El usuario al hacer clic en el botón desasignar, se desasocia de tal tarea.
- El usuario visualiza las tareas asignadas a los usuarios desde el listado de tareas del proyecto.

### *Mockups*

- Vista del tablero de un proyecto donde se puede observar que el usuario1 puede unirse o abandonar tareas



## 024 Recordatorio de tareas "hoy"

### *Historia de usuario*

Como usuario quiero que la aplicación cree un sistema de recordatorios de tareas programadas para hoy en el apartado de notificaciones de la aplicación

### *Detalles*

- Habrá una lista de tareas cuya fecha coincida con la actual.
- Se mostrarán en la barra de notificaciones

### *COS (Condiciones de satisfacción)*

El usuario logueado puede ver si tiene o no en la pestaña de notificaciones recordatorios de tareas para hoy

## *Mockups*

- Vista de un usuario logueado con una tarea que vence hoy:

usuario1	Tareas	Equipos	
	<p>Tareas hoy:</p> <p>- Salir a correr</p>		

## **025 Notificaciones por correo electrónico**

### *Historia de usuario*

Como usuario quiero que se envíen notificaciones por correo electrónico a los usuarios de la aplicación.

### *Detalles*

La aplicación tendrá un correo para enviar mails a los usuarios.

Se creará un servicio de enviar correos para:

- Un usuario se da de alta.
- Un usuario quiere recuperar la contraseña.
- Un usuario recibe una invitación del equipo.
- Un usuario se une al equipo.

### *COS (Condiciones de satisfacción)*

Cuando se realice una de las 4 acciones mencionadas anteriormente se debe de recibir un correo con el asunto y cuerpo deseado.

## Mockups

### GMAIL

<b>ToDoList 15</b>	<b>Email:</b> todolist15@gmail.com <b>Asunto:</b> Restablecer contraseña
...	Presione el siguiente enlace para restablecer su contraseña: <a href="#">Enlace</a>

### GMAIL

<b>ToDoList 15</b>	<b>Email:</b> todolist15@gmail.com <b>Asunto:</b> Usuario1 registrado
...	El usuario <b>Usuario1</b> se ha registrado con éxito. <a href="#">ACCEDER</a>

## GMAIL

<b>ToDoList 15</b>	<b>Email:</b> todolist15@gmail.com <b>Asunto:</b> Invitación a equipo
...	El usuario <b>Usuario1</b> quiere que formes parte de su equipo.
	<b>UNIRME</b>

## 2. Funcionalidades implementadas

### 014 Tarea con fecha y vista de calendario

Desde ahora las tareas contarán con una fecha límite opcional que por defecto se inicializará como un valor nulo.

Tarea.java

```
1 @Column(name = "fecha_limite")
2 @Temporal(TemporalType.DATE)
3 private Date fechaLimite;
4 ...
5 public Tarea(Usuario usuario, String titulo) {
6     this.usuario = usuario;
7     this.titulo = titulo;
8     this.fechaLimite = null;
9     usuario.getTareas().add(this);
10 }
```

Para contar con el menor acoplamiento posible, se añadió un método de servicio que añade una nueva tarea a partir de un DTO que contiene todos los detalles de la tarea en sí. De este modo, en caso de ampliar los atributos de la clase Tarea únicamente sería necesario modificar la clase encargada de encapsularlos y no necesitaremos añadir nuevos métodos de servicios.

### TareaData.java

```
1 public class TareaData {  
2     private String titulo;  
3     private Date fechaLimite;  
4  
5     public String getTitulo() {  
6         return titulo;  
7     }  
8     public void setTitulo(String titulo) {  
9         this.titulo = titulo;  
10    }  
11 }  
12  
13     public Date getFechaLimite() {  
14         return this.fechaLimite;  
15     }  
16  
17     public void setFechaLimite(Date fechaLimite) {  
18         this.fechaLimite = fechaLimite;  
19     }  
20 }
```

### TareaService.java

```
1     @Transactional  
2     public Tarea nuevaTareaUsuario(Long idUsuario, TareaData tareaDTO) {  
3         logger.debug("Añadiendo tarea " + tareaDTO.getTitulo() + " al usuario " + idUsuario);  
4         Usuario usuario = usuarioRepository.findById(idUsuario).orElse(null);  
5         if (usuario == null) {  
6             throw new TareaServiceException("Usuario " + idUsuario + " no existe al crear tarea "  
+ tareaDTO.getTitulo());  
7         }  
8  
9         Tarea tarea = new Tarea(usuario, tareaDTO.getTitulo());  
10        tarea.setFechaLimite(tareaDTO.getFechaLimite());  
11        tareaRepository.save(tarea);  
12        return tarea;  
13    }
```

Para implementar la vista de calendario se utilizó la librería [FullCalendar](#), que nos otorga gran flexibilidad para integrarlo con la nuestra clase encargada de representar las tareas del usuario.

Para ello, fue necesario exprimir las posibilidades del motor de plantillas Thymeleaf y encontrar el modo de insertar sintaxis thymeleaf entre código Javascript para ser capaces de iterar en el modelo de listado de tareas que se inserta en la plantilla.

```
listaTareas.html

1 <script type="text/javascript">
2   document.addEventListener('DOMContentLoaded', function () {
3     var calendarEl = document.getElementById('calendario');
4     var calendar = new FullCalendar.Calendar(calendarEl, {
5       headerToolbar: {
6         left: "prev,next today",
7         center: "title",
8         right: "dayGridMonth,timeGridWeek,timeGridDay"
9       },
10      height: 800,
11      contentHeight: 780,
12      aspectRatio: 3,
13      initialView: 'dayGridMonth',
14      nowIndicator: true,
15    });
16
17    /*[# th:each="tarea : ${tareas}"]*/
18    console.log("[(${tarea.fechaLimite})]");
19    calendar.addEvent({
20      title: "[(${tarea.titulo})]",
21      start: "[(${tarea.fechaLimite})]",
22    })
23  /*[/]*/
24
25  calendar.updateSize();
26  calendar.render();
27 });
28 </script>
```

## 015 Editar perfil de usuario

Para implementar esta historia de usuario se han realizado las siguientes acciones:

- Se ha definido una nueva plantilla Thymeleaf encargada de solicitar los datos que deseamos modificar al usuario.
- Se han introducido dos nuevas rutas en el controlador del modelo usuario:
  - GET /account, destinado a enviar a la plantilla los datos del usuario una vez se ha realizado su búsqueda.
  - POST /account, destinado a guardar los cambios realizados sobre el usuario en la base de datos.

#### UserController.java

```
1  @GetMapping("/account")
2  public String Account(Model model){
3      Long idUser = managerUserSession.usuarioLogeado();
4      if (idUser != null) {
5          Usuario user = usuarioService.findById(idUser);
6          model.addAttribute("usuario", user);
7      }
8      else{
9          throw new UsuarioNotFoundException();
10     }
11
12     return "account";
13 }
14
15 @PostMapping("/account")
16 public String edit(Usuario user, Model model) {
17     Usuario usuario = usuarioService.findByEmail(user.getEmail());
18     usuario.setPassword(user.getPassword());
19     usuario.setNombre(user.getNombre());
20     usuarioService.editar(usuario);
21
22     return "redirect:/account";
23 }
```

- Se ha definido un nuevo método dentro de la capa servicio del usuario que se encarga de editar los datos del usuario, de esta manera cuando la plantilla recibe los datos del usuario que desea modificar llama al siguiente método encargado de guardar los datos en la base de datos.

#### UsuarioService.java

```
1  @Transactional
2  public Usuario editar(Usuario usuario) {
3      Optional<Usuario> usuarioBD =
4          usuarioRepository.findByEmail(usuario.getEmail());
5      if (usuarioBD.isPresent())
6          if (usuario.getPassword() == null){
7              throw new UsuarioServiceException("El usuario no tiene password");
8          } else if (usuario.getNombre() == null) {
9              throw new UsuarioServiceException("El usuario no tiene nombre");
10         }
11         else{
12             return usuarioRepository.save(usuario);
13         }
14     else throw new UsuarioServiceException("El usuario no existe en la base de
15         datos");
16 }
```

- Hemos definido una serie de test cuya funcionalidad es probar que efectivamente funciona el servicio destinado a editar los datos introducidos por la plantilla.

```
UsuarioServiceTest.java
```

```

1 @Test
2     public void servicioEditarUsuario() {
3         Usuario usuario = new Usuario("usuario.prueba2@gmail.com");
4         usuario.setNombre("Alvaro");
5         usuario.setFechaNacimiento(new Date(1999,02,27));
6         usuario.setPassword("12345678");
7         usuarioService.registrar(usuario);
8         Usuario usuarioBaseDatos =
9             usuarioService.findByEmail("usuario.prueba2@gmail.com");
10        assertThat(usuarioBaseDatos).isNotNull();
11        assertThat(usuarioBaseDatos.getPassword()).isEqualTo(usuario.getPassword());
12        usuario.setPassword("123");
13        usuarioService.editar(usuario);
14        usuarioBaseDatos = usuarioService.findByEmail("usuario.prueba2@gmail.com");
15        assertThat(usuarioBaseDatos.getPassword()).isEqualTo(usuario.getPassword());
16    }

```

## Test 1

```
@Test
public void servicioEditarUsuario2() {
    Usuario usuario = new Usuario("usuario.prueba2@gmail.com");
    usuario.setNombre("Alvaro");
    usuario.setFechaNacimiento(new Date(1999,02,27));
    usuario.setPassword("12345678");
    usuarioService.registrar(usuario);
    Usuario usuarioBaseDatos =
        usuarioService.findByEmail("usuario.prueba2@gmail.com");
    assertThat(usuarioBaseDatos).isNotNull();
    assertThat(usuarioBaseDatos.getPassword()).isEqualTo(usuario.getPassword());
    usuario.setPassword("123");
    usuario.setNombre("Alberto");
    usuarioService.editar(usuario);
    usuarioBaseDatos = usuarioService.findByEmail("usuario.prueba2@gmail.com");
    assertThat(usuarioBaseDatos.getPassword()).isEqualTo(usuario.getPassword());
    assertThat(usuarioBaseDatos.getNombre()).isEqualTo(usuario.getNombre());
}
```

## Test 2

### 016 Administrador del equipo

Para implementar esta historia de usuario se han realizado las siguientes acciones:

- Se ha añadido a la clase **Equipo** un atributo de administrador que establece una relación de un equipo contiene un administrador. Por otra parte, un usuario puede administrar varios equipos.
- La clase EquipoService ahora implementa un nuevo método que recibe también el usuario que será el administrador. Este será pasado por el controlador con el usuario que está logueado en ese instante.

```

● ● ●

1 @Transactional
2     public Equipo crearEquipo(String nombre, String descripcion, Usuario admin){
3         if(nombre == "") throw new EquipoServiceException("El nombre del equipo no puede estar
4             vacio");
5         Equipo e = new Equipo(nombre);
6         e.setDescripcion(descripcion);
7         e.setAdmin(admin);
8         equipoRepository.save(e);
9         return e;
10    }

```

- A la hora de **proteger** las acciones de editar, eliminar... se ha modificado el middleware que anteriormente servía para los administradores generales permitiendo ahora el paso de los usuarios que son administradores.

```

● ● ●

1 private void checkAdminUserLogged(Equipo e) {
2     if (!managerUserSession.isUsuarioLogeado())
3         throw new UsuarioNoLogeadoException();
4     Long idUser = managerUserSession.usuarioLogeado();
5     Usuario u = usuarioService.findById(idUser);
6     Long idAdmin = new Long(-1);
7     if(e.getAdmin() != null) idAdmin = e.getAdmin().getId();
8     if(!u.isAdmin() && idUser != idAdmin){
9         throw new UsuarioNoAdminException();
10    }
11 }

```

## 017 Comentarios equipo

Para implementar esta historia de usuario se han realizado las siguientes acciones

- Se ha creado el modelo **ComentarioEquipo**. Este modelo está relacionado con un equipo y un usuario. Tiene un string que pertenece al comentario, y una fecha de publicación.
- Se ha creado la relación de que un **Equipo** tiene muchos **ComentarioEquipo** representando así a los comentarios de un equipo.
- Se ha creado un servicio llamado **ComentarioEquipoService** con los métodos para crear y eliminar un comentario.
- Se ha creado el controlador **ComentarioEquipoController** con las siguientes rutas:
  - DELETE /equipos/{id}/comentarios/{comentarioId} - Elimina el comentario
  - POST /equipos/{id}/comentarios

En ambos casos se pasa por el middleware que revisa si el usuario se ha unido al equipo.



```

1 @Transactional
2     private void userJoinedTeam(Equipo e) {
3         if (!managerUserSession.isUsuarioLogeado())
4             throw new UsuarioNoLogeadoException();
5         Long idUser = managerUserSession.usuarioLogeado();
6         Usuario u = usuarioService.findById(idUser);
7         Long idAdmin = new Long(-1);
8         if(e.getAdmin() != null) idAdmin = e.getAdmin().getId();
9         if(!equipoService.usuarioPerteneceEquipo(e, u) && idUser != idAdmin){
10             throw new UsuarioNotJoinedTeamException();
11         }
12     }

```

- Se han creado las excepciones **ComentarioNotFoundException** por si se intenta borrar un comentario que no existe y **UsuarioNotJoinedTeamException** por si la acción la lleva a cabo un usuario que no está unido en el equipo.
- En la vista de **detallesEquipo.html** se muestra el formulario y los comentarios. Para comprobar que un usuario pertenece al equipo se utiliza el método de Java contains.

## 018 Tareas con estado de progreso y vista en tablero

Las tareas desde ahora contarán con uno de los siguientes estados posibles: **To Do**, **In Progress** y **Done**. Representados por un objeto enumerado.

Status.java

```
1 public enum Status {  
2     TODO, IN_PROGRESS, DONE  
3 }
```

Cuando se crea una nueva tarea, el estado siempre será **To Do**.

Tarea.java

```
1 public Tarea(Usuario usuario, String titulo) {  
2     this.usuario = usuario;  
3     this.titulo = titulo;  
4     this.fechaLimite = null;  
5     this.status = Status.TODO;  
6     usuario.getTareas().add(this);  
7 }
```

Cuando se modifica el estado de una tarea a **Done**, la fecha límite de esta pasa a ser null; ya que se entiende que la información sobre la fecha límite de una tarea es irrelevante cuando la tarea se ha hecho.

Tarea.java

```
1 public void changeStatus(Status status) {  
2  
3     if (status == Status.DONE)  
4         this.fechaLimite = null;  
5  
6     this.status = status;  
7 }
```

La forma de representar el estado de las tareas es mediante un tablero tipo “kanban” que, al igual que en la vista de calendario, nos obligaba a insertar sintaxis thymeleaf en código Javascript para poder insertar las tareas en el tablero.

### listaTareas.html

```
1 <script>
2 ...
3
4 document.addEventListener('DOMContentLoaded', function () {
5   /*[# th:each="tarea : ${tareas}"]*/
6   console.log("Añadiendo tareas al board");
7   KanbanTest.addElement("[(${tarea.status})]", {
8     id: "[(${tarea.id})]",
9     title: "[(${tarea.titulo})]"
10  });
11 /*[/]*/
12 });
13 </script>
```

La forma de modificar el estado de las tareas es mediante *drag and drop* para que la experiencia de usuario sea la más dinámica posible. Por lo que para modificar el estado, se implementó un controller *REST*.

### TareaController.java

```
1 @PatchMapping("/tareas/{id}")
2 @ResponseBody
3 public String cambiarEstadoTarea(@PathVariable(value="id") Long idTarea,
4   @RequestBody String status) {
5   Tarea tarea = tareaService.findById(idTarea);
6   if (tarea == null) {
7     throw new TareaNotFoundException();
8   }
9   comprobarUsuarioLogeado(tarea.getUsuario().getId());
10
11  tareaService.changeStatus(idTarea, Status.valueOf(status));
12  return "";
13 }
```

## 019 Pestaña de 'Mis equipos'

Para implementar esta historia de usuario se han realizado las siguientes acciones:

- Se ha definido una nueva plantilla Thymeleaf encargada de visualizar los equipos a los que pertenece el usuario.
- Se ha modificado el controlador de Equipos creando un nuevo método que devuelve la plantilla anteriormente mencionada una vez recabados todos

los equipos a los que pertenece el usuario y almacenados en un hashset, una especie de lista compuesta por la entidad equipos.

- No ha hecho falta definir un servicio ya que hemos reutilizado algunos de los métodos existentes.

EquipoController.java

```
1 @GetMapping("/mis-equipos")
2     public String listadoMisEquipos(Model model, HttpSession session) {
3         isAnyUserLogged();
4         Long idUsuarioLogeado = managerUserSession.usuarioLogeado();
5         Usuario usuario = userService.findById(idUsuarioLogeado);
6         Set<Equipo> equipos = usuario.getEquipos();
7         model.addAttribute("usuario", usuario);
8         model.addAttribute("equipos", equipos);
9         return "misEquipos";
10    }
```

- Hemos definido una serie de test para probar la plantilla anteriormente mencionada, de manera que establecemos dos equipos y añadimos al usuario a uno de los equipos para ver si efectivamente dentro de la pestaña 'Mis equipos' aparece el equipo al que nos hemos suscrito. Posteriormente se añadiría al usuario al otro equipo y eliminaría de un equipo para finalizar la prueba.

### EquipoWebTest.java

```
1  @Test
2      public void getMisEquipos() throws Exception{
3
4          Equipo e1 = equipoService.crearEquipo("PruebaEquipo1");
5          Equipo e2 = equipoService.crearEquipo("PruebaEquipo2");
6
7
8          Usuario usuario = createUser();
9          when(managerUserSession.usuarioLogeado()).thenReturn(usuario.getId());
10         when(managerUserSession.isUserLogeado()).thenReturn(true);
11
12         this.mockMvc.perform(post("/equipos/" + e1.getId().toString() + "/usuarios/" +
13             usuario.getId().toString()));
14
15         this.mockMvc.perform(get("/equipos"))
16             .andExpect(content().string
17                 (allOf(containsString("Abandonar"))));
18
19         this.mockMvc.perform(get("/mis-equipos"))
20             .andExpect(content().string
21                 (allOf(containsString("PruebaEquipo1"))));
22
23         this.mockMvc.perform(post("/equipos/" + e2.getId().toString() + "/usuarios/" +
24             usuario.getId().toString()));
25
26         this.mockMvc.perform(get("/equipos"))
27             .andExpect(content().string
28                 (allOf(containsString("PruebaEquipo1"),
29                     containsString("PruebaEquipo2"))));
30
31         this.mockMvc.perform(delete("/equipos/" + e1.getId().toString() + "/usuarios/" +
32             usuario.getId()));
33         this.mockMvc.perform(get("/mis-equipos"))
34             .andExpect(content().string
35                 (allOf(containsString("PruebaEquipo2"))));
36     }
```

## 019 Añadir equipos a los que administras a la pestaña de 'Mis equipos'

Para implementar esta historia de usuario se han realizado las siguientes acciones:

- Se ha modificado la plantilla Thymeleaf de 'Mis equipos' encargada de visualizar los equipos a los que pertenece el usuario para que además muestre un listado de equipos a los que administrámos.
- Se ha modificado el método anteriormente creado para devolver equipos a los que un usuario pertenece para que además devuelva los equipos que administrámos en forma de lista.

- No ha hecho falta definir un servicio ya que hemos reutilizado algunos de los métodos existentes.

```
EquipoController.java
```

```
1 @GetMapping("/mis-equipos")
2     public String listadoMisEquipos(Model model, HttpSession session) {
3         isAnyUserLogged();
4         Long idUsuarioLogeado = managerUserSession.usuarioLogeado();
5         Usuario usuario = usuarioService.findById(idUsuarioLogeado);
6         Set<Equipo> equipos = usuario.getEquipos();
7         List<Equipo> equiposAdministrados = new ArrayList<>();
8         for(Equipo e: equipoService.findAllOrderedByName()){
9             if(e.getAdmin().equals(usuario)){
10                 equiposAdministrados.add(e);
11             }
12         }
13         model.addAttribute("usuario", usuario);
14         model.addAttribute("equipos", equipos);
15         model.addAttribute("equiposAdministrados", equiposAdministrados);
16         return "misEquipos";
17     }
```

- Hemos definido una serie de test para probar la plantilla anteriormente mencionada, de manera que establecemos dos equipos administrados por un usuario y revisamos que aparezcan dentro de la vista de equipos administrados, posteriormente desvinculamos al usuario de uno de los equipos para ver si la cadena devuelta no contiene el equipo del cual el usuario ya no es administrador.

### EquipoWebTest.java

```
1 @Test
2     public void getMisEquipos() throws Exception{
3
4         Equipo e1 = equipoService.crearEquipo("PruebaEquipo1");
5         Equipo e2 = equipoService.crearEquipo("PruebaEquipo2");
6
7
8         Usuario usuario = createUser();
9         Usuario usuario2 = new Usuario("user2@ua");
10        usuario2.setPassword("123");
11        usuario2.setIsAdmin(false);
12        usuario2.setBlocked(false);
13        usuarioService.registrar(usuario2);
14        Equipo e1 = equipoService.crearEquipo("PruebaEquipo1", "", usuario);
15        Equipo e2 = equipoService.crearEquipo("PruebaEquipo2", "", usuario2);
16
17        when(managerUserSession.usuarioLogeado()).thenReturn(usuario.getId());
18        when(managerUserSession.isUserLogeado()).thenReturn(true);
19
20 @@ -588,10 +592,32 @@ public void getMisEquipos() throws Exception{
21             (allOf(containsString("PruebaEquipo1"),
22                   containsString("PruebaEquipo2"))));
23
24         this.mockMvc.perform(delete("/equipos/" + e1.getId().toString() + "/usuarios/"
25             + usuario.getId()));
26         this.mockMvc.perform(delete("/equipos/" + e2.getId().toString() + "/usuarios/"
27             + usuario.getId()));
28         this.mockMvc.perform(get("/mis-equipos"))
29             .andExpect(content().string
30                 (allOf(containsString("PruebaEquipo2"))));
31             (allOf(containsString("PruebaEquipo1"))));
32     }
33
34     @Test
35     public void getMisEquiposAdministrados() throws Exception{
36
37         Usuario usuario = createUser();
38         Equipo e1 = equipoService.crearEquipo("PruebaEquipo1", "", usuario);
39         Equipo e2 = equipoService.crearEquipo("PruebaEquipo2", "", usuario);
40
41         when(managerUserSession.usuarioLogeado()).thenReturn(usuario.getId());
42         when(managerUserSession.isUserLogeado()).thenReturn(true);
43
44         this.mockMvc.perform(get("/equipos"))
45             .andExpect(content().string
46                 (allOf(containsString("PruebaEquipo1"),
47                   containsString("PruebaEquipo2"))));
48
49         this.mockMvc.perform(get("/mis-equipos"))
50             .andExpect(content().string
51                 (allOf(containsString("PruebaEquipo1"),
52                   containsString("PruebaEquipo2"))));
53     }
```

## 020 Recuperar contraseña por correo electrónico

Para implementar esta historia de usuario se han realizado las siguientes acciones:

- Se ha modificado la entidad usuario creando una atributo code que servirá para restablecer la contraseña vía email.

```
Usuario.java

1 private int code = new Random().ints(10000, 99999).findFirst().getAsInt();
2 ...
3 public int getCode() {
4     return code;
5 }
6
7 public void setCode(int code) {
8     this.code = code;
9 }
```

- Además hemos creado un nuevo servicio que se encargará de enviar emails desde una dirección que nosotros hemos establecido.

```
EnvioEmail.java

1 package madstodolist.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.mail.SimpleMailMessage;
5 import org.springframework.mail.javamail.JavaMailSender;
6 import org.springframework.stereotype.Service;
7
8 @Service
9 public class EnvioEmail {
10
11
12     //Importante hacer la inyección de dependencia de JavaMailSender:
13     @Autowired
14     private JavaMailSender mailSender;
15
16     //Pasamos por parametro: destinatario, asunto y el mensaje
17     public void sendEmail(String to, String subject, String content) {
18
19         SimpleMailMessage email = new SimpleMailMessage();
20
21         email.setTo(to);
22         email.setSubject(subject);
23         email.setText(content);
24
25         mailSender.send(email);
26     }
27 }
```

- Hemos definido una serie de rutas en el controlador de login:
  - GET /reset, destinada a devolver la plantilla en la que introducimos el email de la cuenta que deseamos recuperar.

```
LoginController.java
```

```

1 @GetMapping("/reset")
2     public String resetForm(Model model) {
3         model.addAttribute("loginData", new LoginData());
4         return "reset";
5     }
6

```

- POST /reset, destinada a almacenar los datos introducidos en la plantilla anterior y enviar el email llamando al servicio habilitado para tal efecto.

```
LoginController.java
```

```

7     @PostMapping("/reset")
8     public String resetSubmit(@ModelAttribute LoginData loginData, BindingResult
9                               result, Model model) {
10
11         if (result.hasErrors()) {
12             return "reset";
13         }
14
15         if (usuarioService.findByEmail(loginData.getMail()) == null) {
16             model.addAttribute("loginData", loginData);
17             model.addAttribute("error", "El usuario no existe");
18             return "reset";
19         }
20         SimpleMailMessage email = new SimpleMailMessage();
21         email.setTo(loginData.getMail());
22         email.setSubject("Restablecer contraseña");
23         StringBuffer body = new StringBuffer();
24         body.append("*****\n");
25         body.append("Correo electrónico AUTOMATIZADO - No responder a este
26         mensaje\n");
27         body.append("*****\n");
28         body.append("Fecha: ");
29         body.append(new Date());
30         body.append("\n");
31         body.append("_____\n");
32         body.append("Este es su enlace para restablecer la contraseña:");
33         body.append("\n");
34         body.append("http://localhost:8080/newPassword/user=" +
35         usuarioService.findByEmail(loginData.getMail()).getId() + "code=" +
36         usuarioService.findByEmail(loginData.getMail()).getCode());
37         body.append("\n");
38
39         body.append("_____
40         ");
41         email.setText(body.toString());
42         sender.send(email);
43         model.addAttribute("enviado", "Se ha enviado un correo electrónico para
44         restablecer la contraseña");
45         return "reset";
46     }

```

- GET /newPassword/user={id}code={code}, destinada a devolver la plantilla de recuperación de contraseña en la que se habilita un campo donde introducimos la nueva contraseña.  
 POST /newPassword/user={id}code={code}, destinada a almacenar la nueva contraseña y llamar al método editar del servicio usuario para que la guarde en la base de datos.

```

41
42     @GetMapping("/newPassword/user={id}code={code}")
43     public String newPasswordForm(@PathVariable(value="id") Long idUsuario,
44                                     @PathVariable(value="code") int code, Model model) {
45         model.addAttribute("loginData", new LoginData());
46         model.addAttribute("code", code);
47         model.addAttribute("idUsuario", idUsuario);
48         return "newPassword";
49     }
50

```

- POST /newPassword/user={id}code={code} destinada a almacenar la nueva contraseña y llamar al método editar del servicio usuario para que la guarde en la base de datos.

```

51     @PostMapping("/newPassword/user={id}code={code}")
52     public String newPasswordSubmit(@PathVariable(value="id") Long idUsuario,
53                                     @PathVariable(value="code") int code, @ModelAttribute LoginData loginData,
54                                     BindingResult result, Model model) {
55
56         if (result.hasErrors()) {
57             return "redirect:/newPassword/user=" + idUsuario + "code=" + code;
58         }
59         Usuario usuario = usuarioService.findById(idUsuario);
60         usuario.setPassword(loginData.getPassword());
61         usuario.setCode(new Random().ints(10000, 99999).findFirst().getAsInt());
62         usuarioService.editar(usuario);
63         return "redirect:/login";
64     }
65

```

- Hemos diseñado además una serie de test para probar que las plantillas funcionan correctamente con las rutas anteriormente mencionadas.

```

recuperarContraseñaWebTest.java
24 public class recuperarContraseñaWebTest {
25     @Autowired
26     private UsuarioService usuarioService;
27
28     @Autowired
29     private MockMvc mockMvc;
30
31     @Test
32     public void recuperarContraseña() throws Exception {
33
34         Usuario anaGarcia = new Usuario("ana.garcia@gmail.com");
35         anaGarcia.setNombre("Ana");
36         anaGarcia.setPassword("12345678");
37         usuarioService.registrar(anaGarcia);
38
39         this.mockMvc.perform(post("/login")
40                             .param("eMail", "ana.garcia@gmail.com")
41                             .param("password", "12345678"))
42             .andExpect(status().is3xxRedirection())
43             .andExpect(redirectedUrl("/usuarios/" + anaGarcia.getId() +
44 "/tareas"));
45
46         this.mockMvc.perform(post("/newPassword/user=" + anaGarcia.getId() + "code=" + anaGarcia.get
47 tCode())
48             .param("password", "1234"))
49             .andExpect(status().is3xxRedirection())
50             .andExpect(redirectedUrl("/login"));
51
52         this.mockMvc.perform(post("/login")
53                             .param("eMail", "ana.garcia@gmail.com")
54                             .param("password", "1234"))
55             .andExpect(status().is3xxRedirection())
56             .andExpect(redirectedUrl("/usuarios/" + anaGarcia.getId() +
57 "/tareas"));
58     }
59 }

```

## 021 Equipos públicos o privados

Se trata de una de las funcionalidades más complejas por la lógica que contiene la funcionalidad. A grandes rasgos, se resume en lo siguiente:

- Los equipos podrán ser públicos o privados
- Al momento de crear un equipo, por defecto se crea como público
- Únicamente el creador del equipo tendrá capacidad para modificar la visibilidad del equipo
- A un equipo privado, solo podrán unirse otros usuarios por medio de **invitaciones**
- Únicamente el creador del equipo podrá enviar invitaciones para unirse al equipo

- Para enviar una invitación, será necesario el correo electrónico del usuario que se desea enviar. El creador del equipo no recibirá *feedback* de si el correo introducido corresponde a un usuario que existe o no en el sistema. Ya que entendemos que así evitamos que mediante el envío de invitaciones se conozcan a todos los usuarios de la aplicación
- Una invitación solo se creará una vez. Es decir, si existe una invitación del Equipo X al usuario Y, aunque se le vuelva a invitar no se persistirá una nueva invitación
- Cuando un usuario acepta una invitación, automáticamente pasa a formar parte del equipo

La invitación se representa mediante la clase homónima que contiene el identificador del equipo que invita y el del usuario invitado.

Invitacion.java

```

1 @Entity
2 @Table(name="invitaciones")
3 public class Invitacion implements Serializable {
4
5     private static final long serialVersionUID = 1L;
6
7     @Id
8     @GeneratedValue(strategy = GenerationType.IDENTITY)
9     private Long id;
10    @NotNull
11    private Long idEquipo;
12    @NotNull
13    private Long idUsuario;
14
15    public Invitacion() {}
16
17    public Invitacion(Long idEquipo, Long idUsuario) {
18        this.idEquipo = idEquipo;
19        this.idUsuario = idUsuario;
20    }
21
22    public Long getUserId() {
23        return this.idUsuario;
24    }
25
26    public Long getEquipoId() {
27        return this.idEquipo;
28    }
29 }
```

Al igual que para crear una nueva tarea, para crear invitaciones el método de servicio encargado recibe un DTO con la información de dicha invitación.

InvitacionService.java

```
1 @Service
2 public class InvitacionService {
3
4     @Autowired
5     private InvitacionRepository invitacionRepository;
6
7     @Transactional
8     public void invitar(InvitacionData invitacionDTO) {
9         Invitacion invitacion = new
10            Invitacion(invitacionDTO.getIdEquipo(), invitacionDTO.getIdUsuario());
11            invitacionRepository.save(invitacion);
12    }
13 }
```

InvitacionData.java

```
1 public class InvitacionData {
2     private Long idEquipo;
3     private Long idUsuario;
4
5     public Long getIdEquipo() {
6         return idEquipo;
7     }
8     public void setIdEquipo(Long idEquipo) {
9         this.idEquipo = idEquipo;
10    }
11    public Long getIdUsuario() {
12        return idUsuario;
13    }
14    public void setIdUsuario(Long idUsuario) {
15        this.idUsuario = idUsuario;
16    }
17 }
```

Al igual que con el tablero kanban, la forma de aceptar o denegar una invitación es mediante un click a la invitación, que envía una petición http manejada por un controller *rest* para dinamizar al máximo la experiencia del usuario.

### InvitacionController.java

```
1 @PatchMapping("/invitacion/{id}")
2 @ResponseBody
3 public String aceptar(@PathVariable(value="id") Long idInvitacion) {
4     Invitacion invitacion = invitacionService.findById(idInvitacion);
5     comprobarUsuarioLogeado(invitacion.getUsuarioId());
6     invitacionService.aceptar(invitacion);
7     return "";
8 }
```

### Invitations.js

```
1 function aceptInvitacion(urlPatch) {
2     const url = 'http://localhost:8080' + urlPatch;
3
4     fetch(url, {
5         method: 'PATCH'
6     }).then((res) => {
7         location.reload();
8     })
9 }
```

## 022 El equipo puede contener proyectos que a su vez contienen tareas

Para implementar esta historia de usuario se han realizado las siguientes acciones:

- Se han creado 2 nuevos modelos: **Proyecto** y **TareaProyecto**. Empezando por **TareaProyecto**, tiene un nombre, un id y un status (que viene definido por el enumerado que usa el modelo Tarea). Esta TareaProyecto tiene un proyecto asignado. Por otro lado, el modelo proyecto contiene una **lista de TareaProyecto**. El proyecto viene definido por un id y un nombre. Un proyecto pertenece a un equipo. De esta forma las relaciones son las siguientes:

- Un **Equipo** puede tener ninguno o muchos **Proyectos**
- Un **Proyecto** pertenece a un **Equipo** y puede tener ninguna o muchas **TareaProyecto**
- Una **TareaProyecto** siempre pertenece a un proyecto.

- Una vez se han creado los modelos, y sus correspondientes **repositorios** se crean los servicios. La definición de los servicios es la siguiente:

```

1 class ProyectoService {
2     // Crea un proyecto a partir de un nombre y un equipo
3     public Proyecto crearProyecto(String nombre, Long equipoId);
4     // Obtiene un proyecto a partir de su identificador
5     public Proyecto getById(Long id);
6     // Elimina un proyecto según su identificador
7     public void eliminarProyecto(Long id);
8 }
9
10 class TareaProyectoService {
11     // Crea una tarea de un proyecto a partir de un nombre para la tarea y al proyecto al que
12     // pertenece
13     public TareaProyecto crearTareaProyectoService(String nombre, Long proyectoId);
14     // Encuentra una TareaProyecto por su id
15     public TareaProyecto findById(Long tareaId);
16     // Elimina una TareaProyecto por su id
17     public void eliminarTareaProyecto(Long tareaId);
18     // Cambia el estado de una tarea al que se le pasa por parámetro de la TareaProyecto con
19     // el id
20     public TareaProyecto cambiarEstado(Long tareaId, Status status);
21 }

```

También se han creado las **excepciones** necesarias para controlar errores como que la tarea o el proyecto no se encuentren a la hora de borrarlos, cambiar el estado, etc...

- Una vez los servicios ya tienen los métodos para manipular los datos del modelo, se crean los siguientes controladores:

#### **- ProyectoController:**

- Es el encargado de manejar las acciones de crear y borrar proyectos de un equipo. Utiliza el middleware de la historia de usuario 016 para revisar que el que realiza la acción de crear o borrar es el administrador del equipo. En el se controlan las siguientes rutas:

**GET /proyectos/{id}:** Ruta encargada de mostrar el tablero Kanban con toda la información de un proyecto.

**POST /equipos/{id}/proyecto:** Con los datos necesarios en el body crea en el equipo id un proyecto.

**DELETE /equipos/{id}/proyecto/{id}:** Elimina un proyecto de un equipo

**GET /equipos/{id}/proyectos/nuevo:** Devuelve la vista con el formulario para crear un nuevo proyecto donde se le indica el

nombre. Al igual que el POST y el DELETE se encuentran protegidos por el middleware que revisa si el usuario que realiza la acción es el administrador del equipo.

- **TareaProyectoController:**

- Es el encargado de gestionar las peticiones para crear, eliminar o modificar TareaProyecto dentro de un Proyecto. Gestiona las siguientes rutas:

**PATCH /tareasproyecto/{id}:** Con el middleware de si el usuario pertenece al equipo, actualiza el estado de la tarea al indicado en el cuerpo de la petición.

**DELETE /proyectos/{id}/tareas/{tareaid}:** Elimina la tarea del proyecto. También filtra la petición con el middleware para ver si el usuario pertenece al equipo.

**POST /proyectos/{id}/tareas.** Crea una tarea en el proyecto con los datos pasados en el cuerpo de la petición. También utiliza el middleware.

- Las vistas que se han modificado son:

- **detallesEquipo.** Ahora muestra los proyectos si eres miembro del equipo. Además si eres administrador tienes un botón para añadir o eliminar proyectos.

- **formNuevoProyecto:** Es la encargada de mostrar un formulario con un input para el nombre que permite crear proyectos para un equipo.

- **detallesProyecto.** Esta vista muestra todas las tareas del proyecto en el tablero Kanban. También tiene un pequeño formulario en la parte de arriba que permite añadir tareas a todos los miembros del equipo. Para implementar el tablero Kanban se ha utilizado la librería de javascript **jKanban**. Para mapear los datos en Javascript se ha utilizado en la etiqueta `<script>` el atributo `th:inline="javascript"` que ha permitido añadir instrucciones Thymeleaf dentro del código javascript. Un ejemplo de como se gestiona las instrucciones de Thymeleaf dentro del script es a la hora de implementar la tarea dentro del tablero. La variable **buttonDelete** hace referencia al botón que lanza una petición DELETE para eliminar la tarea.

```
1 <script type="text/javascript" th:inline="javascript">
2
3
4     document.addEventListener('DOMContentLoaded', function () {
5         /*[# th:each="tarea : ${projeto.tareasProjeto}"]*/
6         let buttonDelete = `<button type="button" class="close" onclick="del('/proyectos/' +
7             [${projeto.id}] + '/tareas/' + [${tarea.id}] + ')" aria-label="Close"><span aria-
8             hidden="true">&times;</span></button>`;
9         console.log("Añadiendo tareas al board");
10        KanbanTest.addElement("[${tarea.status}]", {
11            id: "[${tarea.id}]",
12            title: "[${tarea.nombre}]" + buttonDelete
13        });
14    /*/]*/
15 });
16 </script>
```

## 023 Las tareas de un proyecto de equipo pueden ser asociadas a usuarios

Para implementar esta historia de usuario se han realizado las siguientes acciones

- Se ha añadido la relación entre el modelo **Usuario** y **TareaProyecto**. Una TareaProyecto puede contener muchos usuarios.
- En el **TareaProyectoService** se han creado los métodos de añadir y eliminar usuarios de una tarea, también se gestionan los posibles errores cuando los usuarios no estén en el equipo.
- En las vistas utilizando el método de java se le añade al lado del botón de eliminar tarea uno para unirse o abandonar si están en él.

```

1     document.addEventListener('DOMContentLoaded', function () {
2         let buttonDelete, htmlUsuarios, buttonJoinOrLeave, usuario
3         /*[# th:each="tarea : ${proyecto.tareasProyecto}"]*/
4         let buttonDelete = `<button type="button" class="close" onclick="del('/proyectos/' + [(${proyecto.id})] +
+ '/tareas/' + [(${tarea.id})])" aria-label="Close"><span aria-hidden="true">&times;</span></button>
5         buttonDelete = `<button type="button" class="close" onclick="del('/proyectos/' + [(${proyecto.id})] +
+ '/tareas/' + [(${tarea.id})])" aria-label="Close"><span aria-hidden="true">&times;</span></button>
6
7         htmlUsuarios = ''
8         /*[# th:each="usuarioTarea : ${tarea.usuarios}"]*/
9         usuario = "[(${usuarioTarea.nombre})]"
10        htmlUsuarios = htmlUsuarios + `<div style="margin-top: 5px;"> ` + usuario + `</div>
12        /*[/]*/
13
14        buttonJoinOrLeave = ''
15        /*[# th:if="${tarea.usuarios.contains(usuario)}"]*/
16        buttonJoinOrLeave = buttonJoinOrLeave + `<div onclick="del('/tareaproyecto/' + [(${tarea.id})] +
+ '/usuarios/' + [(${usuario.id})])" class="d-flex justify-content-center"><button class="btn btn-danger btn-
sm">Salir</button></div>
17        /*[/]*/
18        /*[# th:if="${!tarea.usuarios.contains(usuario)}"]*/
19        buttonJoinOrLeave = buttonJoinOrLeave + `<div onclick="post('/tareaproyecto/' + [(${tarea.id})] +
+ '/usuarios/' + [(${usuario.id})])" class="d-flex justify-content-center"><button class="btn btn-success btn-
sm">Unirse</button></div>
20        /*[/]*/
21        console.log("Añadiendo tareas al board");
22        KanbanTest.addElement("[(${tarea.status})]", {
23            id: "[(${tarea.id})]",
24            title: "[(${tarea.nombre})]" + buttonDelete
25            title: "[(${tarea.nombre})]" + buttonDelete + htmlUsuarios + buttonJoinOrLeave
26        });
27        /*[/]*/
28    });

```

## 024 Tareas hoy

Para implementar esta historia de usuario se han realizado las siguientes acciones

- En el modelo **Tarea** se ha creado un método que permite revisar si la tarea vence en la fecha de hoy.

```

1 public boolean isToday(){
2     if(fechaLimite == null) return false;
3     Date now = new Date();
4     Calendar cal1 = Calendar.getInstance();
5     Calendar cal2 = Calendar.getInstance();
6     cal1.setTime(now);
7     cal2.setTime(fechaLimite);
8     boolean sameDay = cal1.get(Calendar.DAY_OF_YEAR) == cal2.get(Calendar.DAY_OF_YEAR) &&
9             cal1.get(Calendar.YEAR) == cal2.get(Calendar.YEAR);
10    return sameDay;
11 }

```

- Desde la vista simplemente se accede a las tareas del usuario logueado y se realiza un for que muestra la tarea si se cumple la condición de la función de arriba.

```
1 <div><b>Tareas para hoy:</b></div>
2 <th:block th:each="tarea: ${usuario.tareas}">
3     <th:block th:if="${tarea.isToday()}">
4         <div class="tareaHoy">- <span th:text="${tarea.titulo}"></span></div>
5     </th:block>
6 </th:block>
```

- Para el contador se realiza con Javascript, ya que una vez está el documento cargado realiza la cuenta de cuantos objetos div con class **tareaHoy** existen y lo añade al icono de la campana.

```
1 document.addEventListener('DOMContentLoaded', function () {
2     let element = document.getElementById("notificationSize")
3
4     let element2 = document.getElementsByClassName("tareaHoy")
5     element.innerHTML = parseInt(element.innerHTML) + element2.length
6
7});
```

## 025 Notificaciones por correo electrónico

Para implementar esta historia de usuario se han realizado las siguientes acciones:

- Hemos editado el método que se encarga de enviar la invitación a equipo a los diferentes usuarios para añadir un envío de correo electrónico en caso de que la función tenga éxito.

InvitacionService.java

```
1 private void checkInvitacionRepetida(InvitacionData invitacionDTO) {
2     List<Invitacion> invitacion =
3         invitacionRepository.findByIdEquipoAndIdUsuario(invitacionDTO.getIdEquipo(),
4             invitacionDTO.getIdUsuario());
4 @@ -55,6 +62,29 @@ public void invitar(InvitacionData invitacionDTO) {
5
6     Invitacion invitacion = new Invitacion(invitacionDTO.getIdEquipo(),
7         invitacionDTO.getIdUsuario());
8     invitacionRepository.save(invitacion);
9     try{
10         SimpleMailMessage email = new SimpleMailMessage();
11         email.setTo(usuario.getEmail());
12         email.setSubject("Invitación a un equipo");
13         StringBuffer body = new StringBuffer();
14         body.append("*****\n");
15         body.append("Correo electrónico AUTOMATIZADO - No responder a este
16         mensaje\n");
17         body.append("*****\n");
18         body.append("Fecha: ");
19         body.append(new Date());
20         body.append("\n");
21         body.append("Solicitud de unión del equipo " + equipo.getNombre() + ",\n
22         para aceptar la invitación haga clic en el siguiente enlace: ");
23         body.append("\n");
24         body.append("http://localhost:8080/login");
25         body.append("\n");
26         body.append("*****\n");
27         email.setText(body.toString());
28         sender.send(email);
29     } catch (MailAuthenticationException e){
30         System.out.println("El correo introducido no es valido -> " +
e.getMessage());
31     }
31 }
```

- Hemos editado el método que se encarga de registrar a los nuevos usuarios para que en el caso que tenga éxito el registro envíe un email al usuario avisando de que ha sido registrado correctamente en el servidor de nuestra aplicación.

UserService.java

```
1 public Usuario blockUser(Long idUser){  
2     logger.debug("Bloqueando al usuario " + idUser + "...");  
3     @@ -72,7 +79,32 @@ else if (usuario.getEmail() == null)  
4         throw new UserServiceException("El usuario no tiene email");  
5     else if (usuario.getPassword() == null)  
6         throw new UserServiceException("El usuario no tiene password");  
7     else return usuarioRepository.save(usuario);  
8     else {  
9         try {  
10             SimpleMailMessage email = new SimpleMailMessage();  
11             email.setTo(usuario.getEmail());  
12             email.setSubject("Registro en ToDoList");  
13             StringBuffer body = new StringBuffer();  
14             body.append("*****\n");  
15             body.append("Correo electrónico AUTOMATIZADO - No responder a este  
mensaje\n");  
16             body.append("*****\n");  
17             body.append("Fecha: ");  
18             body.append(new Date());  
19             body.append("\n");  
20             body.append("_____  
_____  
");  
21             body.append("\n");  
22             body.append("Te has registrado correctamente en ToDoList con el  
siguiente usuario: " + usuario.getEmail() + ".");  
23             body.append("\n");  
24             body.append("http://localhost:8080/login");  
25             body.append("\n");  
26             body.append("_____  
_____  
");  
27             email.setText(body.toString());  
28             sender.send(email);  
29         } catch (MailAuthenticationException e){  
30             System.out.println("El correo introducido no es valido -> " +  
e.getMessage());  
31         }  
32         return usuarioRepository.save(usuario);  
33     }  
34 }
```

- No ha sido necesario realizar test a los métodos anteriormente mencionados ya que han sido comprobados anteriormente en la historia de recibir un correo electrónico en caso de olvidar la contraseña.

### 3. Puesta en producción

#### *Dockerización de la aplicación*

Se ha subido la versión 1.4.0 al contenedor a docker hub.

<https://hub.docker.com/r/sergioaluua/mads-todolist-equipo15>

#### *Despliegue en producción*

Se han añadido a los schemas ya existentes:

- schema-1.4.0.sql - Schema de la versión 1.4.0 de la base de datos
- schema-1.3.0-1.4.0 - Script de migración de la versión 1.3.0 a la 1.4.0

También se ha añadido el backup con datos de la presentación:

- backup20122022.sql

El esquema de la base de datos resultante es el siguiente:

```
✓ mads (4)
  ✓ public
    ✓ tables (9)
      ✓ comentarios_equipo
        ✓ columns
          ↗ id bigint
          ↗ comentario character varying(255)
          ↗ fecha timestamp without time zone
          ↗ equipo_id FK bigint
          ↗ usuario_id FK bigint
        ✓ index
          ↗ comentarios_equipo_pkey id PRIMARY
      ✓ equipo_usuario
        ✓ columns
          ↗ fk_equipo FK bigint
          ↗ fk_usuario FK bigint
        ✓ index
      ✓ equipos
        ✓ columns
          ↗ id bigint
          ↗ descripcion character varying(255)
          ↗ is_private boolean
          ↗ nombre character varying(255)
          ↗ admin_id FK bigint
        ✓ index
      ✓ invitaciones
        ✓ columns
          ↗ id bigint
          ↗ id_equipo bigint
          ↗ id_usuario bigint
        ✓ index
      ✓ proyectos
        ✓ columns
          ↗ id bigint
          ↗ nombre character varying(255)
          ↗ equipo_id FK bigint
        ✓ index
      ✓ tareaproyecto_usuario
        ✓ columns
          ↗ fk_usuario FK bigint
          ↗ fk_tareaproyecto FK bigint
        ✓ index
```

✓	□	tareas
✓	□	columns
	🔗	id bigint
	🕒	fecha_limite date
	🕒	status character varying(255)
	🕒	titulo character varying(255)
	🔗	usuario_id FK bigint
>	🔗	index
✓	□	tareas_proyecto
✓	□	columns
	🔗	id bigint
	🕒	nombre character varying(255)
	🕒	status integer
	🔗	proyecto_id FK bigint
>	🔗	index
✓	□	usuarios
✓	□	columns
	🔗	id bigint
	🕒	blocked boolean
	🕒	code integer
	🕒	email character varying(255)
	🕒	fecha_nacimiento date
	🕒	is_admin boolean
	🕒	nombre character varying(255)
	🕒	password character varying(255)
>	🔗	index

## 4. Informe sobre la evolución del desarrollo

### *Informe general*

La práctica se ha desarrollado siguiendo un **enfoque ágil**. Hemos utilizado los siguientes elementos/metodologías:

- **TDD:** Se ha intentado desarrollar al máximo siguiendo la metodología Test Driven Development en los servicios y modelos. Esto nos ha permitido que a la hora de revisar el código e integrarlo al ser funcionalidades más grandes que la práctica anterior tenemos certeza de que todo lo implementado ha sido probado.
- **Tablero de Trello:** El tablero de Trello nos ha sido útil a la hora de conocer el estado actual del proyecto y ver cómo vamos a la hora de la entrega.
- **Historias de usuario desarrolladas en Google Drive:** Se han realizado documentos de Drive para cada historia de usuario antes de empezar con el desarrollo. También se han diseñado los mockups de la interfaz para cada una, esto nos ha ayudado a la hora de diseñar las vistas ya que sabíamos qué era exactamente lo que teníamos que implementar de forma visual.
- **Revisor en los pull request:** Al contrario que la práctica anterior, en esta antes de hacer merge de una rama con develop era necesario que un compañero revisara el código. Si éste detectaba algo que se podía mejorar se ponía en común con el equipo para discutirlo.
- **Pair Programming:** Se han realizado sesiones de Pair Programming que se comentarán más abajo.
- **Reuniones semanales:** Hemos hecho casi todas las semanas una reunión para conocer el estado del proyecto y los posibles problemas que hemos podido tener.
- **Integración Continua:** Cada vez que se subía algo al repositorio se ejecutaban los tests de integración.
- **Priorización del valor de negocio:** A la hora de proponer historias de usuario se han seleccionado aquellas que aportaban más valor de negocio a la aplicación, por lo que historias de usuario enfocadas al usuario administrador al no aportar un valor de negocio tan grande como las que se han implementado se han descartado.

## *Informe de la estimación*

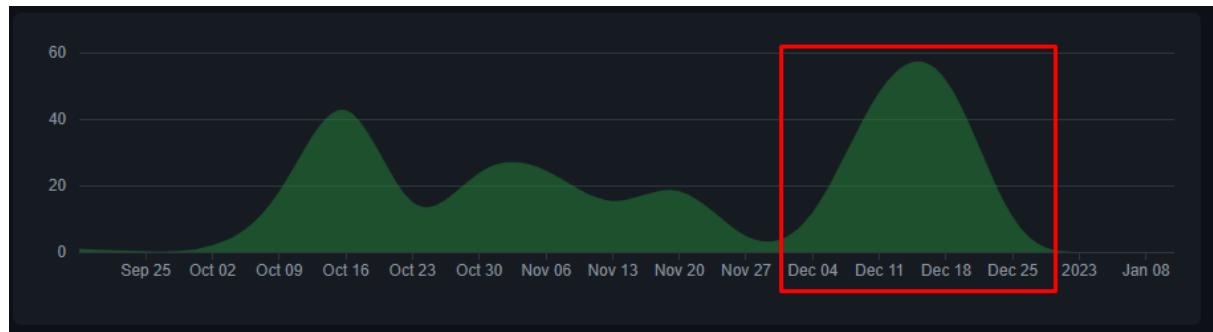
La estimación que se ha realizado al inicio de la práctica fue **extremadamente positiva**. Entendemos que al no haber trabajado con Spring Boot tanto como hasta ahora, puede haber fallos a la hora de estimar el tiempo. Vamos a ver la tabla sobre la comparación de la estimación antes de hacer la práctica y una vez hecha.

Nombre de la historia	Estimación Inicial	Tiempo final
<i>014 Tarea con fecha y vista de calendario</i>	1	2
<i>015 Editar perfil de usuario</i>	1	1
<i>016 Administrador del equipo</i>	1	2
<i>017 Añadir comentarios a los equipos</i>	1	1
<i>018 Tareas con estado de progreso y vista en tablero</i>	1	2
<i>019 Añadir equipos a los que administras a la pestaña de 'Mis equipos'</i>	1	1
<i>020 Recuperar contraseña por correo electrónico</i>	1	1
<i>021 Equipos públicos o privados</i>	2	2
<i>022 El equipo puede contener proyectos que a su vez contienen tareas</i>	2	2
<i>023 Las tareas de un proyecto de equipo pueden ser asociadas a usuarios</i>	1	2
<i>024 Recordatorios de tareas "hoy"</i>	1	1
<i>025 Notificaciones por correo electrónico</i>	1	1

Como podemos observar, han habido **4/12** que han sido mal estimadas, lo que significa que 1 de cada 3 historias de usuario han tenido una mala estimación. Aún así el equipo ha decidido no dejar ninguna tarea.

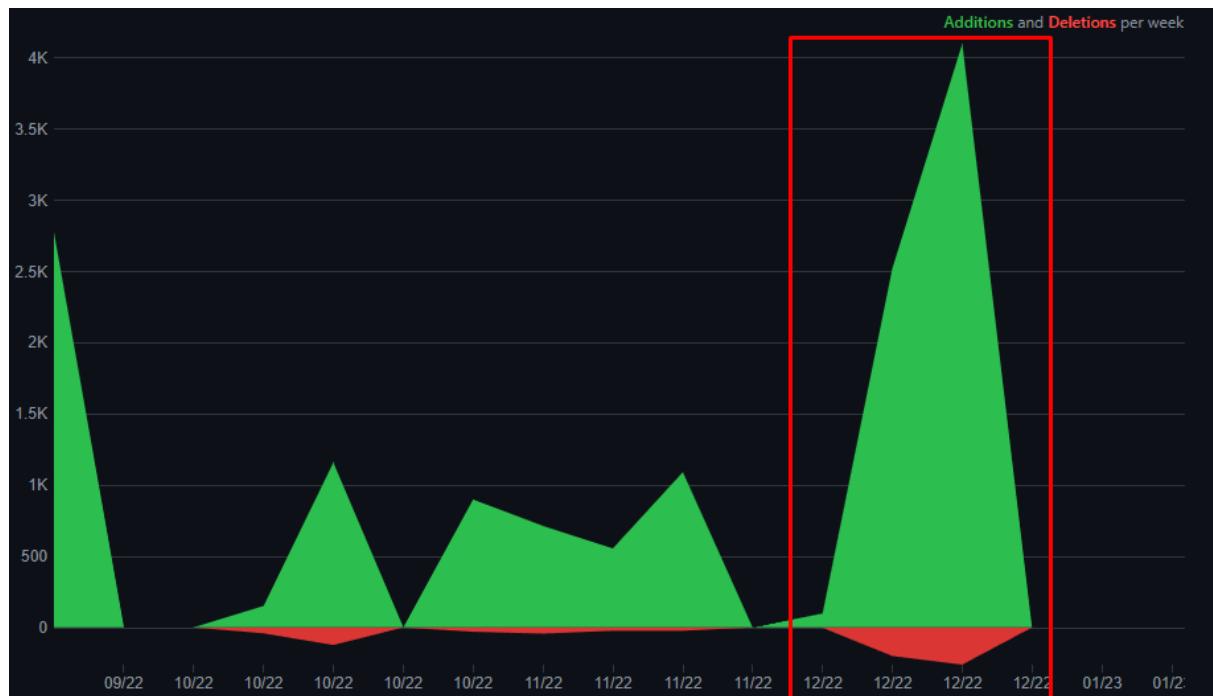
## *Informe del tiempo de desarrollo*

Si entendemos que un commit representa una **mini-funcionalidad** de la aplicación, podemos analizar el siguiente gráfico que nos proporciona Github.

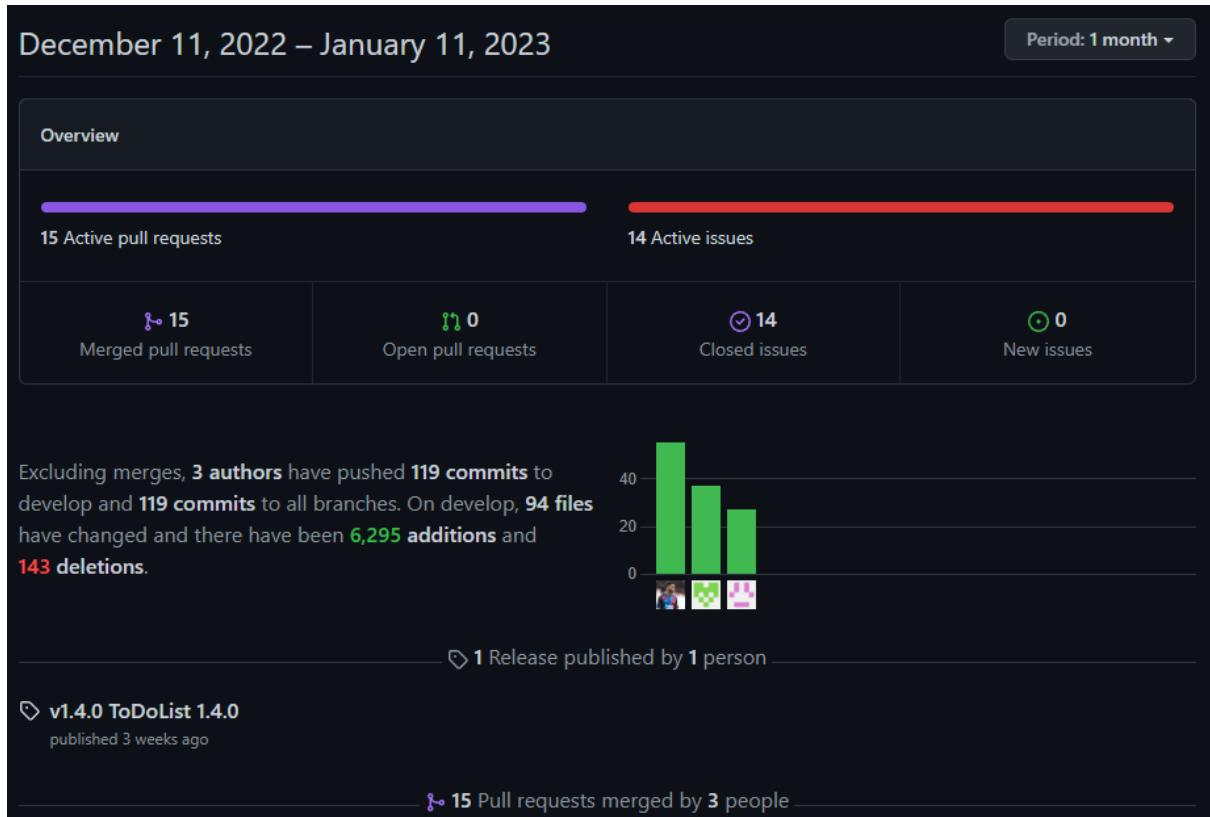


Podemos observar primero, que el **volumen** de commits ha sido muchísimo mayor al de anteriores prácticas. También podemos ver que en la primera semana hay menos commits que en las siguientes, esto es debido a que la primera semana se estaban **definiendo y desarrollando** las **historias de usuario** por lo que aún no estábamos desarrollando código.

También podemos observar cómo también la frecuencia de añadir código se corresponde con la gráfica de los commits:



Aquí tenemos también otro informe que nos proporciona Github del día **11/12/2022 - 11/01/2023**



También se han realizado capturas del tablero de **Trello** durante el desarrollo de la versión.

Backlog TodoList (equipo 15)

Backlog (1)

- 2 El equipo puede contener proyectos que a su vez contienen tareas
- 1 Tareas contienen estado como el tablero Kanban (podrán ser representadas de esa manera tamb)
- 1 Las tareas de un proyecto de equipo pueden ser asociadas a usuarios
- 1 Tarea con etiqueta
- 1 Recordatorios de tareas "hoy"

Backlog (2)

- 2 Equipos públicos o privados
- 1 Notificaciones por correo electrónico
- 1 Recuperar contraseña por correo electrónico
- 1 Pestaña de 'Mis equipos'

Seleccionadas

- 1 Añadir comentarios a los equipos
- 1 Notificaciones por correo electrónico
- 1 Recuperar contraseña por correo electrónico
- 1 Pestaña de 'Mis equipos'

En marcha

- 1 014 Tarea con fecha y vista de calendario
- 1 015 Editar perfil de usuario
- 1 016 Administrador del equipo

En pruebas

- + Add a card

Terminadas

- + Add a card

Backlog TodoList (equipo 15)

Board

Power-Ups Automation Filter Share

Backlog (1) Backlog (2) Seleccionadas En marcha En pruebas Terminadas

+ Add a card + Add a card

El equipo puede contener proyectos que a su vez contienen tareas

Equipo público o privados

Notificaciones por correo electrónico

020 Recuperar contraseña por correo electrónico

Notificaciones por correo electrónico

021 Equipo público o privado

Notificaciones por correo electrónico

022 El equipo puede contener proyectos que a su vez contienen tareas

023 Tareas de un proyecto de equipo pueden ser asociadas a usuarios

024 Recordatorios de tareas "hoy"

025 Tarea con etiqueta

026 Recordatorios de tareas "hoy"

027 Tarea con fecha y vista de calendario

028 Pestaña de "Mis equipos"

029 Editar perfil de usuario

030 Administrador del equipo

031 Tareas con estado de progreso y vista en tablero

032 Tarea con fecha y vista de calendario

033 Pestaña de "Mis equipos"

034 Añadir comentarios a los equipos

Backlog TodoList (equipo 15)

Board

Power-Ups Automation Filter Share

Backlog (1) Backlog (2) Seleccionadas En marcha En pruebas Terminadas

+ Add a card + Add a card

Tarea con etiqueta

Recordatorios de tareas "hoy"

Añadir equipos a los que administras a la pestaña de "Mis equipos"

Las tareas de un proyecto de equipo pueden ser asociadas a usuarios

020 Recuperar contraseña por correo electrónico

021 Equipo público o privado

022 El equipo puede contener proyectos que a su vez contienen tareas

023 Tareas de un proyecto de equipo pueden ser asociadas a usuarios

024 Recordatorios de tareas "hoy"

025 Tarea con etiqueta

026 Recordatorios de tareas "hoy"

027 Tarea con fecha y vista de calendario

028 Pestaña de "Mis equipos"

029 Editar perfil de usuario

030 Administrador del equipo

031 Tareas con estado de progreso y vista en tablero

032 Tarea con fecha y vista de calendario

033 Pestaña de "Mis equipos"

034 Añadir comentarios a los equipos

+ Add another list

Backlog TodoList (equipo 15)

Board

Power-Ups Automation Filter Share

Backlog (1) Backlog (2) Seleccionadas En marcha En pruebas Terminadas

+ Add a card + Add a card

Notificaciones por correo electrónico

024 Recordatorios de tareas "hoy"

Notificaciones por correo electrónico

025 Tarea con etiqueta

026 Recordatorios de tareas "hoy"

027 Tarea con fecha y vista de calendario

028 Pestaña de "Mis equipos"

029 Editar perfil de usuario

030 Administrador del equipo

031 Tareas con estado de progreso y vista en tablero

032 Tarea con fecha y vista de calendario

033 Pestaña de "Mis equipos"

034 Añadir comentarios a los equipos

+ Add a card

### *Resumen sobre la evolución del desarrollo*

Podemos concluir que, a pesar de una mejorable estimación de tiempo de las funcionalidades, se han podido alcanzar los objetivos planteados. El uso de **TDD**, **Integración Continua**, tablero **Trello**, nos ha facilitado el tener un desarrollo **organizado y coherente** entre todos los miembros del equipo.

Por otra parte, las tareas se han ido **acumulando** debido a la dificultad que pensábamos que tendrían en un primer momento. También las **dependencias** entre las funcionalidades nos hacían priorizar, por ejemplo, la tarea de las invitaciones requería que el usuario administrador existiera ya en la aplicación para que las gestione.

En conclusión, la evolución del desarrollo podemos valorarla de forma **positiva** y después de este informe tendríamos la certeza de que se podría **mejorar** a la hora de estimar y priorizar las distintas historias de usuario.

## **5. Informe sobre las sesiones de *pair programming***

Se realizaron dos sesiones de *pair programming*.

### *Sergio y Eugenio*

Sesión presencial para implementar la vista en calendario de las tareas. En apartados anteriores explicamos que esta tarea resultó complicarse más de lo esperado. En gran parte se debe a que resultó complicado implementar la vista en calendario.

Se contaban con dos opciones: implementarlo de cero utilizando html, css y js o bien utilizar una librería de terceros.

Debido al limitado conocimiento del equipo en css, se optó por la utilización de una librería de terceros que nos diese parte del trabajo hecho. Lo que no se contaba era que la integración no sería tan sencilla como se esperaba, y es que resultó complicado averiguar cómo manipular los datos del modelo que se le insertan a la vista desde código Javascript.

La sesión de pair programming entre Sergio y Eugenio consistió en barajar entre varias posibles librerías y escoger. La elegida fue FullCalendar.io por la relación entre estética, facilidad de implementación y posibilidades que nos brindaba.

Una vez escogida la librería, en una sesión de 2h de pair programming quedó integrado el calendario de FullCalendar en las plantillas Thymeleaf del proyecto usando el modelo de la lista de tareas del usuario.

### *Eugenio y Sergio*

Sergio y Álvaro realizaron online esta sesión de *pair programming* para implementar la funcionalidad consistente en la creación de un usuario administrador de equipo, que tendría privilegios adicionales en comparación con los usuarios normales.

Comenzaron repasando las condiciones de satisfacción y cómo podrían implementarla. Una vez que estuvieron de acuerdo en el diseño del código requerido, comenzaron a escribirlo.

La sesión ayudó a corregir algunos errores depurando entre los dos el código y cambiando los roles en la sesión.

Al final de la sesión, la funcionalidad de creación de usuarios administradores de equipo había sido completamente implementada y estaba lista para ser probada.

Sergio y Álvaro revisaron juntos el código e hicieron algunos pequeños ajustes antes de enviarlo a la rama de desarrollo.

En resumen, la sesión de pair programming fue una experiencia colaborativa y productiva, donde Sergio y Álvaro trabajaron juntos para diseñar e implementar la funcionalidad de manera eficiente y con buenas prácticas en mente.

## 6. Resultado de la retrospectiva

¿Qué ha ido bien?

- Mediante el Pair Programming hemos reducido la tasa de errores y las malas decisiones de diseño logrando aumentar la calidad del código gracias a la transmisión de conocimiento entre los desarrolladores.
- Gracias a TDD hemos evitado escribir más código del que realmente necesitábamos, pudiendo apreciar una fluidez y mejora en el rendimiento de nuestra aplicación.
- Gracias a la buena comunicación entre los miembros del equipo y al haber aplicado el concepto de 'Brainstorming' o 'Lluvia de ideas' hemos logrado generar un Backlog de calidad, con ideas innovadoras.
- Hemos logrado finalizar el proyecto con la mayor parte de las historias de usuario definidas en su día.

¿Qué ha ido mal?

- Fallo con la estimación del tiempo de desarrollo de cada historia de usuario.
- Duplicación de código en algunas clases.

¿Qué hacer para mejorar?

- Aplicar la técnica de Refactorización para evitar problemas de duplicación de código y por tanto mejorar el rendimiento.
- Estimar el tiempo de desarrollo de una historia de usuario midiendo lo que se ha tardado en desarrollar sus semejantes.