Hecho por Pedro, Sergio y Pau.

Salvador Gadea, 2º DAM

Proyecto Intermodular **2025-2026**

2º DAM Salvador Gadea Pedro, Sergio y Pau

Índice

Índice 2
Introducción 3
Primera evidencia: Gestión de Hilos (backend) 3
Segunda evidencia: Geolocalización (aplicación móvil) 4
Tercera evidencia: Persistencia en Base de Datos 9
Cuarta evidencia: Configuración externa 10
Quinta evidencia: Sistema de Logs 11
Sexta evidencia: Tests unitarios 11
Memoria de digitalización 12
Metodología 12
Control de versiones 13
¿Cómo poner en marcha el proyecto? 15
Conclusión 16

Introducción

Como encargados de la aplicación y proyecto “theGreenWay”, hemos hecho un documento que explicará cómo hemos realizado cada una de los requisitos que se nos pedían al principio del proceso de desarrollo, así como su funcionamiento y localización dentro de la raíz del proyecto.

También hablaremos de los problemas que hemos tenido a la hora de hacer cosas como el GitHub, el Trello o la implementación de nuevas características de la aplicación (nuevas “Features”).

Además, mencionaremos cosas como la relevancia para la compañía del proyecto “theGreenWay” y las ventajas que puede aportar. Por último, haremos una reflexión de lo que nos ha parecido trabajar en este proyecto y del equipo.

Primera evidencia: Gestión de Hilos (backend)

Lo primero que se nos pedía era un programa hecho en Java que hiciese de servidor y que fuese capaz de manejar múltiples peticiones al mismo tiempo para luego almacenar dicha información y/o procesarla.

Las peticiones tenían que poder ser hechas desde cualquier sitio y teníamos que utilizar la tecnología de Socket y ServerSocket para escuchar en determinados puertos.

La aplicación móvil debía ser capaz de mandar y recibir información del servidor usando el protocolo TCP.

El servidor consiste básicamente en un bucle infinito que recibe llamadas TCP usando ServerSocket. Cuando un usuario está intentando conectarse, invoca un hilo dedicado a procesar sus datos (haciendo que sea capaz de manejar tantas llamadas al mismo tiempo como la máquina en la que está alojado lo permita) y le manda el Socket.

Al llamar a un nuevo hilo crea una clase de tipo GestorCliente.java que procesa la llamada. La lógica del servidor se encuentra en la ruta de archivos (partiendo desde la raíz): src/Main.java

La clase GestorCliente.java detecta si la llamada es para pedir autenticación o para mandar datos basándose en la longitud del mensaje. Si la llamada es para autenticar, comprueba las credenciales usando BBDD y devuelve un OK si son correctas (puede ser fácilmente reforzado usando un generador de claves). Si el mensaje son datos, se mandan a BBDD, que los almacena en la base de datos. La lógica de GestorCliente se encuentra en la ruta desde la raíz: src/GestorCliente.java. Al cerrar el hilo, se asegura de cerrar el socket con un finally {socket.close()}

Segunda evidencia: Geolocalización (aplicación móvil)

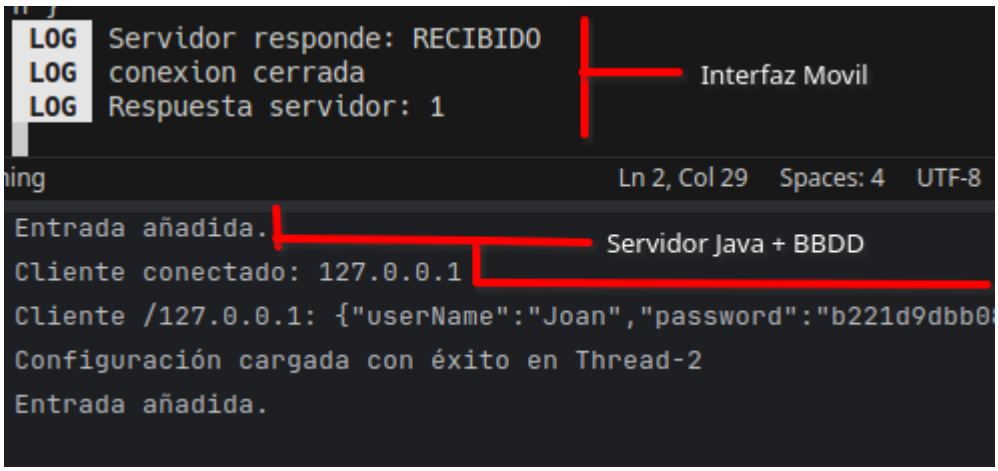
El Frontend de theGreenWay se ha desarrollado íntegramente en react native con la herramienta de Expo la cual dispone de varias librerías que han facilitado mucho el desarrollo de la app.

El funcionamiento de la aplicación se basa en sincronizar varios componentes fundamentales para adquirir la ubicación precisa en coordenadas y poder representarlas gráficamente en tiempo real sobre el mapa.

Primero de todo, para la gestión de usuarios, hemos implementado un sistema de autenticación robusto con sesión persistente en el dispositivo, gracias a los métodos crypto-js. Además, la aplicación valida las credenciales directamente con la base de datos, permitiendo el paso siempre que el servidor de la señal de “RECIBIDO”.
(AppMovil/hooks/useAuth.ts).

```
export function useAuth() {
      . . .

   const SESSION_FILE = ((FileSystem as any).documentDirectory) + 'session.txt';

   useEffect(() => {
       const checkSession = async () => {
           try {
               const savedUser = await
FileSystem.readAsStringAsync(SESSION_FILE);
               if (savedUser) {
                   router.replace({
                       pathname: "/(drawer)",
                       params: { idUsuario: savedUser }
```

```
                    });
                }
            } catch (e) {
            }
        };
        checkSession();
    }, []);

    const onLoginPress = async () => {
        setError("");

        if (!user || !pass) {
            setError("Por favor, rellena todos los campos.");
            return;
        }

        setCargando(true);

        try {
            const passEncryptada= CryptoJS.SHA256(pass).toString();
            const esValido = await validarCredenciales(user, passEncryptada);
```



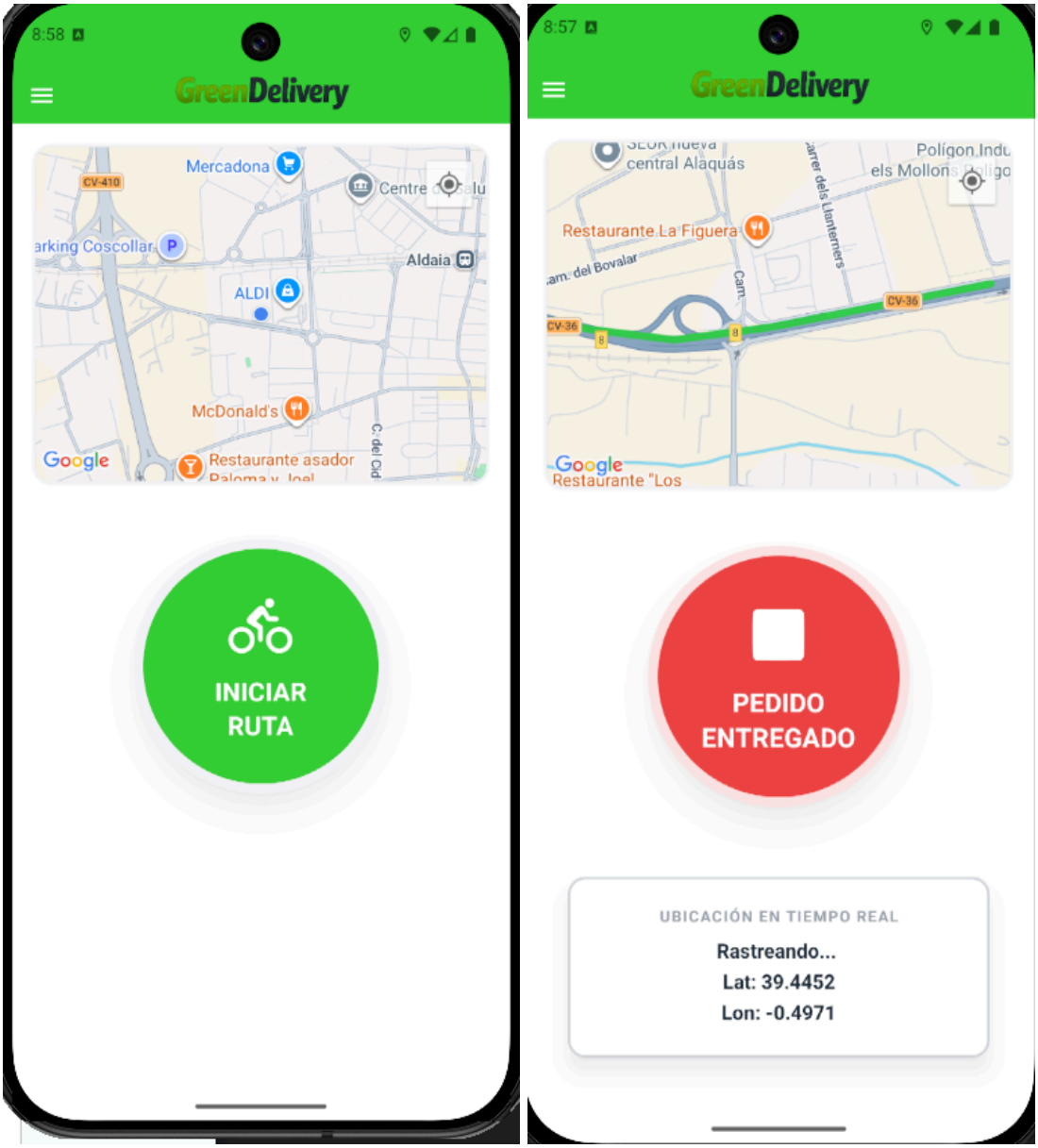
Para interactuar con la ubicación del dispositivo, empleamos la librería expo-location, la cual primero verifica que el usuario haya autorizado el uso de GPS y después hace una lectura mediante el método getCurrentPositionAsync (desde la raíz) (AppMovil/hooks/useRastreador.tsx)

```
let location = await Location.getCurrentPositionAsync({});
      const lat = location.coords.latitude;
      const lon = location.coords.longitude;
      setRuta(prev => [...prev, { latitude: lat, longitude: lon }]);

      setMensaje(`Rastreando... \nLat: ${lat.toFixed(4)}\nLon:
${lon.toFixed(4)}`);

      await enviarCoordenadas(lat, lon, userName);
```

Una vez obtenidas las coordenadas, la aplicación las renderiza gracias a la librería react-native-maps utilizando Google Maps para el mapa en la interfaz.

La interfaz está compuesta de 2 elementos dinámicos, MapView, que es el lienzo donde se despliega el mapa y el polyline que se encarga de dibujar el historial de la ruta. Todos estos componentes se encuentran en la interfaz en la que se encuentra el tracker (AppMovil/app/(drawer)/index.tsx)

```
<MapView
         provider={PROVIDER_GOOGLE}
         style={styles.map}
         initialRegion={initialRegion}
         showsUserLocation={true}
         showsMyLocationButton={true}
       >
      <Polyline
           coordinates={ruta}
           strokeColor={modoNoche ? "#32CD32" : "#32cd4cff"}
           strokeWidth={6}
           lineCap="round"
           lineJoin="round"
           geodesic={true}/></MapView>
```



Finalmente, el componente que completa todo este flujo es el encargado de transmitir estos datos al servidor java, para ello se emplea una comunicación mediante Sockets TCP utilizando la librería react-native-tcp-socket. Este módulo se encarga de serializar las coordenadas a JSON, establece la conexión con la IP y puerto definidos en .env y envía los datos.(AppMovil/hooks/useServidor.tsx)

```
const enviarCoordenadas = (latitud: any, longitud: any, userName: any) => {

    return new Promise((resolve, reject) => {
      const datos = {
        lat: latitud,
        lon: longitud,
        userName: userName
      };

      const jsonEnviar = JSON.stringify(datos);
      console.log("enviando por tcp...", jsonEnviar);

      const cliente = TcpSocket.createConnection({
        port: PUERTO,
        host: IP_SERVIDOR,
      }, () => {
        cliente.write(jsonEnviar + '\n');
      });
```

```
 LOG  enviando por tcp... {"lat":39.45954,"lon":-0.4704567,"userName":"Joa
n"}
 LOG  Servidor responde: RECIBIDO
 LOG  conexion cerrada
```

Tercera evidencia: Persistencia en Base de Datos

En este apartado se nos pedía que usáramos una base de datos para almacenar la información de la aplicación de los usuarios en MySQL y lo conectásemos con la aplicación.

En el primer Sprint se creó el documento “dump.sql” que contenía un dump de la base de datos con unos diez usuarios ya insertados. Estos son los usuarios de la tabla (con las contraseñas sin encriptar):

(1, 'Joan', 'hola'),

(2, 'Iker', 'hola1'),

(3, 'Gerard', 'hola2'),

(4, 'Carles', 'hola3'),

(5, 'Sergio', 'hola4'),

(6, 'Xavi', 'hola5'),

(7, 'Andrés', 'hola6'),

(8, 'David', 'hola7'),

(9, 'Fernando', 'hola8'),

(10, 'Jesús', 'adios')

Nosotros montamos el entorno de base de datos usando Docker pero hay otras formas de hacerlo. Docker, sin embargo, es la más fácil con diferencia. En el primer sprint también hicimos la lógica dentro de BBDD.java, que se usa para conectar con la base de datos. Después añadimos otra clase dentro de BBDD que se encargaba de hacer el inicio de sesión con la base de datos. La ruta para acceder a dump.sql desde la raíz es: src/dump.sql.

Tanto en el móvil como en la base de datos, se usa la encriptación SHA2 (256), que es bastante estándar.

En BBDD, tenemos dos clases, insertarCliente (datos) y comprobarCredenciales. La segunda clase recoge una clase de tipo Usuario (que contiene básicamente un usuario y una contraseña) y hace una conexión con la base de datos para corroborarlos. Si coincide devuelve un “SÍ”, y si no, devuelve un “NO” directamente al móvil.

En la primera clase, se recoge una clase de tipo Cliente, que es básicamente unas coordenadas y un id de usuario. BBDD inserta los datos (previniendo contra inserción de SQL) y devuelve un “RECIBIDO” o “NO_RECIBIDO” para debuggear. La ruta de BBDD desde la raíz es src/BBDD.java. Las rutas de las clases Cliente y Usuario son las siguientes: src/Classes/Cliente.java y src/Classes/Usuarios.java.

La estructura de la BBDD consta de dos tablas. La primera, usuarios, guarda los usuarios y contraseñas. La segunda, hace referencia a la anterior, y guarda las posiciones de los conductores asegurándose de que son únicas respecto a tiempo y usuario.

Cuarta evidencia: Configuración externa

Este apartado lo trabajamos y terminamos en el segundo Sprint,aquí se nos pedía que tuviéramos un archivo en texto plano llamado “server.properties” en el que se guardasen unas credenciales, unos datos que no estuviesen escritos directamente en el código fuente. La ruta del archivo es server.properties (está en la raíz).

Creamos la clase Credenciales, que llama al archivo plano server.properties, lee su contenido y lo almacena en la clase Credenciales, desde donde se pueden usar unos getters para obtener los valores sensibles. Dicha clase tiene un constructor vacío para poder instanciarla y un método inicializar() que asigna los valores (inicialmente vacíos) a los leídos en el archivo.

Está instanciada en Main (para el puerto en el que el servidor escucha) y en BBDD (para conectarse a la base de datos mediante enlace, usuario y contraseña). La ruta hasta el archivo Credenciales es la siguiente (desde la raíz): src/Credenciales.java.

También hicimos un archivo de credenciales para el móvil, .env que se encuentra en la ruta AppMovil/.env desde la raíz. Este archivo proporciona la IP del servidor al que conectarse y el puerto en el que mirar.

Quinta evidencia: Sistema de Logs

Se nos pedía hacer o reforzar un sistema de logs para detectar o documentar errores.

Hemos creado una clase llamada LogsController con un método inicializar() y sin constructor, pero solo hace falta ejecutarlo una vez en main. Básicamente utiliza la clase de Java Logger y modifica el output por defecto de un error para que sea más agradable a la vista y asigna como ruta por defecto para el logdump la ruta (desde la raíz): logs/log.log.

Al instanciar el logger como variable estática en cada clase y poniéndolo dentro del catch sensible, cualquier error queda refactorizado y almacenado en log.log. La ruta de LogsController (desde la raíz) es la siguiente: src/Logs/LogsController.java.

Este es un ejemplo de aspecto de los logs que aparecen por pantalla y se almacenan:

```
Configuración cargada con éxito en main
[2026-02-11 00:23:28] [SEVERE] [Main lambda$main$0] Exception in thread main: / by zero.
```

Sexta evidencia: Tests unitarios

Se nos pedía hacer una clase de prueba en la que hiciéramos distintas pruebas unitarias para probar la robustez del programa.

Hicimos una clase llamada PruebasUnitarias.java en src/Test/PruebasUnitarias.java. En dicha clase se creó un entorno controlado cuyo funcionamiento fuese exactamente el mismo que Main, GestorCliente y BBDD, sin tener lógica detrás. Se le aplicaron 11 pruebas cubriendo todos los posibles inputs, algunas de las cuales esperaban errores fatales y todas ellas pasaron sin problemas. Al final de la sesión de debugeo se aplicó algo de la lógica de la clase al programa principal para corregir posibles brechas de contención, probando así su eficiencia.

Las pruebas unitarias funcionan haciendo uso de la tecnología JUnit4, instalada en el proyecto mediante librería jar.

Memoria de digitalización

El valor de theGreenWay reside en solucionar la ceguera logística de la última milla.

Permitimos que pequeños negocios pasen de gestionar sus pedidos por teléfono a tener un control visual en tiempo real de toda su flota, mejorando la seguridad del repartidor y los tiempos de entrega.

El presupuesto se basa en 80 horas de desarrollo Full Stack efectivo, divididas entre:

- Arquitectura de servidor en Java Concurrente y optimización de Base de Datos MySQL.
- Desarrollo de App en React Native con gestión de estado global, persistencia segura y geolocalización.

No es una app genérica basada en plantillas. Hemos desarrollado un protocolo de comunicación personalizado sobre TCP. Esto garantiza una transmisión de datos ultrarrápida y un consumo mínimo de batería y datos móviles para el repartidor.

Entonces el coste neto de la Aplicación completa sería de unos 1500 € aprox.

Metodología

Enlace Repositorio Trello: ENLACE (PULSAR AQUÍ).

Nuestro Trello ha acabado con una estructura bastante organizada y completa, la cual se basa en las fases de Backlog, Sprint WIP, testing y los “Done Sprints”, que son los sprints que ya hemos hecho en los cuales repartimos las tareas anteriores y que se dan por hechas. Ahora vamos a desarrollar por qué tenemos el Trello como lo tenemos y cómo hemos llegado hasta aquí.

Empezamos cronológicamente. Al principio no entendimos bien cómo teníamos que dividirnos las tareas del proyecto en cuestión, cometimos el error de pensar que las tareas proporcionadas serían lo suficientemente simples para ser realizadas sin ser partidas.

Reorganizamos y reestructuramos las tareas, las subdividimos de una manera más lógica y eficiente. No ayudó la falta de un compañero con el que no sabíamos si podríamos contar con él o no, resultando que al final no pudimos contar con él (abandonó el proyecto).

Tampoco entendíamos bien las fases, por lo tanto al principio cometimos errores durante el proceso de catalogar cuándo una tarea estaba en proceso, cuándo estaba lista para ser testeada y cuándo estaba finalizada.

Una vez entendimos todo esto empezamos con el proyecto, el Backlog se llenó de más tareas, pero dichas tareas eran más manejables y con un objetivo claro y sencillo. Resumimos y definimos en cada etiqueta del backlog qué había que hacer y cómo hacerlo.

En los primeros sprints se cometieron errores como pasar de la fase sprint a testing o de un WIP a un done. En los últimos sprints, la frecuencia de estos errores se redujo. Con el tiempo nos acostumbramos a usar Trello de manera correcta.

Por último, con la utilización y el tiempo, la información que había hecho se fue acumulando en una única fase “sprint”, lo que causó desorganización, así que por comodidad se tomó la decisión de crear un “Done Sprint” por cada uno de los Sprints.

Control de versiones

Realizamos el control de versiones con GitHub, localizado en el siguiente repositorio: ENLACE AQUÍ (PULSAR).

Hemos usado la forma “Production”, “Release{sprint}” y “{NombreFeature}-Feature”. Empezamos a usarla correctamente después del primer sprint, ya que en el primero lo hicimos mal. A partir del primer sprint desarrollamos un sistema de clonado y ramificación que hasta el sexto sprint nos ha servido.

En el segundo sprint empezamos a trabajar de forma correcta. He aquí una captura del segundo sprint en GitHub:

Tuvimos bastantes problemas con el control de versiones, principalmente porque ninguno de los participantes del equipo sabía cómo hacerlo. Tuvimos que aprendernos todos los comandos de Git y solucionar algunos problemas extras que salieron.

Algunas veces tuvimos que hacer chapuzas para poder subir los cambios o hacer merges porque GitHub tiene problemas con los archivos sql y/o porque el sistema de archivos cambiaba y obligaba a refactorizar desde terminal.

Una vez, sin querer, nos cargamos un sprint entero y tuvimos que hacer un respaldo (lo que nos llevó una hora).

Por último, hacia el cuarto sprint, las cosas empezaron a complicarse al hacer algunos merges, sobre todo al intentar actualizar el dump.sql. Esta es la captura del cuarto sprint:

Además, ahora que nos hemos fijado, parece ser que el historial del tercer sprint ha sido completamente eliminado de GitHub, probablemente debido a alguna chapuza que hicimos al inicio del cuarto (foto mostrando el hueco de una semana).

En general, podemos decir que trabajar con GitHub este proyecto nos ha beneficiado, ya que hemos ganado experiencia con el mismo y hemos aprendido a manejarlo de forma básica-media.

¿Cómo poner en marcha el proyecto?

Para poner en marcha el proyecto el primer paso es clonar el repositorio de GitHub. Sin embargo, una vez clonado habrá que usar un IDE para importar las librerías JAR que están dentro de lib.

Una vez hecho, hay que montar una base de datos funcional dentro de la LAN del ordenador, (que puede ser hecho por Docker) y ejecutar src/dump.sql. También hay que tener listo un emulador de móvil con una versión actual (no hace falta que sea la última) o un móvil conectado por cable USB con la opción depuración por USB activada.

Luego, hay que cambiar las credenciales. Hay que editar el archivo server.properties para configurar las credenciales de acceso a la base de datos como se desee y clonar el archivo .env.template (ruta AppMovil/.env.template), eliminar .template del nombre y cambiar la IP y puerto al que el móvil se conectará.

Por último, solo queda configurar la aplicación móvil. Para hacer esto, hay que situarse con la terminal dentro de la carpeta AppMovil y ejecutar los siguientes comandos:

npm install
npm install expo
npx expo install react-native-maps

Para la siguiente parte se requerirá como mínimo jdk-17.0.17+10 instalado en el ordenador. Si no se dispone de jdk 17, se puede descargar de forma local en Adoptium, por ejemplo. Si se dispone de permisos de administrador, se puede simplemente hacer sudo apt install java o sudo apt update java. De todas maneras, hay que configurar el home de Java. Se puede hacer usando los siguientes comandos:

```
export JAVA_HOME={ruta al sdk}
export PATH=$JAVA_HOME/bin:$PATH
```

Por último, se debe hacer npx expo run:android -d (dentro de la carpeta AppMovil) y elegir un dispositivo. Una vez hecho, y suponiendo que se ha configurado todo lo demás de forma correcta, lo único que quedaría por hacer es entrar en src/Main.java e iniciar el servidor.

Conclusión

Trabajando en este proyecto, el cual hicimos con solo tres miembros, y con miembros bastante inexpertos, hemos aprendido a trabajar las tecnologías Java, React-Native, TypeScript y SQL en entornos de trabajo realistas.

Aparte de eso, el concepto del proyecto es interesante y las relaciones con los clientes fueron creativas cuanto menos. A pesar de las dificultades, podemos decir que nos lo hemos pasado bien haciendo este proyecto y que al menos alguien del equipo ha aprendido algo.

Empezamos pensando que no seríamos capaces de cumplir con las expectativas, pero acabamos superando el listón. En conclusión, ha sido una experiencia interesante y hemos adoptado técnicas y formas de trabajar en un trabajo real que antes no teníamos, así como más habilidad para trabajar en grupo.