



Proyecto intermodular

Activi+

Nombre del alumno: Renaud Bronchart

Curso académico: DAM - Desarrollo de Aplicaciones Multiplataforma (segundo año)

Tutora/Tutor del proyecto:



ÍNDICE PAGINADO

1. Justificación del proyecto	3
2. Introducción	3
3. Objetivo	3
4. Desarrollo : teórica	4
4.1 Análisis	4
4.2.1 Aplicación móvil:	4
4.1.2 Aplicación web con Spring BOOT :	4
4.2 capa de presentación	5
4.2.1 Aplicación móvil:	5
4.2.1 Aplicación web con Spring BOOT :	5
5. Desarrollo : resultado	6
5.1 Aplicación Móvil	6
5.1.1 Arquitectura del proyecto	6
5.1.2 Firestore base (Acceso a datos)	9
5.1.3 Creación de una función para poder descargar una actividad en un PDF como una factura(Sistema de gestión empresarial)	11
5.1.4 Resultado final de nuestro proyecto móvil:	12
5.2 Resultado Aplicación Web SpringBoot	13
5.2.1 Arquitecta de aplicación	13
5.2.3 Función para manejar hilos (Programación de servicios y procesos)	14
5.2.4 Ejemplo de unas dependencias utilizadas (Desarrollo de interfaces):	16
5.2.5 Base de datos MySQL y su conexión (Acceso a datos)	16
5.2.6 Otro programa utilizado para el proyecto.	17
5.2.7 Resultado final de nuestra pagin Web con sus datos	17
6. Conclusiones	19
7. Bilbiographia	<i>Erreur ! Signet non défini.</i>

1. Justificación del proyecto

En nuestros días, muchas personas quieren hacer diferentes actividades que implican costos. Por tanto, no siempre tienen una herramienta útil que les permita guardar y controlar estos datos de manera organizada.

Este proyecto tiene como objetivo desarrollar una aplicación móvil y web que permite guardar estos datos. Esto permitirá planificar actividades de manera mas eficaz.

Este proyecto intermodular se pretende aplicar los conocimientos adquiridos en diversos módulos del segundo año del ciclo formativo de Desarrollo de Aplicaciones Multiplataforma (DAM), como Acceso a Datos, Desarrollo de Interfaces, Sistemas de Gestión Empresarial, Programación de Servicios y Procesos, y Programación Multimedia y Dispositivos Móviles.

2. Introducción

En la actualidad, existen diversas herramientas para almacenar información, pero no hay muchas que combinan de manera eficiente la capacidad de registrar el costo individual de cada actividad y calcular su costo total de forma automática.

Esta aplicación, diseñada para estar disponible en plataformas digitales, permitirá a los usuarios acceder a sus datos en cualquier momento y desde cualquier lugar.

La aplicación permitirá calcular automáticamente el precio de cada actividad y el total de todas las actividades guardadas. Además, los usuarios podrán ver todas las actividades y ordenarlas según el precio.

3. Objetivo

El objetivo principal de este proyecto es desarrollar dos aplicaciones intermodular (aplicación móvil y web).

Estas aplicaciones gestionaran listas de actividades y mostrara el precio total de todas las actividades registradas. Se actualizará el precio total si el usuario borra un modifica un precio de una actividad.

El usuario podría:

- Registrar cada actividad con el precio total de actividad
- Modificar cada actividad
- Borrar cada actividad
- Ver cada actividades y el precio total de cada actividades

4.1 Análisis

4.2.1 Aplicación móvil:

Esta aplicación móvil será desarrollada con el lenguaje Kotlin, y Firebase será utilizado como base de datos para poder almacenar los datos.

Vamos a implementar un CRUD que permite crear, leer, actualizar y borrar y permite gestionar las actividades de un usuario.

Esta aplicación será conectada a una base de datos (Firebase) para permitir guardar los datos.

Vamos a implementar la funcionalidad de descarga de facturas desde la aplicación, permitiendo a los usuarios generar y obtener documentos en formato PDF de cada actividad.

En la página de lista de las actividades, vamos a crear una función que permite calcular y enseñar el total de todas las actividades registradas

4.1.2 Aplicación web con Spring BOOT:

Esta aplicación web será desarrollado con HTML, CSS y Javascript por la parte Front End.

Por la parte back, el Lenguaje sera java utilizando herramienta Spring Boot y su framework Srping.

Vamos a desarrollar un CRUD con la implementación de anotaciones, y vamos a utilizar diferentes dependencias como Maven, para para la gestión de dependencias.

Para guardar los datos, una base de datos con MySQL será creado para hacer la relación con nuestra aplicación web.

Aplicación será diseñado en diferentes capas:

- Controlador
- Modelo
- capas de persistencia/repositor
- Servicio
- La parte Front

Vamos a implementar un código que permite manejar hilos y que permite calcular y enseñar el total de todas las actividades registradas.

4.2 capa de presentación

La capa de presentación, también conocida como capa de interfaz de usuario, es la interfaz gráfica de nuestro sistema, que proporciona la interacción entre los usuarios y el sistema.

A continuación, mostraremos el diseño preliminar de la parte visual de nuestras aplicaciones

4.2.1 Aplicación móvil:

En esta aplicación móvil, vamos a diseñar una aplicación que mostrara un menú que permite acceder a diferentes tareas como guardar o ver actividades.



4.2.1 Aplicación web con Spring BOOT :

Vamos a crear un diseño simple con HTML y CSS. Habrá Un formulario para guardar, modificar los datos. Y una tabla para mostrar, modificar y editar los datos.

Lista de Actividades					
ID	Nombre Actividad	Numero de Personas	Precio	Total Precio	
1	SKI	6	24 €	144 €	
2	VELO	10	6€	60 €	BORRAR EDITAR
3	MUSEO	5	100€	500€	BORRAR EDITAR
4	BOOTCAMP	30	400€	1200€	BORRAR EDITAR
5	FOOT	5	15€	75 €	BORRAR EDITAR

Precio Total
5000 €

Formulario	
Nombre :	Ski
Numero de personas:	10
Precio :	55 €
Agregar actividad	

5.1 Aplicación Móvil

Hemos hecho aplicación con Android Studio. El lenguaje utilizado para aplicación es Kotlin. Es uno de los mejores lenguajes de programación para desarrollar en Android. La sintaxis de Kotlin es muy similar a Java.

Jetpack Compose fue utilizado durante todo el proyecto de aplicación.

Jetpack Compose es un kit de herramientas moderno diseñado para simplificar el desarrollo de IU. Combina un modelo de programación reactivo con la concisión y facilidad de uso del lenguaje de programación Kotlin¹

5.1.1 Arquitectura del proyecto

Carpeta navegación:

Dentro de esta carpeta, vamos a crear diferentes clases para construir nuestra “capa de presentación UI”

A continuación, se puede ver las diferentes páginas de la carpeta.

AppNavigation.kt que tiene como rol la gestión de navegación y rutas. Eso va a permitir la gestión de navegación entre diferentes pantallas de la app.

```
@Composable // indica pertenece a Jetpack compose y define parte IU
fun AppNavigation() {
    val navigationController = rememberNavController() // para crear instancia del controlador navegacion
    NavHost(
        navigationController = navigationController,
        startDestination = AppScreens.PaginaBienvenida.ruta // para coger primera pagina app que es paginaBienvenida..
    ) {
        composable(AppScreens.PaginaBienvenida.ruta) {
            PaginaBienvenida(navigationController)
        }

        composable(AppScreens.MenuInicio.ruta) {
            MenuInicio(navigationController)
        }

        composable(AppScreens.CrearActividad.ruta) {
            CrearActividad(navigationController)
        }

        composable(AppScreens.ListaActividad.ruta) {
            ListaActividad(navigationController)
        }

        composable(AppScreens.EditarActividad.ruta) {
            EditarActividad(navigationController)
        }

        composable(AppScreens.BorrarActividad.ruta) {
            BorrarActividad(navigationController)
        }
    }
}
```

¹ Fuente: developer.android.com

También hemos creado un “sealed class” llamado **AppScreens** que van definir las rutas de navegación.

```
sealed class AppScreens(val ruta: String) {  
    object PaginaBienvenida: AppScreens(ruta: "PaginaBienvenida")  
    object MenuInicio: AppScreens(ruta: "MenuInicio")  
    object CrearActividad: AppScreens(ruta: "CrearActividad")  
    object EditarActividad: AppScreens(ruta: "EditarActividad")  
    object BorrarActividad: AppScreens(ruta: "BorrarActividad")  
    object ListaActividad: AppScreens(ruta: "ListaActividad")  
}
```

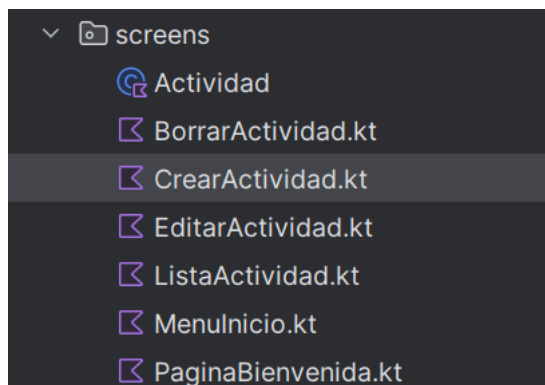
Carpeta navegación:

Dentro de esta carpeta, vamos a tener

- una data class de Actividad que servirá para almacenar datos.

```
data class Actividad(  
    val nombreActividad: String = "",  
    val precioPorActividad: String = "",  
    val personasPorActividad: String = "",  
    val precioTotalActividad: String = ""  
)
```

- unas clases de cada pagina para hacer el CRUD.





A continuación, vamos a ver unos ejemplos de código.

En la página crearActividad, vamos a implementar un formulario que permite al usuario ingresar informaciones para crear una nueva actividad.

Para manejar los datos ingresados, utilizaremos la función “remember” de Jetpack Compose que nos permite almacenar el estado de las variables.

```
var nombreActividad by remember { mutableStateOf( value: "" ) }

OutlinedTextField(value = nombreActividad, onValueChange = { nombreActividad = it },
    label = { Text( text: "Nombre Actividad " ) }
)

Spacer(modifier = Modifier.size(16.dp))

var precioPorActividad by remember { mutableStateOf( value: "" ) }
OutlinedTextField(
    value = precioPorActividad,
    onValueChange = {
        if (it.isEmpty() || it.all { char -> char.isDigit() }) {
            precioPorActividad = it
        }
    },
    label = { Text( text: "Precio" ) },
    keyboardOptions = KeyboardOptions.Default.copy(keyboardType = KeyboardType.Number)
)
```

Luego, vamos a tener que interactuar con la base de datos FireBase Firestore para guardar datos en una colección llamada actividades.

```
val db = FirebaseFirestore.getInstance() //instancia de Firestore para realizar operaciones como leer o escribir datos
val coleccion = "actividades"

var mensajeConfirmacion by remember { mutableStateOf( value: "" ) }

val dato = hashMapOf() // Crea un mapa de datos clave-valor que se enviará a Firestore
    "nombreActividad" to nombreActividad,
    "precioPorActividad" to precioPorActividad,
    "personasPorActividad" to personasPorActividad,
    "precioTotalActividad" to precioTotalActividad.toString()
)

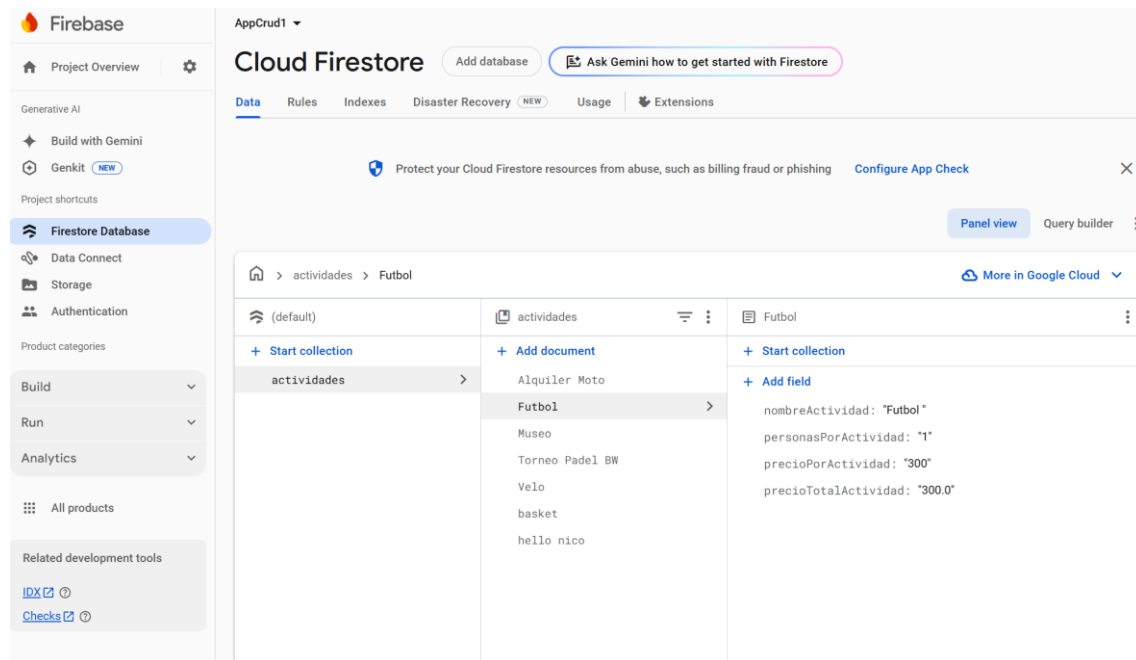
Button( // Botón para guardar datos
    onClick = {
        db.collection(coleccion) // acceder a la coleccion que hemos creado actividades
            .document(nombreActividad)
            .set(dato)
            .addOnSuccessListener {
                mensajeConfirmacion = "Datos guardados correctamente"

                nombreActividad = ""
                precioPorActividad = ""
                personasPorActividad = ""
            }
    }
)
```


5.1.2 Firestore base (Acceso a datos)

Cloud Firestore es una base de datos NoSQL alojada en la nube a la que pueden acceder tus apps para Apple, Android y la Web directamente desde los SDK nativos. Cloud Firestore también está disponible en los SDKs nativos de Node.js, Java, Python, Unity, C++ y Go, además de las APIs de REST.²

Vista de la base de datos en internet.



Se puede ver la colección que hemos creado que se llama actividades con cada documento que corresponde a una actividad. Dentro de este documento, hay un objeto JSON con los datos (campos) guardados.

Hay tener en cuenta que hay que configurar la parte "Rules" para poder conectarse y seguir viendo los datos.

Por ejemplo, a nosotros, hemos tenido un problema para obtener los datos porque timestamp.date había pasado de tiempo.

² FUENTE : [Firestore](#) | [Firebase](#)



Para poder relacionar la base de datos con nuestro proyecto en Android Studio, hemos seguido estas instrucciones³ y hemos añadido el archivo `Google.service.json`, las dependencias y la implantación en el proyecto como explicado en la página de Firebase.

³ Fuente : AppCrud1 - Add app - General -

Add Firebase to your Android app

✓ Register app
Android package name: com.example.appcrud, App nickname: AppCrud

2 Download and then add config file

Instructions for Android Studio below | [Unity](#) [C++](#)

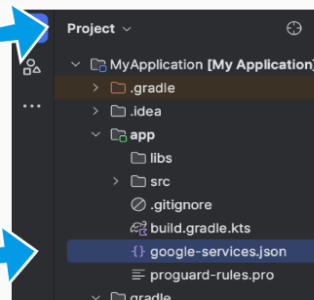
Download google-services.json

Switch to the Project view in Android Studio to see your project root directory.

Move your downloaded `google-services.json` file into your module (app-level) root directory.



Next



3 Add Firebase SDK

4 Next steps



5.1.3 Creación de una función para poder descargar una actividad en un PDF como una factura⁴⁵(Sistema de gestión empresarial)

Para poder integrar el curso de Sistema de Gestión Empresarial, hemos pensado a hacer una clase que permite la descarga de una actividad. El documento descargado, será un documento PDF.

Para realizar esta función, la ayuda de la documentación de página de Android developer y Chatgpt fue utilizado como referenciado.

```
class DocumentUtil {  
    // creacion funcion para crear un PDF a partir de una actividad  
    // Context : se hace referencia a un objeto que se necesita para acceder a todos los recursos  
    // tipo file  
    fun createPdfFile(context : Context, actividad : Actividad): File? {  
        try {  
            val pdfDirectory = File(context.getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS), child: "Actividades")  
            if (!pdfDirectory.exists()) { // si el directorio no existe, lo crea  
                pdfDirectory.mkdirs()  
            }  
            // crear archivo dentro del directorio Actividades  
            val pdfFile = File(pdfDirectory, child: "${actividad.nombreActividad}.pdf")  
  
            val writer = PdfWriter(pdfFile) // Crea un escritor (writer) de PDF que escribirá en el archivo pdfFile  
            val pdfDocument = com.itextpdf.kernel.pdf.PdfDocument(writer) // escribir el contenido en el PDF  
            val document = Document(pdfDocument)  
  
            document.add(Paragraph(text: "Detalles de Actividad"))  
            document.add(Paragraph(text: "Nombre de Actividad: ${actividad.nombreActividad}"))  
            document.add(Paragraph(text: "Precio de Actividad: ${actividad.precioPorActividad} €"))  
            document.add(Paragraph(text: "Cantidad de Personas: ${actividad.personasPorActividad}"))  
            document.add(Paragraph(text: "Precio Total: ${actividad.precioTotalActividad} €"))  
  
            document.close()  
  
            return pdfFile // devolver el archivo PDF  
        } catch (e: Exception) {  
            e.printStackTrace()  
            return null  
        }  
    }  
}
```

⁴ Fuente: [Cómo imprimir documentos personalizados](#) | Android Developers

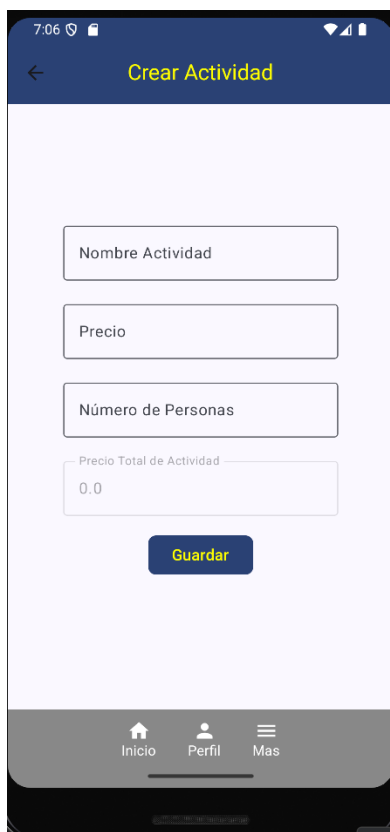
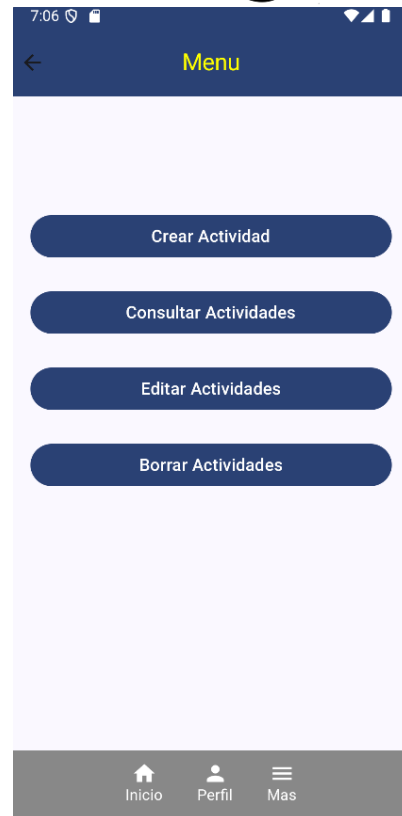
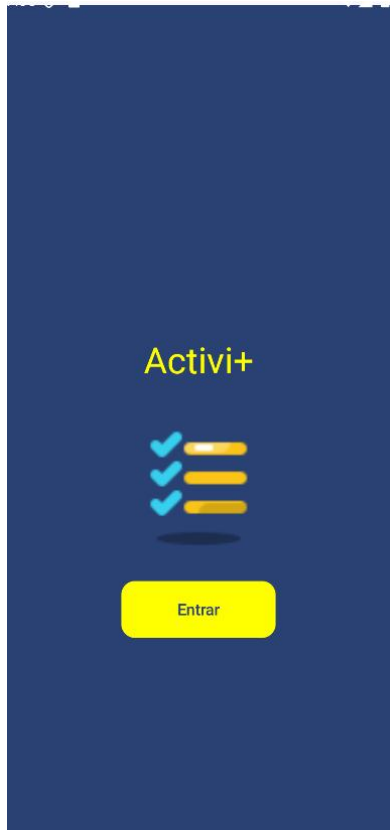
⁵ Fuente: <https://chatgpt.com/>

5.1.4 Resultado final de nuestro proyecto móvil:



INSTITUTO
NEBRIJA

Formación
Profesional










5.2 Resultado Aplicación Web SpringBoot

5.2.1 Arquitecta de aplicación

A continuación, vamos a ver las diferentes capas y sus anotaciones (Desarrollo de interfaces)

- >  com.example.demo
- >  com.example.demo.controller
- >  com.example.demo.model
- >  com.example.demo.repository
- >  com.example.demo.service

Controller: Gestionar las solicitudes HTTP, interactuar con los servicios.

```
// Endpoint para obtener una actividad por su ID
@GetMapping("/actividades/{id}")
public Actividad obtenerActividadPorId (@PathVariable Long id) {
    return actividadService.obtenerViaje(id);
}

// Endpoint para agregar una nueva actividad // RequestBody, usado para convertir
// el cuerpo de la solicitud a un objeto java
@PostMapping("/actividades")
public void agregarActividad(@RequestBody Actividad actividad) {
    actividadService.agregarActividad(actividad);
}

// Endpoint para eliminar una nueva actividad con su ID
// PathVariable en SpringBoot permite recibir parametros desde la URL
@DeleteMapping("/actividades/{id}")
public void eliminarActividad(@PathVariable Long id) {
    actividadService.eliminarActividad(id);
}
```

Model: Representa las entidades(los datos) de aplicación

```
@Entity // Mapear clases Java a tablas de la base de datos
public class Actividad {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id; // Identificador único de la actividad
    private String nombre; // Nombre de la actividad
    @Column(name = "num_pers") // Dar el nombre de la columna en la BDD mysql
    private int numPers;
    private double precio; // Precio por persona

    @Column(name = "total_precio")
    private double totalPrecio; // Prix total calculé comme numPers * precio
}
```

Repository: Capa que interactúa directamente con la base de datos



```
//La anotación @Repository indica a Spring que esta interfaz es un componente de acceso a datos
//permitiendo que Spring gestione la inyección de dependencias y maneje excepciones específicas de persistencia.
@Repository
public interface ActividadRepository extends JpaRepository<Actividad, Long> {
    List<Actividad> findByViajeId(Long viajeId);
}
```

Service: Contener la lógica de negocio

```
@Service // Servicio para gestionar la lógica de las actividades
public class ActividadService {

    private final ActividadRepository actividadRepository;

    // Constructor de la clase, que recibe el repositorio para interactuar con la base de datos
    public ActividadService(ActividadRepository actividadRepository) {
        this.actividadRepository = actividadRepository;
    }

    // Método para retornar todas las actividades
    public List<Actividad> retornarActividades() {
        return actividadRepository.findAll(); // Devuelve todas las actividades almacenadas
    }
}
```

5.2.2 Función para manejar hilos (Programación de servicios y procesos)

Eso va a permitir ejecutar tareas en paralelo y si hay muchas actividades, optimizar el tiempo de procesamiento.

Para poder utilizar la función con hilos, hemos creado una primera función que permite tener el precio total de una actividad según el precio y la cantidad de personas en la actividad.

Esta disponible en el modelo Actividad.

```
public int getNumPers() {
    return numPers;
}

public void setNumPers(int numPers) {
    this.numPers = numPers;
    recalcularTotalPrecio(); // Actualiza el total cada vez que se modifica el número de personas
}

public double getPrecio() {
    return precio;
}

public void setPrecio(double precio) {
    this.precio = precio;
    recalcularTotalPrecio(); // Actualiza el total cada vez que se modifica el precio
}

//
public void recalcularTotalPrecio() {
    this.totalPrecio = this.numPers * this.precio;
}

public double getTotalPrecio() {
    return totalPrecio;
}

public void setTotalPrecio(double totalPrecio) {
    this.totalPrecio = totalPrecio;
}
```

de

Para tener el precio total de todas las actividades, primero hemos tenido que recuperar todas las actividades desde la base de datos.



Luego, un `ExecutorService`, para gestionar y ejecutar tareas en paralelo hemos creado un `ExecutorService` con un pool de hilos con el tamaño de la lista de actividades.

`Callable` es igual a `Runnable` pero una de la diferencia es que puede devolver un valor.

`Future` nos permite tener el resultado de una tarea que se hace en segundo plano.

Nos permite acceder fácilmente a un resultado sin complicar la gestión de hilos.

Nos permite el manejo de tareas asincrónicas.

```
// Método para calcular el precio total de todas las actividades
public double precioTotalActividades() throws InterruptedException, ExecutionException {

    // Recupera todas las actividades desde la base de datos
    List<Actividad> actividades = actividadRepository.findAll();

    // Crea un ExecutorService con un número de threads igual al tamaño de la lista de actividades
    ExecutorService executorService = Executors.newFixedThreadPool(actividades.size());

    // Lista para almacenar los resultados de las tareas futuras (Future)
    List<Future<Double>> futures = new ArrayList<>();

    // Recorre todas las actividades y envía una tarea para calcular su precio total
    for (Actividad actividad : actividades) {
        // Utiliza Callable para cada tarea, que devuelve el totalPrecio de la actividad
        // Callable en vez de Runnable, callable puede devolver en valor ( total precio aqui)
        //
        Callable<Double> task = () -> actividad.getTotalPrecio();
        futures.add(executorService.submit(task)); // Envía la tarea al pool de threads
    }

    // Espera a que todas las tareas terminen y calcula el total general
    double totalGeneral = 0;
    for (Future<Double> future : futures) {
        totalGeneral += future.get(); // Recupera el resultado de cada tarea
    }

    System.out.println("Prix total des activités : " + totalGeneral);

    // Cierra el ExecutorService para liberar los recursos del sistema
    executorService.shutdown();

    // Devuelve el total general acumulado de todas las actividades
    return totalGeneral;
}
```

5.2.3 Ejemplo de unas dependencias utilizadas (Desarrollo de interfaces):

Spring Web: Permite crear controladores que gestionan solicitudes HTTP(GET,POST,PUT,DETE)

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Spring JPA: Para manejar la persistencia de datos de manera eficiente. Facilita las operaciones CRUD

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

MySQLDriver: dependencia que permite a aplicación a conerctarse a una base de datos MySQL. Es como un puente entre la pp y el servidor de BDD.

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <scope>runtime</scope>
</dependency>
```

5.2.4 Base de datos MySQL y su conexión (Acceso a datos)

En nuestro proyecto hemos creado un archivo application.properties, que es un archivo clave en el proyecto, y que se utiliza para definir propiedades de configuración o definir la credenciales y URL de conexión a la BDD.

```
1 spring.application.name=ProyectoPintura
2 spring.datasource.url=jdbc:mysql://localhost:3306/viaje?useSSL=false
3 spring.datasource.username=root
4 spring.datasource.password=
5 spring.jpa.hibernate.ddl-auto=update
```

Por un tema de seguridad, seria mejor utilizar un username otro que root y una contraseña.

Result Grid						
Filter Rows:						
	id	nombre	num_pers	precio	total_precio	viaje_id
▶	1	Velo	6	24	144	10
	2	Museo	20	6	120	10
	3	Velo	5	22	110	12
	13	test	33	22	726	11
	14	Ski	100	1000	100000	10
	15	BootCamp	100	4000	400000	10
*	NULL	NULL	NULL	NULL	NULL	NULL

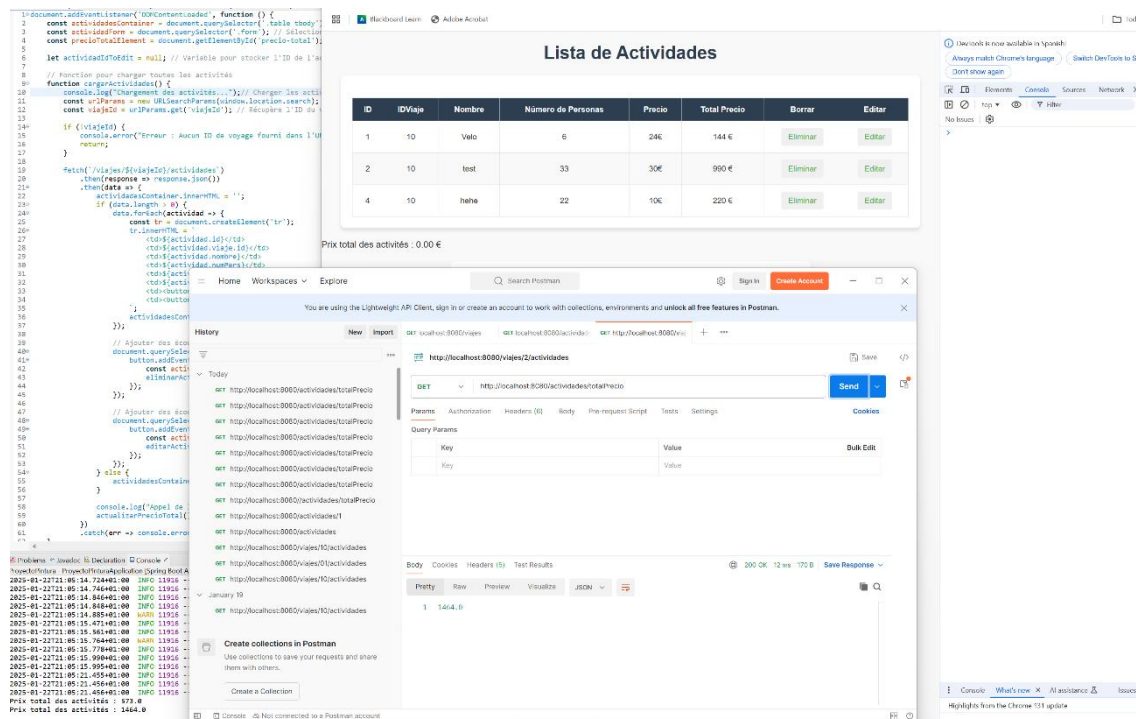
5.2.5 Otro programa utilizado para el proyecto.

Xamp : es una distribución de software que proporciona un entorno de desarrollo de servidores como MySQL, Apache, etc.. Para nuestro proyecto, únicamente MySQL fue utilizado.

MySQL: hemos creado a un local host con una dirección y un Port 3306, que va a permitir la conexión nuestro programa y hacer la interacción. No hemos puesto ninguna contraseña

Postman : es una herramienta de colaboración y desarrollo que permite a los desarrolladores interactuar y probar el funcionamiento de servicios web y aplicaciones. proporcionando una interfaz gráfica intuitiva y fácil de usar para enviar solicitudes a servidores web y recibir las respuestas correspondientes⁶

Para poder ver las diferentes peticiones de nuestro proyecto web, hemos utilizado Postman



The image shows a web browser window displaying a 'Lista de Actividades' (List of Activities) page. The page contains a table with the following data:

ID	IDViaje	Nombre	Número de Personas	Precio	Total Precio	Borrar	Editar
1	10	Velo	6	24€	144 €	Eliminar	Editar
2	10	test	33	30€	990 €	Eliminar	Editar
4	10	hehe	22	10€	220 €	Eliminar	Editar

Below the table, it says 'Prix total des activités : 0.00 €'.

Overlaid on the browser is the Postman API client interface. It shows a collection of requests for the 'http://localhost:3000/actividades' endpoint. The selected request is a GET request to 'http://localhost:3000/actividades/totalPrecio'. The response is shown in the bottom right, indicating a 200 OK status with a response time of 12ms and a body of '1444.0'.

5.2.6 Resultado final de nuestra pagin Web con sus datos

⁶ Fuente : ¿Qué es Postman? ¿Cuáles son sus principales ventajas? - Formadores IT

Lista de Actividades

ID	IDViaje	Nombre	Número de Personas	Precio	Total Precio	Borrar	Editar
1	10	Velo	6	24€	144 €	Eliminar	Editar
2	10	Museo	20	6€	120 €	Eliminar	Editar
14	10	Ski	100	1000€	100000 €	Eliminar	Editar
15	10	BootCamp	100	4000€	400000 €	Eliminar	Editar

Precio Total (€)
500264.00

Nombre:

Número de Personas:

Precio:

Agregar Actividad

6. Conclusiones

Participar y terminar este proyecto fue un reto importante en muchos sentidos y una satisfacción porque significa que hemos podido poner en aplicación la mayoría de los conceptos aprendidos en clases.

La realización del proyecto no fue una tarea fácil debido a los diferentes obstáculos encontrados como la gestión de tiempo y poner en práctica los conocimientos aprendidos.

Durante el proceso, hemos podido aprender, comprender y perfeccionar más los conocimientos.

En conclusión, la realización de este proyecto nos ha permitido aprender a manejar un proyecto con sus problemas, y a meter en práctica los conocimientos aprendidos.

7. Bibliographia

Para la realización de esta memoria se han consultado las siguientes fuentes de contenido.

- [Firestore | Firebase](#)
- [developer.android.com](#)
- [Cómo imprimir documentos personalizados | Android Developers](#)
- <https://chatgpt.com/>