



<b>Nombre de la práctica</b>	<b>Normalización de Base de Datos</b>			<b>No.</b>	<b>1</b>
<b>Asignatura:</b>	<b>Fundamentos de Bases de Datos</b>	<b>Carrera:</b>	<b>Ingeniería en Sistemas Computacionales</b>	<b>Duración de la práctica (Hrs)</b>	<b>8 horas</b>

**NOMBRE DEL ALUMNO:** Julieta Sanchez Mendoza, Ángel Jesús Santiago Hernández, Magali Valencia Clemente  
**GRUPO:**3401

**Encuadre con CACEI:** Registra el (los) atributo(s) de egreso y los criterios de desempeño que se evaluarán en esta práctica.

No. atributo	Atributos de egreso del PE que impactan en la asignatura	Criterios de desempeño	
2	El estudiante diseñará esquemas de trabajo y procesos, usando metodologías congruentes en la resolución de problemas de ingeniería en sistemas computacionales	1	Identifica metodologías y procesos empleados en la resolución de problemas
		2	Diseña soluciones a problemas, empleando metodologías apropiadas al área
3	El estudiante plantea soluciones basadas en tecnologías empleando su juicio ingenieril para valorar necesidades, recursos y resultados esperados.	1	Emplea los conocimientos adquiridos para el desarrollar soluciones
		2	Analiza y comprueba resultados

## II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):

Laboratorio de cómputo y equipo de cómputo personal.

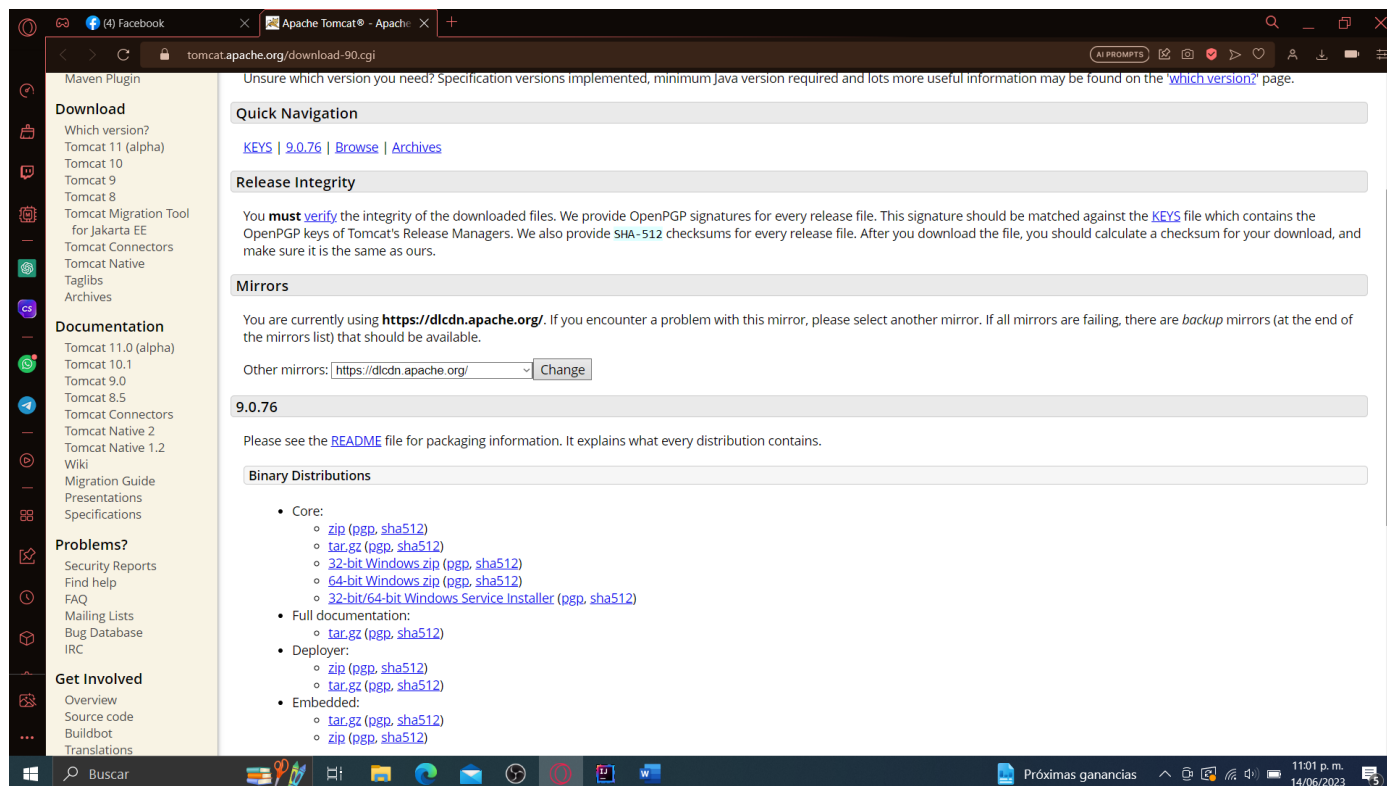
## III. Material empleado:

- Equipo de cómputo

## IV. Desarrollo de la práctica:

### UNIDAD 4

Como primer paso, debemos instalar tomcat desde su pagina



Seleccionamos el archivo.zip

Posteriormente descargamos e instalamos [xampp](#).



## XAMPP Apache + MariaDB + PHP + Perl

### ¿Qué es XAMPP?

XAMPP es el entorno más popular de desarrollo con PHP

XAMPP es una distribución de Apache completamente gratuita y fácil de instalar que contiene MariaDB, PHP y Perl. El paquete de instalación de XAMPP ha sido diseñado para ser increíblemente fácil de instalar y usar.



## XAMPP

### Descargar

Pulsa aquí para otras versiones



### XAMPP para Windows

8.2.4 (PHP 8.2.4)



### XAMPP para Linux

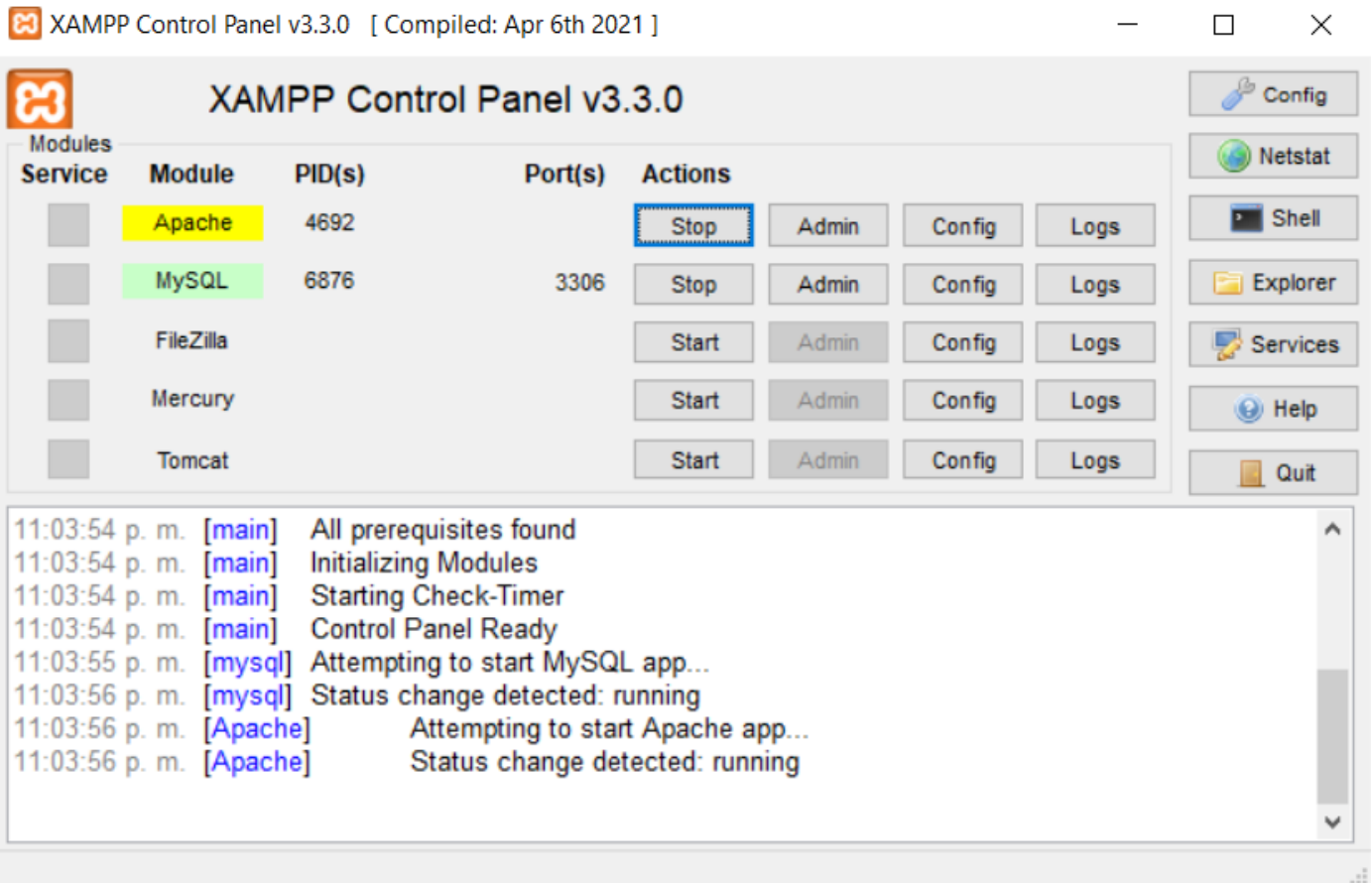
8.2.4 (PHP 8.2.4)



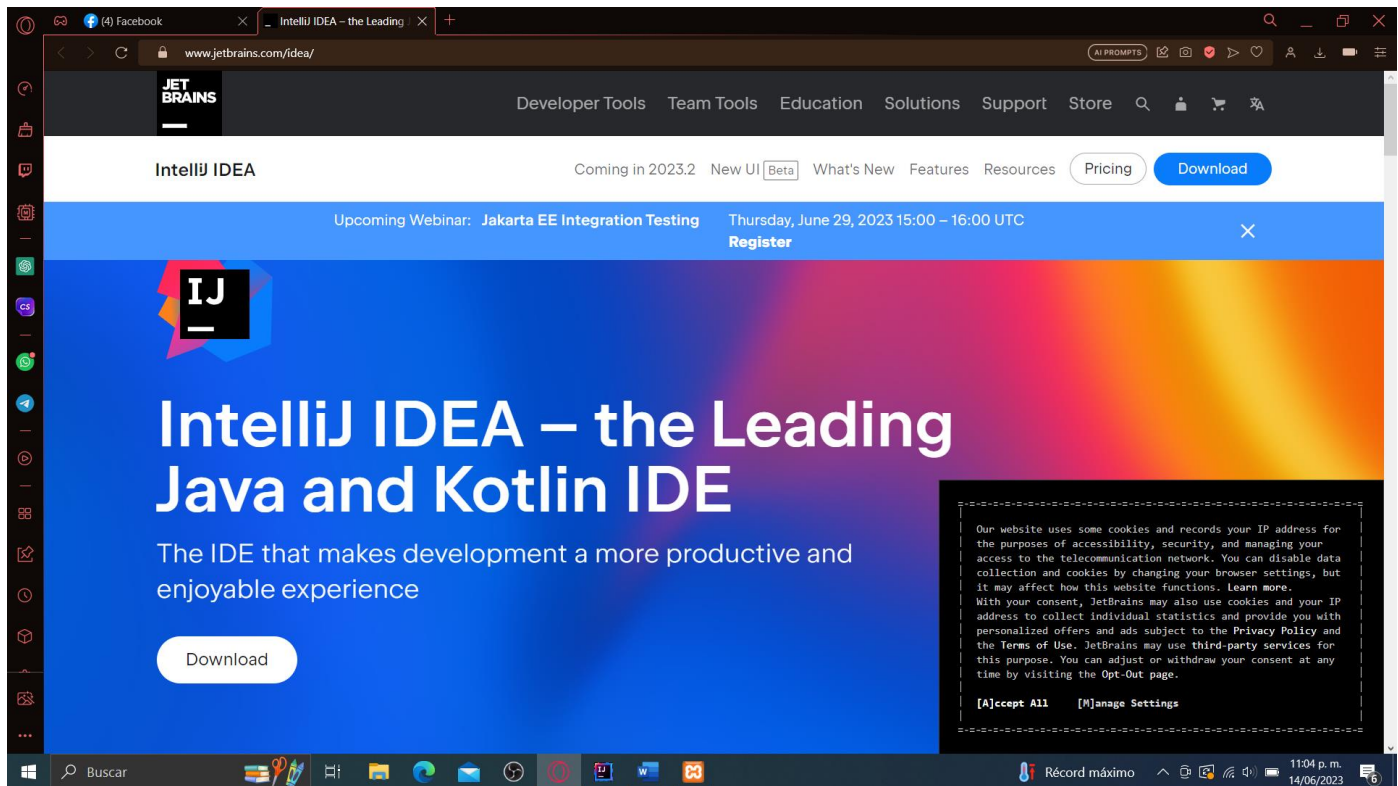
### XAMPP para OS X

8.2.4 (PHP 8.2.4)

Activar MySQL en xampp, si se pone verde funciona, naranja puede tener problemas y rojo no funciona.

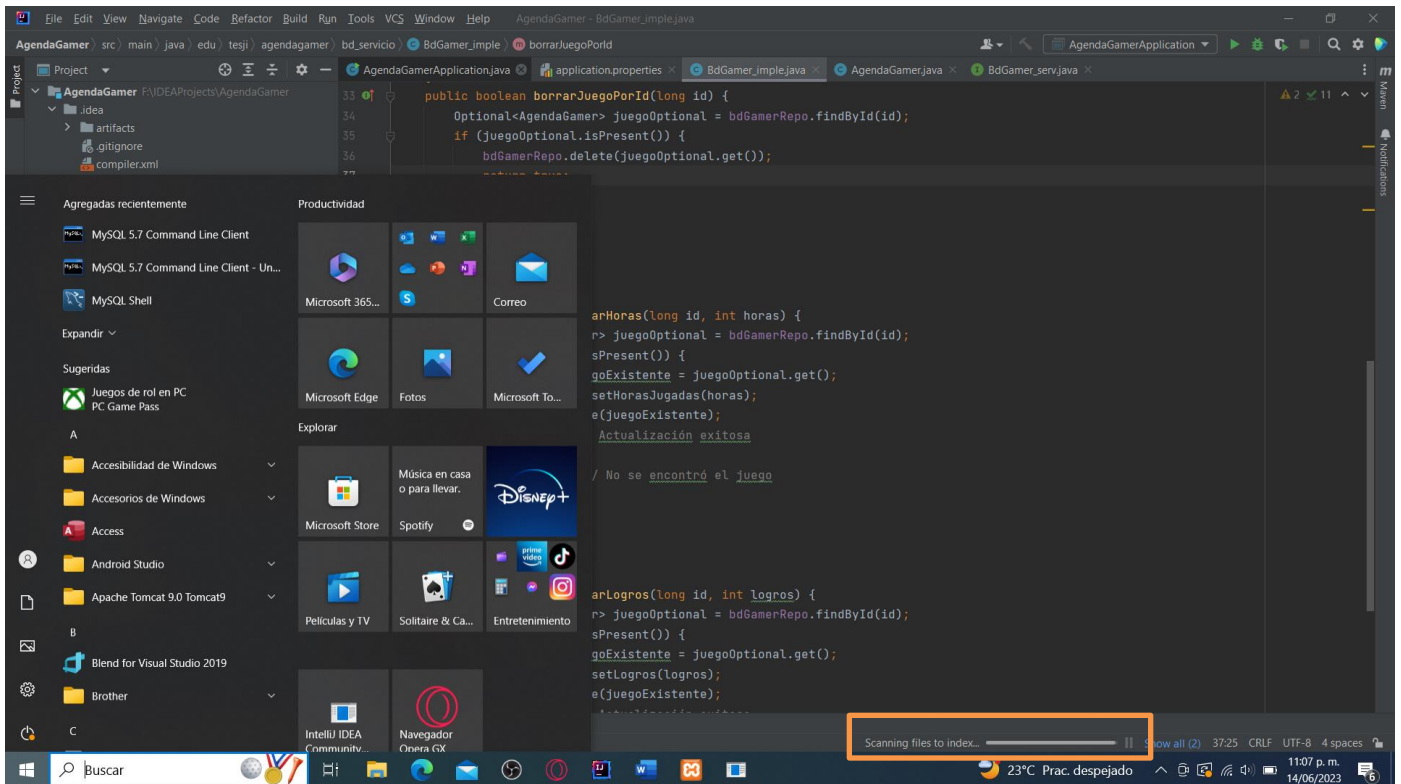
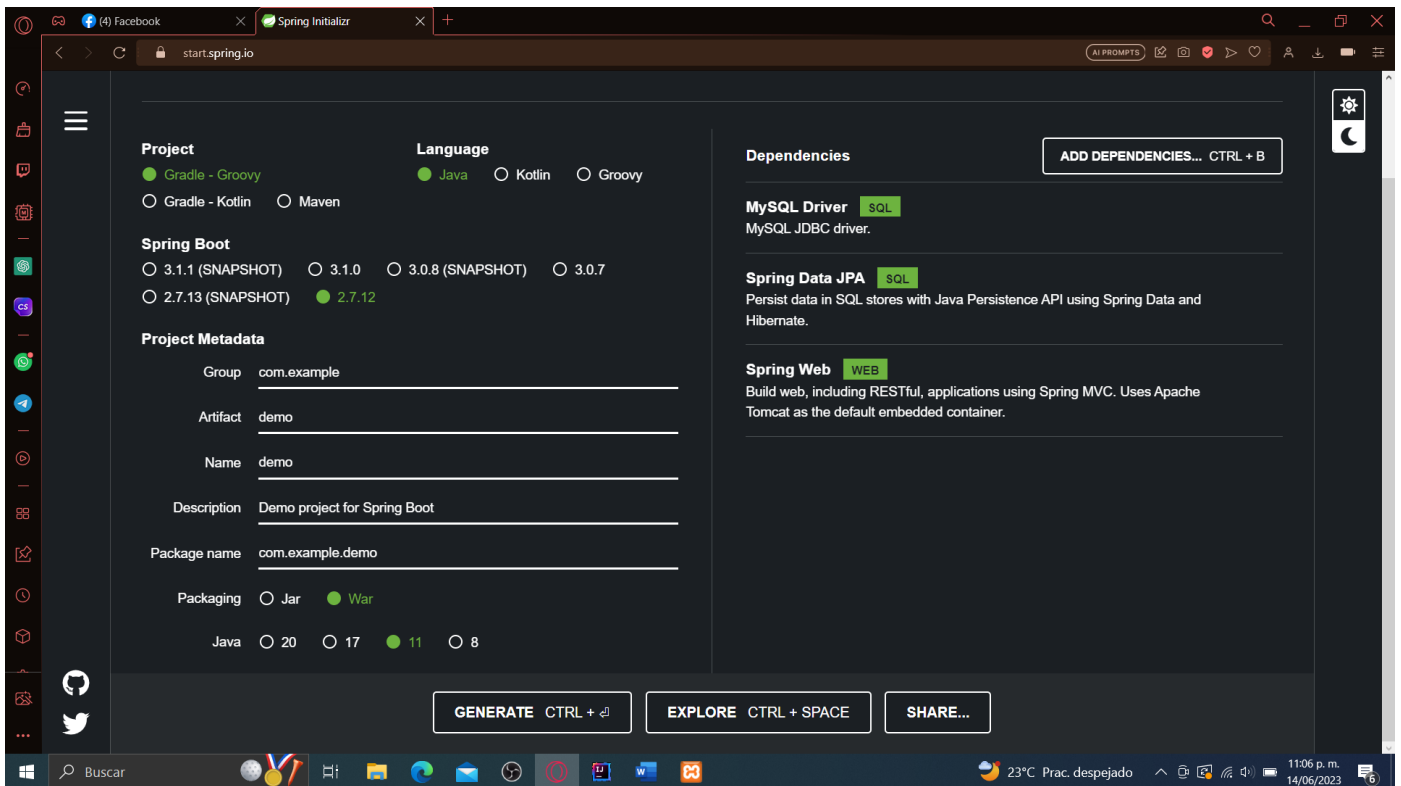


Descargamos IntelliJ que será nuestro gestor y ambiente para desarrollar la api

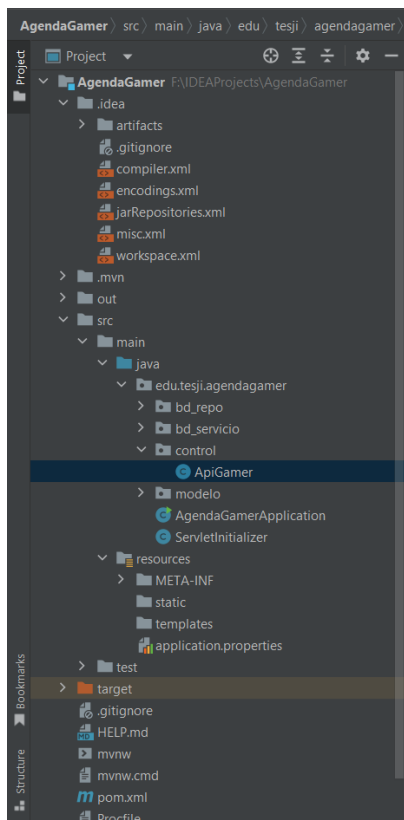


Vamos a [start.spring.io](https://start.spring.io) y configuramos nuestro proyecto:

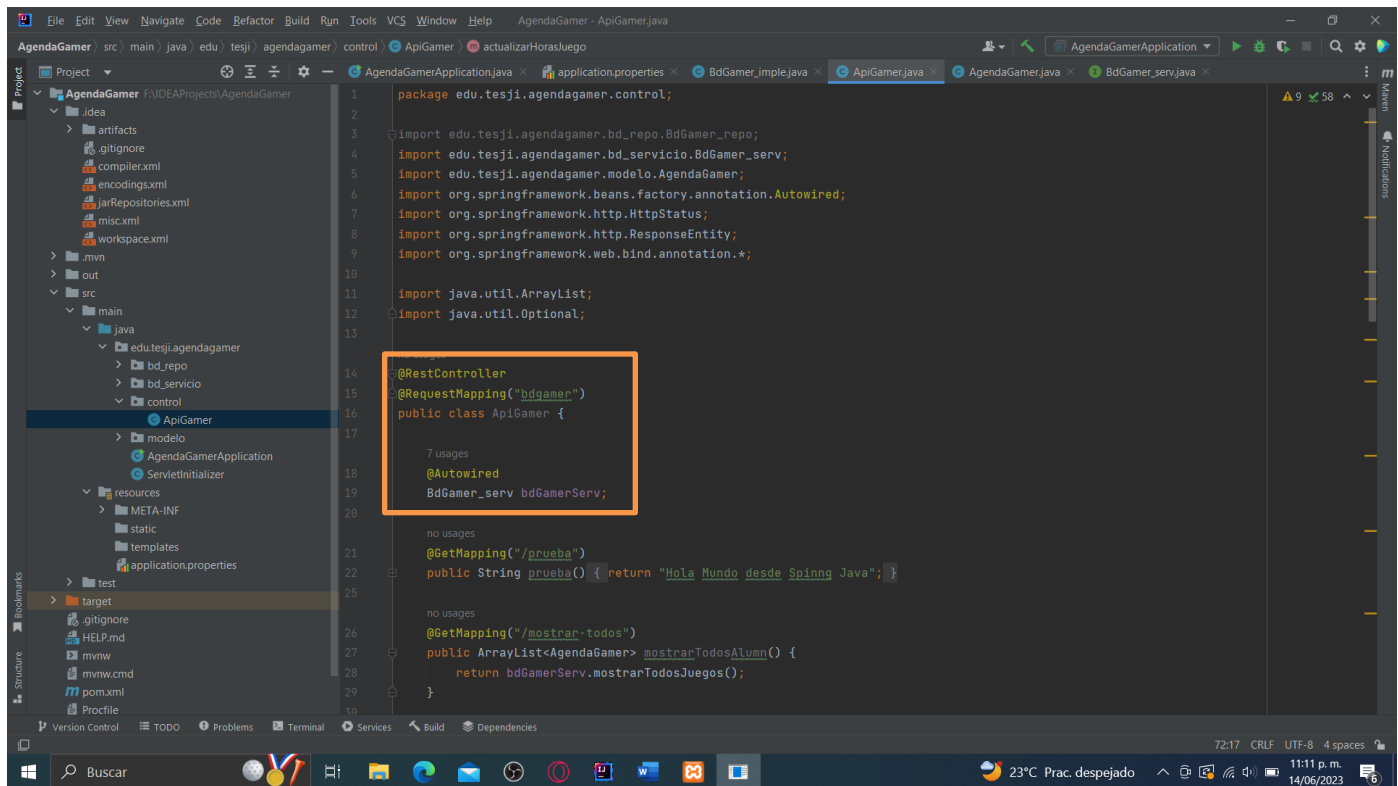
1. En [spring](#) seleccionar Maven y java, el spring boot 2.7, y java 20,17,11 o 8.
2. Inyectar las dependencias de spring web, Spring Data JPA y MySQL driver.
3. Descargar el archivo y descomprimirlo, y abrir la carpeta en IntelliJ con una buena conexión de internet para que se descarguen las dependencias.



Abrir el archivo pom.xml y documentar la dependencia de data JPA que conecta con la BD y la documentamos para evitar errores, creamos el paquete controller en src\main\java y creamos una nueva clase llamada ApiGamer (vamos a crear 4 peticiones, put, delete, get y post).



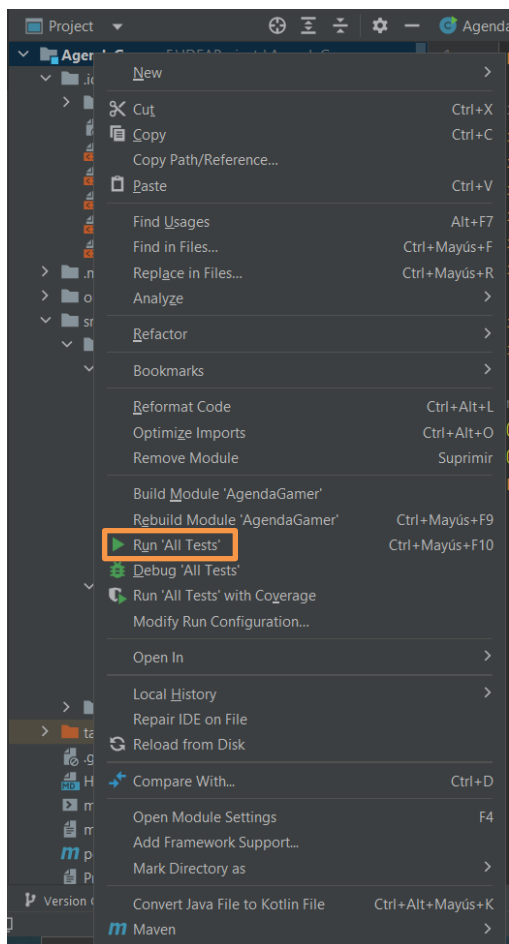
1. En la clase agregamos el RestController para que la clase tome la forma de un servicio de api rest y RequestMapping("/bdgamer") para exponer la ruta con una ruta, Crear el método de prueba para una solicitud get, para que tenga el funcionamiento de una api tiene que llevar las anotaciones de @GetMapping("/prueba") para exponer el método como una solicitud get con la ruta /apisic/prueba.



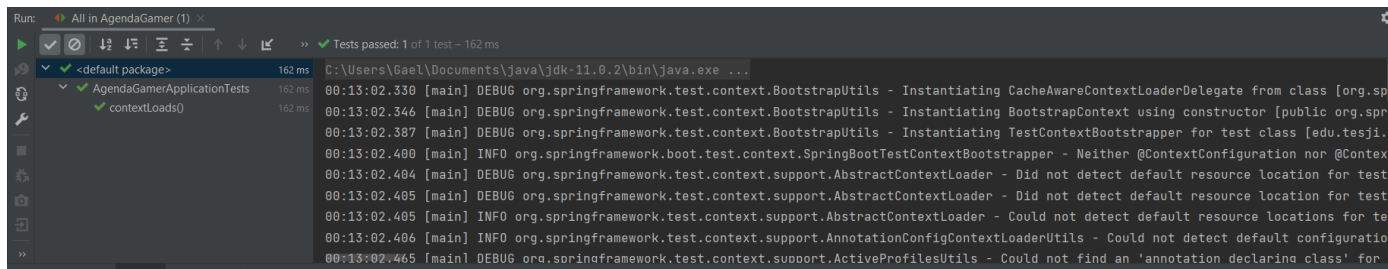
```
1 package edu.tesji.agendagamer.control;
2
3 import edu.tesji.agendagamer.bd_repo.BdGamer_repo;
4 import edu.tesji.agendagamer.bd_servicio.BdGamer_serv;
5 import edu.tesji.agendagamer.modelo.AgendaGamer;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.http.HttpStatus;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.web.bind.annotation.*;
10
11 import java.util.ArrayList;
12 import java.util.Optional;
13
14 @RestController
15 @RequestMapping("bdgamer")
16 public class ApiGamer {
17
18     7 usages
19     @Autowired
20     BdGamer_serv bdGamerServ;
21
22     no usages
23     @GetMapping("/prueba")
24     public String prueba() { return "Hola Mundo desde Spring Java"; }
25
26     no usages
27     @GetMapping("/mostrar-todos")
28     public ArrayList<AgendaGamer> mostrarTodosAlumn() {
29         return bdGamerServ.mostrarTodosJuegos();
30     }
31 }
```

Nos vamos a nuestro proyecto, damos click derecho run all test para testear la aplicación y ver si no hay errores. No inicia el servidor de tomcat en el puerto 8080 hay que configurar los módulos de tomcat en modo **compile**. Ir a project settings en modules y buscar los de org.apache.tomcat y cambiar de provided a compile y aplicar.



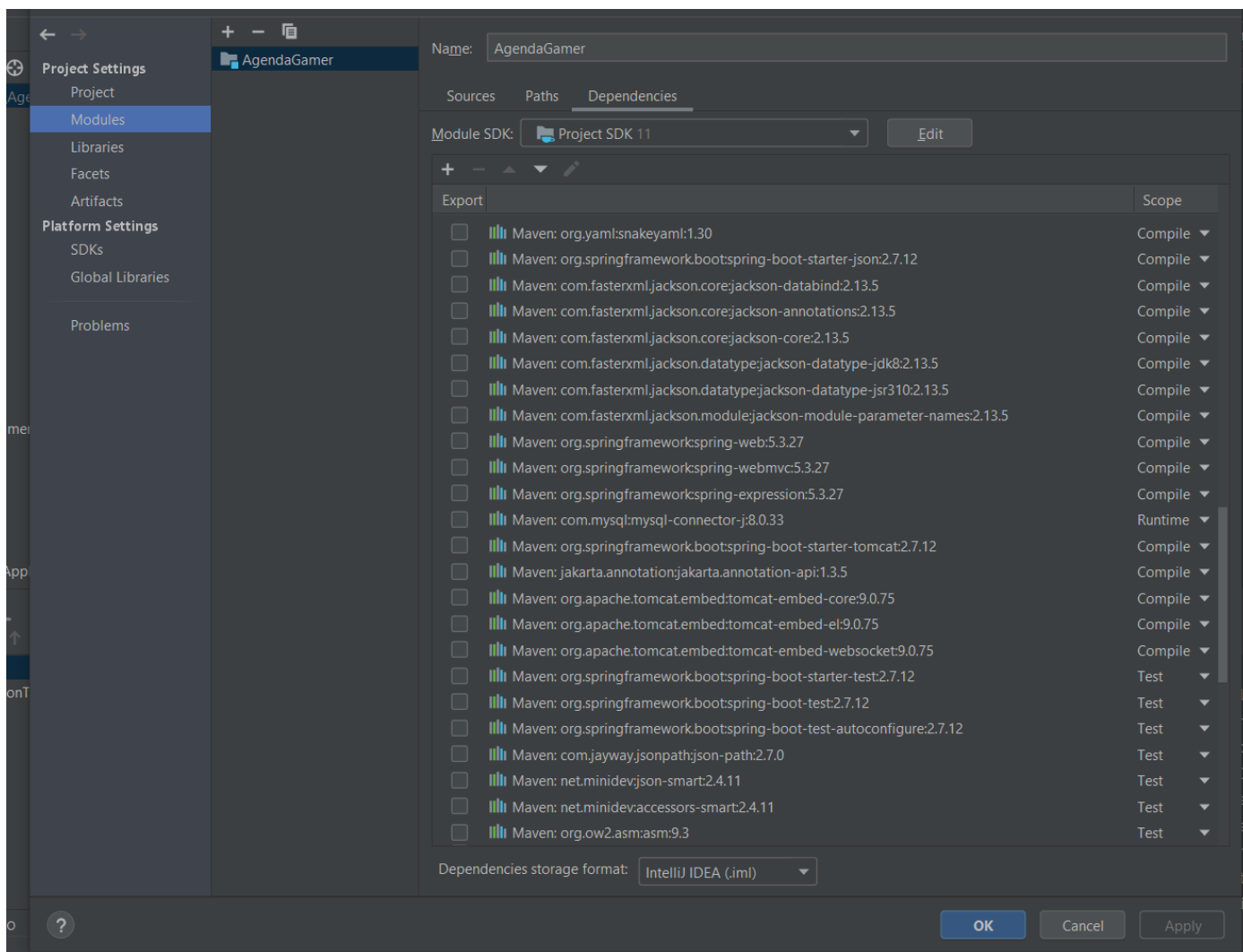


En mi caso si inicializa

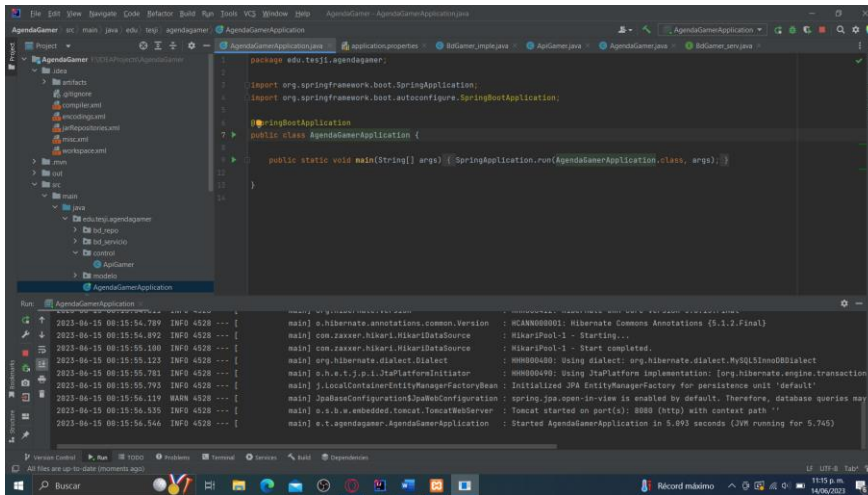


Si no fuera así habría que hacer el cambio a compile:

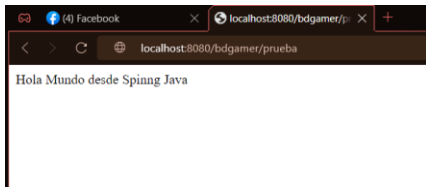




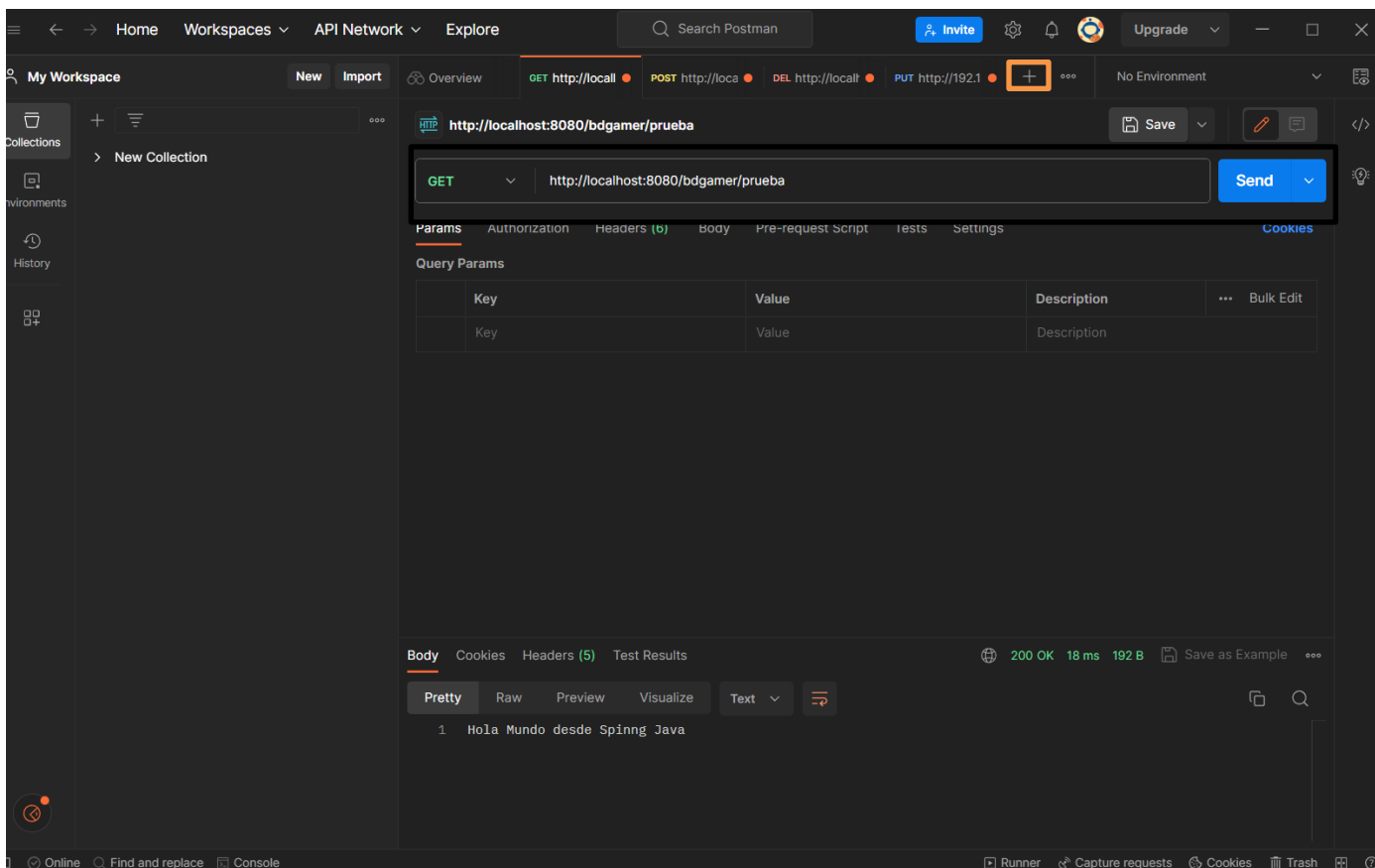
1. Abrir el application que tiene el main.



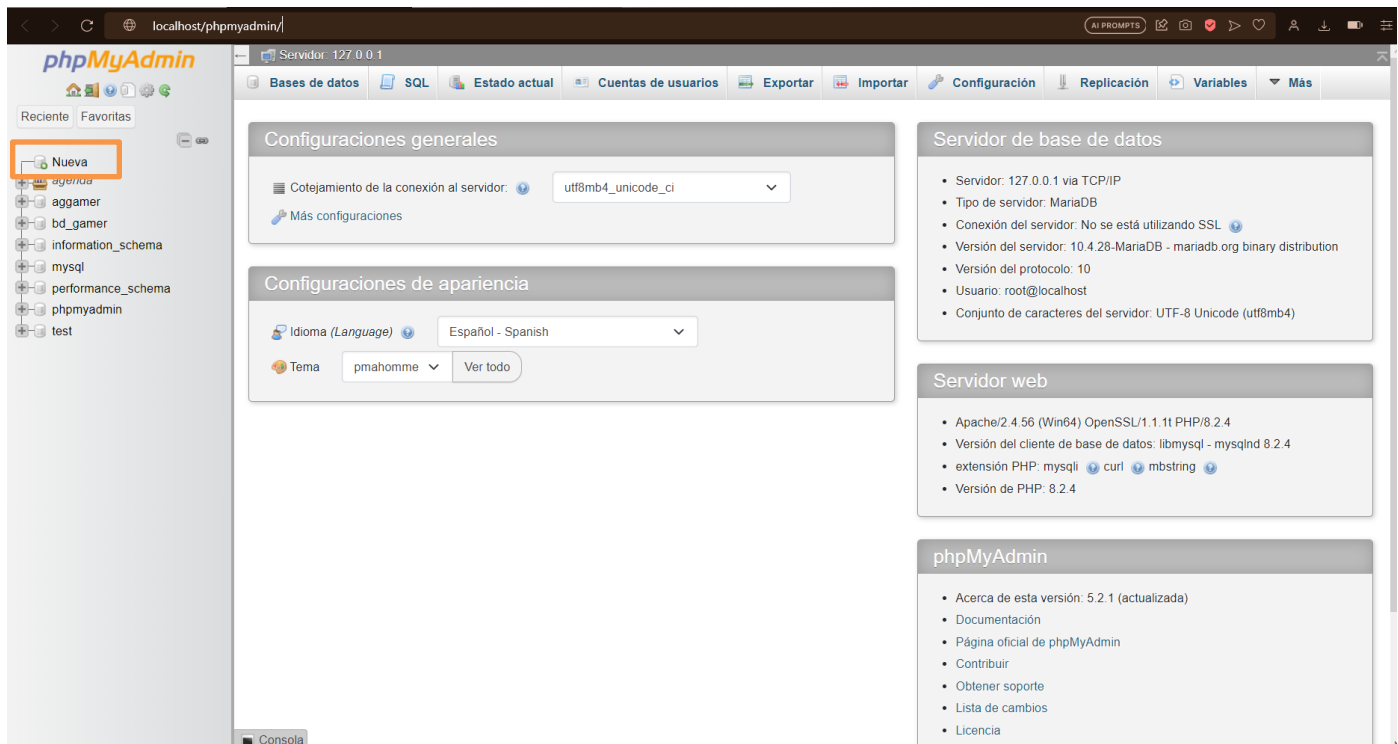
## 2. Testear desde el navegador solo url.



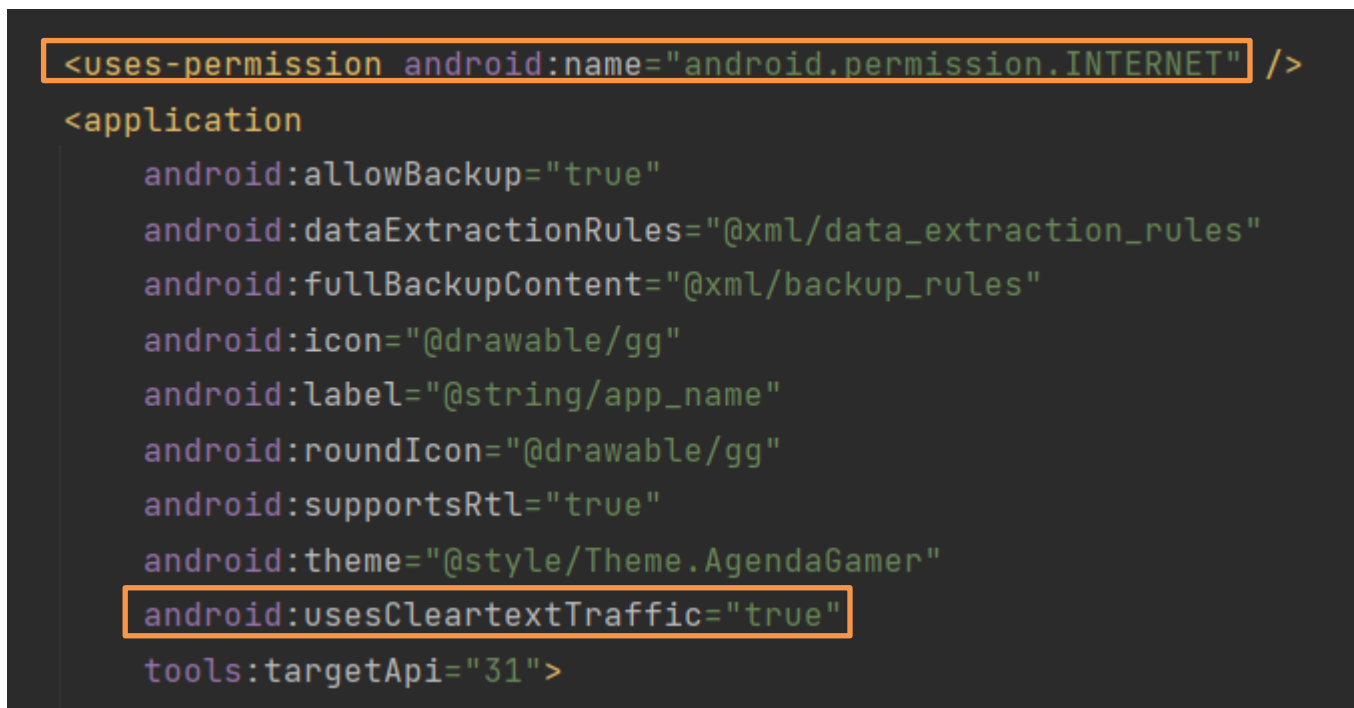
## 4. En postman nueva pestaña con petición get y pegas la ruta.



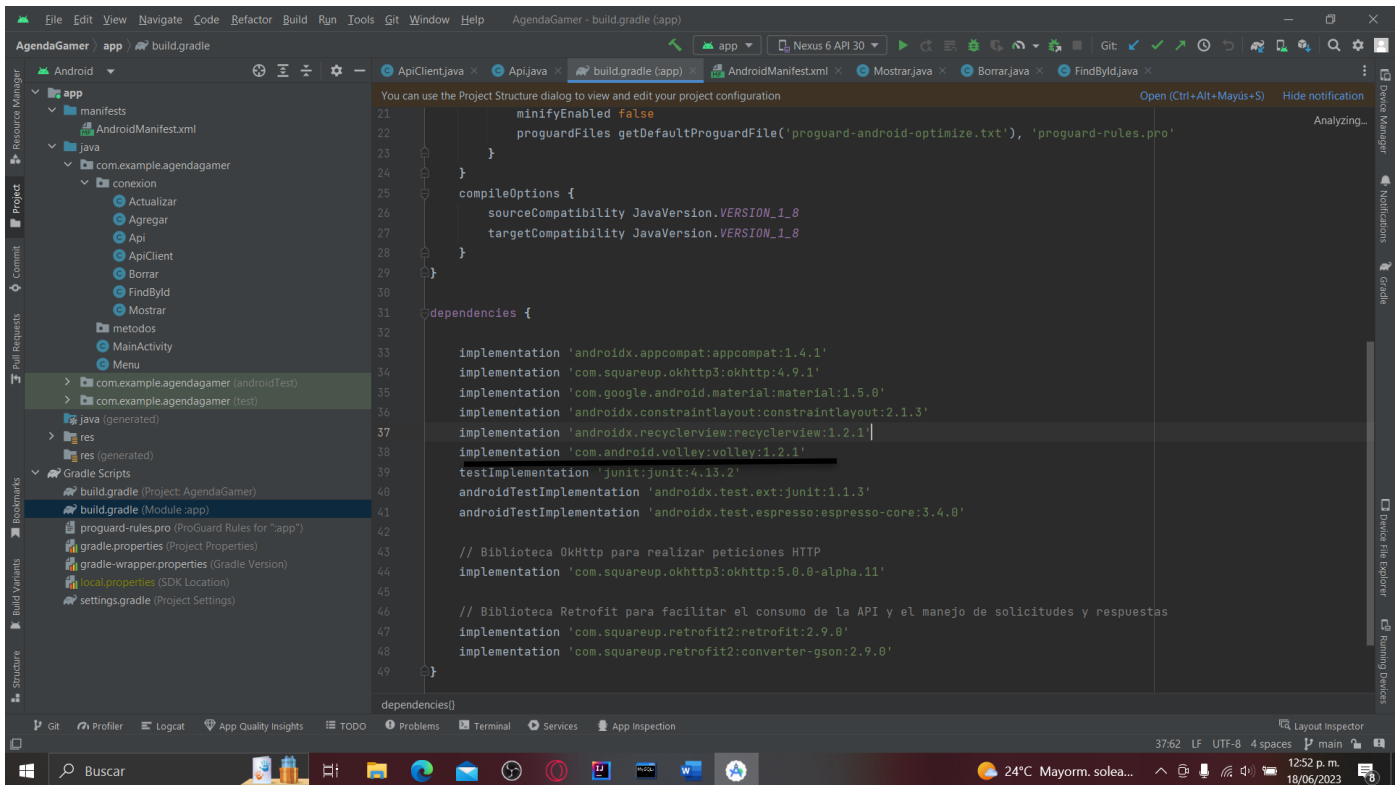
5. En xampp activar también apache.
6. En navegador [localhost/phpmyadmin](http://localhost/phpmyadmin).
7. Crear nueva bd agenda\_isic3401.



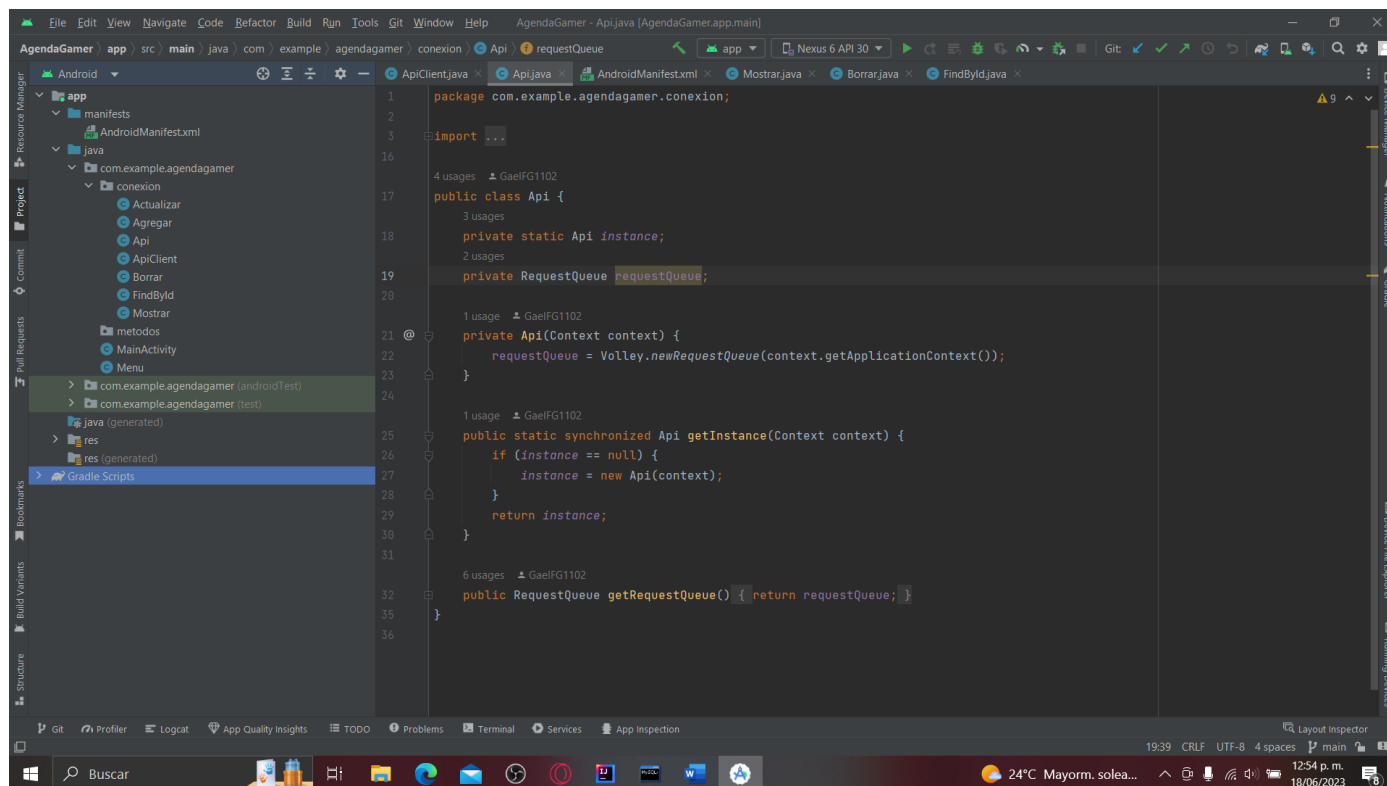
Ahora vamos a la app de Android, agregamos dos permisos fundamentales, el de internet y el de trafico limpio.



Ahora en build gradle agregamos la dependencia de volley



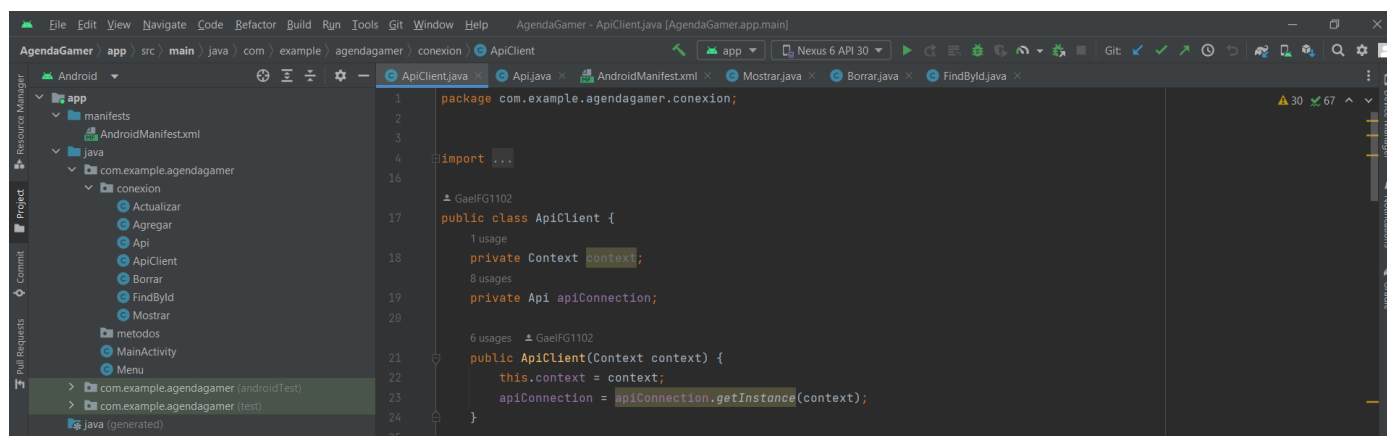
Ahora creamos una clase llamada Api que se encargara de la conexión:



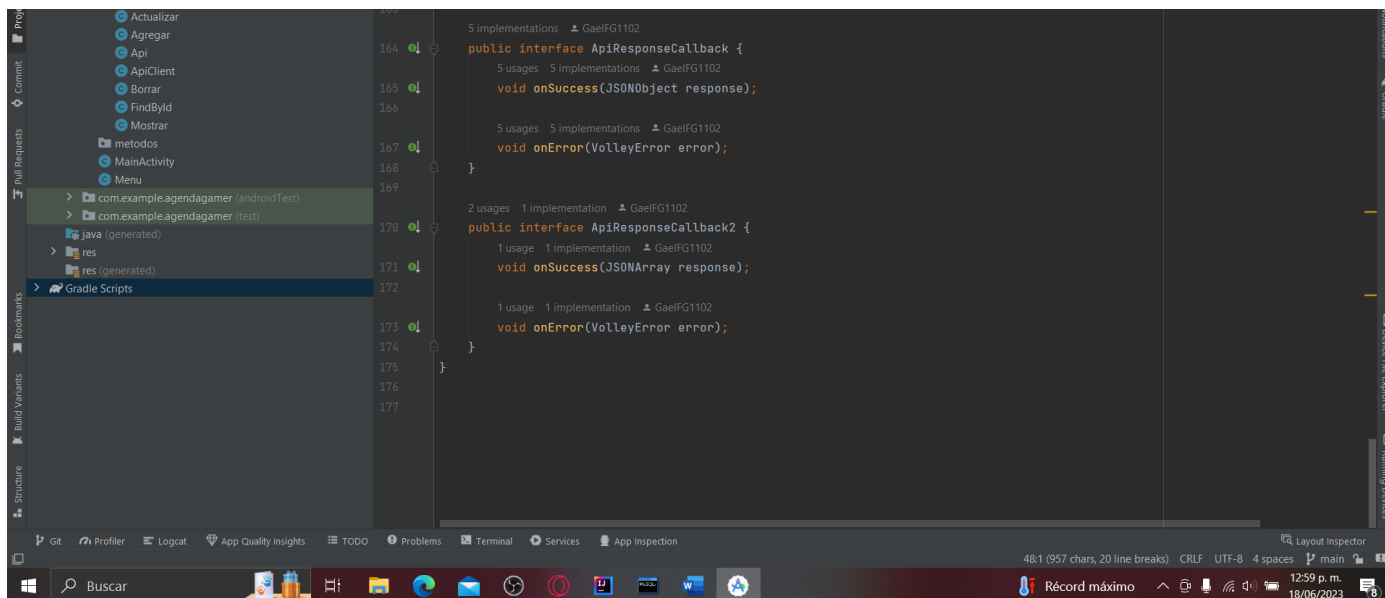
```
1 package com.example.agendagamer.conexion;
2
3 import ...
4
5 4 usages  ▲ GaelFG1102
6 public class Api {
7     3 usages
8     private static Api instance;
9     2 usages
10    private RequestQueue requestQueue;
11
12    1 usage  ▲ GaelFG1102
13    private Api(Context context) {
14        requestQueue = Volley.newRequestQueue(context.getApplicationContext());
15    }
16
17    1 usage  ▲ GaelFG1102
18    public static synchronized Api getInstance(Context context) {
19        if (instance == null) {
20            instance = new Api(context);
21        }
22        return instance;
23    }
24
25    6 usages  ▲ GaelFG1102
26    public RequestQueue getRequestQueue() { return requestQueue; }
27
28 }
29
30 }
```

Esta debe quedar exactamente igual.

Despues una llamada api client que se encargara de las solicitudes, en ella iran las solicitudes para nuestra api y dependiendo la respuesta de la api será la estructura del método tomemos como ejemplo la de mostrar todos.



```
1 package com.example.agendagamer.conexion;
2
3 import ...
4
5 1 usage  ▲ GaelFG1102
6 public class ApiClient {
7     8 usages
8     private Context context;
9     private Api apiConnection;
10
11    6 usages  ▲ GaelFG1102
12    public ApiClient(Context context) {
13        this.context = context;
14        apiConnection = apiConnection.getInstance(context);
15    }
16
17 }
```



Esa es la estructura de la clase, ahora si veamos el ejemplo

```
public void mostrar(final ApiResponseCallback2 callback) {
    String url = "http://172.20.10.2:8080/bdgamer/mostrar-todos";
    JsonRequest request = new JsonRequest(Request.Method.GET, url, null,
        new Response.Listener<JSONArray>() {
            @Override
            public void onResponse(JSONArray response) {
                // Procesa la respuesta JSON y llama al callback
                callback.onSuccess(response);
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                // Llama al callback con el error
                callback.onError(error);
            }
        }
    );

    // Agrega la solicitud a la cola de solicitudes de Volley
    apiConnection.getRequestQueue().add(request);
}
```

En url cambian su url, la dirección ip la sacan de su maquina en terminal con el comando ipconfig

NOTA: La ip tienen que cambiarla cada que cambien de red

Ese es el método para una solicitud de get que devuelve un array de objetos json, ahora veamos el manejo de la respuesta.

Creamos un objeto de Apicliente:

```
private ApiClient apiClient;
```



Le pasamos el contexto de nuestra clase

```
apiClient = new ApiClient(Mostrar.this);
```

Y mandamos a llamar el método, mandándole los parámetros que hayamos declarado en el método, lo que sigue es un manejo básico de objetos JSON, como puede ver solo es cambiar parámetros por los de su api

```
apiClient.mostrar(new ApiClient.ApiResponseCallback2() {  
    1 usage  👤 GaelFG1102  
    @Override  
    public void onSuccess(JSONArray response) {  
        try {  
            StringBuilder data = new StringBuilder();  
  
            if (response.length() == 0) {  
                // No hay registros disponibles  
                data.append("No hay registros disponibles");  
            } else {  
                for (int i = 0; i < response.length(); i++) {  
                    JSONObject jsonObject = response.getJSONObject(i);  
  
                    // Verificar si los campos en el objeto JSON están vacíos  
                    if (!jsonObject.isNull( name: "id") && !jsonObject.isNull( name: "nombre")  
                        && !jsonObject.isNull( name: "horasJugadas") && !jsonObject.isNull( name: "logros")) {  
  
                        // Obtener los valores de los campos en cada objeto JSON  
                        Long id = jsonObject.getLong( name: "id");  
                        String nombre = jsonObject.getString( name: "nombre");  
                        int horasJugadas = jsonObject.getInt( name: "horasJugadas");  
                        int logros = jsonObject.getInt( name: "logros");  
  
                        // Construir la cadena de texto para cada registro  
                        String recordData = "ID: " + id + "\nNombre: " + nombre +  
                            "\nHoras jugadas: " + horasJugadas + "\nLogros: " + logros;  
  
                        // Agregar el registro a la cadena de texto general  
                        data.append(recordData).append("\n\n");  
                    }  
                }  
            }  
        }  
    }  
});
```



```
    }  
    }  
}  
  
// Verificar si no se encontraron registros válidos  
if (data.length() == 0) {  
    data.append("No se encontraron registros válidos");  
}  
  
// Asigna el texto al TextView  
txtres.setText(data.toString());  
} catch (JSONException e) {  
    e.printStackTrace();  
    txtres.setText("Error al procesar la respuesta del servidor");  
}  
}
```

## Borrar: IntelliJ Implementacion

```
@Override  
public boolean borrarJuegoPorId(long id) {  
    Optional<AgendaGamer> juegoOptional = bdGamerRepo.findById(id);  
    if (juegoOptional.isPresent()) {  
        bdGamerRepo.delete(juegoOptional.get());  
        return true;  
    }  
    return false;  
}
```



## En la clase Api

```
@DeleteMapping("/borrar/{id}")
public ResponseEntity<String> borrarJuego(@PathVariable long id) {
    boolean borrado = bdGamerServ.borrarJuegoPorId(id);
    if (borrado) {
        return ResponseEntity.ok( body: "Juego eliminado correctamente.");
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("No se encontró el juego con el ID especificado.");
    }
}
```

## Ahora su implementación en Android

### En apiclient

```
public void delete(final int id, final ApiResponseCallback callback) {
    String url = "http://10.0.8.57:8080/bdgamer/borrar/" + id;
    StringRequest request = new StringRequest(Request.Method.DELETE, url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                // Llama al callback con una respuesta vacía
                callback.onSuccess(null);
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                // Llama al callback con el error
                callback.onError(error);
            }
        }
    ));

    // Agrega la solicitud a la cola de solicitudes de Volley
    apiConnection.getRequestQueue().add(request);
}
```

Manejo de respuesta, ya sea un un botón o algo :



```
btbor.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        apiClient = new ApiClient(context: Borrarr.this);
        String idStr = b.getText().toString();

        // Verificar si el campo está vacío o no es numérico
        if (idStr.isEmpty()) {
            Toast.makeText(context: Borrarr.this, text: "Ingrese un ID válido", Toast.LENGTH_SHORT).show();
            return;
        }

        int id = Integer.parseInt(idStr);

        apiClient.delete(id, new ApiClient.ApiResponseCallback() {
            @Override
            public void onSuccess(JSONObject response) {
                Toast.makeText(context: Borrarr.this, text: "Borrado Correctamente", Toast.LENGTH_LONG).show();
            }
            @Override
            public void onError(VolleyError error) {
            }
        });
    }
});
```

Asumiendo que como con mostrar ya crearon su objeto apiclient

```
private ApiClient apiClient;
```

Ahora para actualizar, aquí yo actualice un dato concreto ustedes pueden hacer con todos: IntelliJ

En la interfaz servicio

```
no usages 1 implementation
boolean actualizarLogros(long id, AgendaGamer juegoActualizado);
```

En implementación:



```
@Override
public boolean actualizarHoras(long id, AgendaGamer juegoActualizado) {
    Optional<AgendaGamer> juegoOptional = bdGamerRepo.findById(id);
    if (juegoOptional.isPresent()) {
        AgendaGamer juegoExistente = juegoOptional.get();

        // Actualiza los campos necesarios del juego existente con los valores del juego actualizado
        juegoExistente.setHorasJugadas(juegoActualizado.getHorasJugadas());
        // ...

        // Guarda los cambios en la base de datos
        bdGamerRepo.save(juegoExistente);

        return true;
    }
    return false;
}
```

**Nota:** Si quieren actualizar todo de putaso, solo copia y pega la siguiente línea

```
juegoExistente.setHorasJugadas(juegoActualizado.getHorasJugadas());
```

**Y cambia el get y set por el método get y set de tu campo a actualizar**

**Ejemplo**

```
juegoExistente.setNombre(juegoActualizado.getNombre());
```

**En api**

```
no usages
@PostMapping("/horas/{id}")
public ResponseEntity<AgendaGamer> actualizarHoras(@PathVariable long id, @RequestBody AgendaGamer juegoActualizado) {
    Optional<AgendaGamer> juegoOptional = bdGamerServ.mostrarJuegoPorId(id);
    if (juegoOptional.isPresent()) {
        AgendaGamer juegoExistente = juegoOptional.get();
        // Actualiza los campos necesarios del juego existente con los valores del juego actualizado
        juegoExistente.setHorasJugadas(juegoActualizado.getHorasJugadas());
        // ...

        // Guarda los cambios en la base de datos
        AgendaGamer juegoActualizadoGuardado = bdGamerServ.guardarJuego(juegoExistente);
        return ResponseEntity.ok(juegoActualizadoGuardado);
    } else {
        // Maneja el caso en el que el juego no existe
        return ResponseEntity.notFound().build();
    }
}
```

**Nota:** Si quieren actualizar todo de putaso, solo copia y pega la siguiente línea

```
juegoExistente.setHorasJugadas(juegoActualizado.getHorasJugadas());
```

**Y cambia el get y set por el método get y set de tu campo a actualizar**

**Ejemplo**

```
juegoExistente.setNombre(juegoActualizado.getNombre());
```

## En Android

### Apiclient:

```
public void actualizarJuego2(final int id,final String dato, final JSONObject
valoresActualizados, final ApiResponseCallback callback) {
    String url = "http://172.20.10.2:8080/bdgamer/logros/" + id;

    JSONObjectRequest request = new JSONObjectRequest(Request.Method.POST, url,
valoresActualizados,
        new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
                // La actualización se ha realizado correctamente
                // Puedes realizar las acciones necesarias después de la
actualización
                callback.onSuccess(response);
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                // Manejar el error de la solicitud
                callback.onError(error);
            }
        }
    ));

    // Agregar la solicitud a la cola de solicitudes de Volley
    apiConnection.getRequestQueue().add(request);
}
```

### Implementacion y manejo:

```
try {
    JSONObject valoresActualizados = new JSONObject();
    valoresActualizados.put("logros", dato);

    // Luego, llamas al método actualizarJuego pasando el ID y el objeto
valoresActualizados
    apiClient.actualizarJuego2(id, "logros", valoresActualizados, new
ApiClient.ApiResponseCallback() {
        @Override
        public void onSuccess(JSONObject response) {
            Toast.makeText(Actualizar.this, "Actualizado Correctamente",
Toast.LENGTH_LONG).show();
        }

        @Override
        public void onError(VolleyError error) {
            Toast.makeText(Actualizar.this, "No se pudo actualizar Correctamente",
Toast.LENGTH_LONG).show();
        }
    });
} catch (JSONException e) {
    e.printStackTrace();
}
```

El campo dat del valor lo puedes sacar de un editext, recuerda si quieres actualizar varios valores a la ves, en el JSONObject tienes que mandar mas valores ejemplo

```
valoresActualizados.put( name: "logros", dat);  
valoresActualizados.put( name: "Nombre", nombre);  
valoresActualizados.put( name: "fecha", fecha);
```

Ultimo ejemplo

FindId

IntelliJ

Implementacion

```
@Override  
public Optional<AgendaGamer> mostrarJuegoPorId(long id) {  
    return bdGamerRepo.findById(id);  
}
```

Api:

```
@GetMapping("/finbyid/{id}")  
public Optional<AgendaGamer> mostrarJuegoPorId(@PathVariable long id) {  
    Optional<AgendaGamer> juegoOptional = bdGamerServ.mostrarJuegoPorId(id);  
    if (juegoOptional.isPresent()) {  
        return bdGamerServ.mostrarJuegoPorId(id);  
    } else {  
        return Optional.empty();  
    }  
}
```

Android:

ApiClient:

```
public void findid(final int id, final ApiResponseCallback callback) {  
    String url = "http://172.20.10.2:8080/bdgamer/finbyid/" + id;  
    JSONObjectRequest request = new JSONObjectRequest(Request.Method.GET, url, null,  
        new Response.Listener<JSONObject>() {  
            @Override  
            public void onResponse(JSONObject response) {  
                // Procesa la respuesta JSON y llamar al callback  
                callback.onSuccess(response);  
            }  
        },  
        new Response.ErrorListener() {  
            @Override  
            public void onErrorResponse(VolleyError error) {  
                // Llama al callback con el error  
                callback.onError(error);  
            }  
        }  
    );  
  
    // Agrega la solicitud a la cola de solicitudes de Volley  
    apiConnection.getRequestQueue().add(request);  
}
```





## Manejo de respuesta:

```
btnfind.setOnClickListener(new View.OnClickListener() {  
    @GaelFG1102  
    @Override  
    public void onClick(View v) {  
        apiClient = new ApiClient( context: FindById.this);  
        String idStr = i.getText().toString();  
  
        if (idStr.isEmpty()) {  
            Toast.makeText( context: FindById.this, text: "Ingrese un ID válido", Toast.LENGTH_SHORT).show();  
            return;  
        }  
  
        int id1 = Integer.parseInt(idStr);  
  
        @GaelFG1102  
        apiClient.findid(id1, new ApiClient.ApiResponseCallback() {  
            5 usages @GaelFG1102  
            @Override  
            public void onSuccess(JSONObject response) {  
                try {  
                    JSONObject jsonResponse = new JSONObject(response.toString());  
                    int iden = jsonResponse.getInt( name: "id");  
                    String nombre = jsonResponse.getString( name: "nombre");  
                    int horasJugadas = jsonResponse.getInt( name: "horasJugadas");  
                    int logros = jsonResponse.getInt( name: "logros");  
  
                    String data = "ID : " + iden + "\nNombre: " + nombre +  
                        "\nHoras: " + horasJugadas + "\nLogros: " + logros;  
                    txtres.setText(data);  
                } catch (JSONException e) {  
                }  
            }  
        }  
    }  
});  
5 usages @GaelFG1102  
@Override  
public void onError(VolleyError error) {  
}  
});  
});
```

Eso seria todo.

## V. Conclusiones: