



PROYECTO INTERMODULAR

PSP, Acceso a Datos, Inglés y PMDM

DESCRIPCIÓN BREVE

Desarrollar juego Android en el que es necesario el intercambio de información entre diferentes dispositivos móviles, como por ejemplo para compartir un ranking de jugadores con puntos y ver qué jugadores se conectan y desconectan del juego “en vivo”

Josué, Ronny y Dani
2º DAM

Índice:

| | |
|---------------------------------------------------------------------|----|
| Sesión 1 | 2 |
| Activar el equipo | 2 |
| Roles | 2 |
| Sesión 2 | 2 |
| Identificar reto | 2 |
| Producto | 2 |
| Sesión 3 | 2 |
| Alternativas | 2 |
| Propuesta final | 2 |
| Sesión 4 | 3 |
| Tablero Kanban | 3 |
| Sesiones 5 y 6 | 3 |
| Diagrama de clases | 3 |
| Diagrama entidad - relación | 4 |
| Gráfico ruta Memory | 5 |
| Sesiones 7-16 | 6 |
| Traducción del interfaz en inglés | 6 |
| Documentación | 6 |
| Programación y pruebas de las aplicaciones cliente y servidor | 6 |
| Android | 6 |
| Mongodb (Atlas) + Heroku | 8 |
| Sesiones 17-20 | 11 |
| Sesión 21 | 11 |
| Coevaluación y autoevaluación | 11 |
| Sesión 22 | 11 |
| Feedback final | 11 |
| Bibliografía | 12 |

Sesión 1

Activar el equipo

Los integrantes del equipo somos Josué, Ronny y Dani.

Roles

Moderador: Dani.

Secretario: Josué.

Controlador del tiempo: Ronny.

Sesión 2

Identificar reto

Vamos a generar un proyecto en el cual adaptaremos uno de nuestros proyectos del módulo Programación Multimedia y Dispositivos Móviles (Memory) para que realice el intercambio de información entre diferentes dispositivos móviles, en concreto un ranking de jugadores con sus puntuaciones.

Producto

Para realizar este reto, necesitaremos que nuestra aplicación móvil (Memory) se comuniquen con otra aplicación desarrollada en JavaScript que actuará como servidor. Dicho servidor se encargará de guardar en una base de datos el ranking global del juego para que pueda ser consultado por los clientes.

Sesión 3

Alternativas

Para la realización de este proyecto vamos a hacer uso de Android Studio para el desarrollo (ya realizado) de memory y poder conectarlo a nuestro servidor. Este servidor lo realizaremos con tecnología node.js y será desarrollado en JavaScript. Por último, para la base de datos tenemos varias opciones: Firebase, SQL Server, MongoDB. Las opciones que más se ajustan a nuestro objetivo son SQL Server y MongoDB.

Propuesta final

Finalmente hemos elegido el proyecto Memory de Josué.

Nos hemos decantado por usar MongoDB como gestor de base de datos.

Sesión 4

Tablero Kanban

Para la gestión del tiempo y organización de las tareas del proyecto utilizamos un tablero Kanban en Trello.

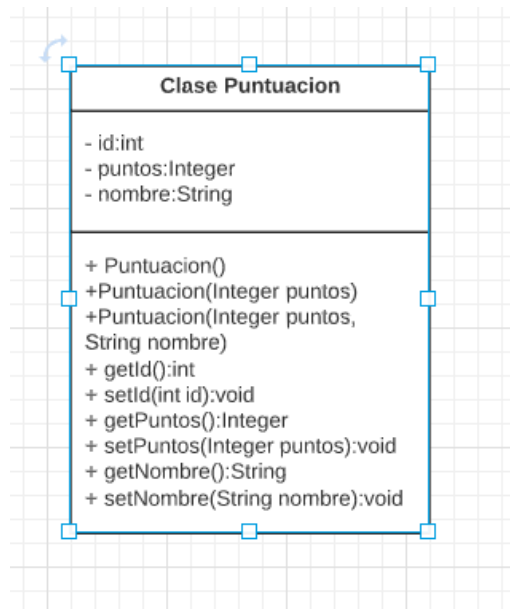
Link del tablero:

<https://trello.com/invite/b/6laCmBwd/3347c6f2129f8203b4a00f6ca8441d55/proyecto-intermodular>

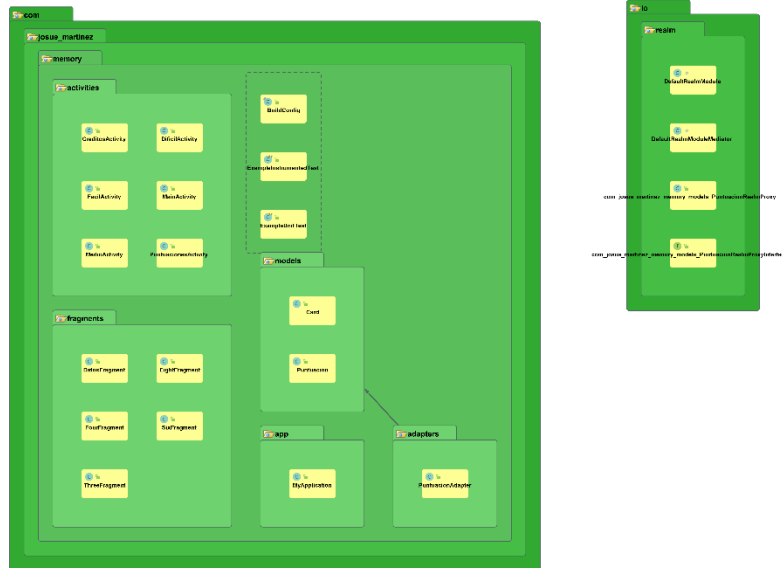
Sesiones 5 y 6

Diagrama de clases

Dentro del proyecto Memory de Josué se distingue claramente el modelo “Puntuación” que nos será útil para el almacenamiento en la base de datos:



Mediante la herramienta de “IrisCode” nos crea varios diagramas de clases del proyecto entero, primero desde la vista de los paquetes:



Y luego desde la vista de todas las clases:

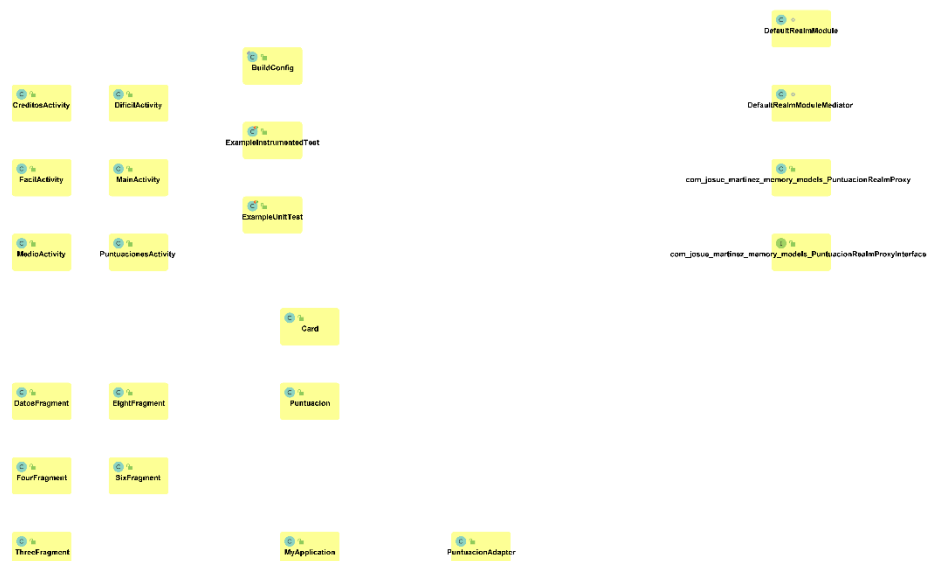


Diagrama entidad - relación

Al tener una base de datos tan sencilla, únicamente tendremos una tabla “Puntuaciones” en la que almacenaremos el idpuntuacion (Int y Primary Key), el nombre (String) y los puntos (Int):

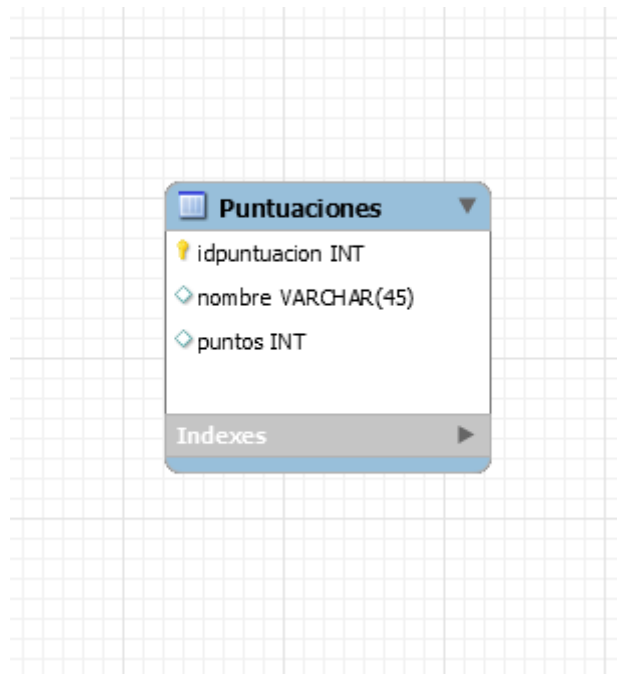
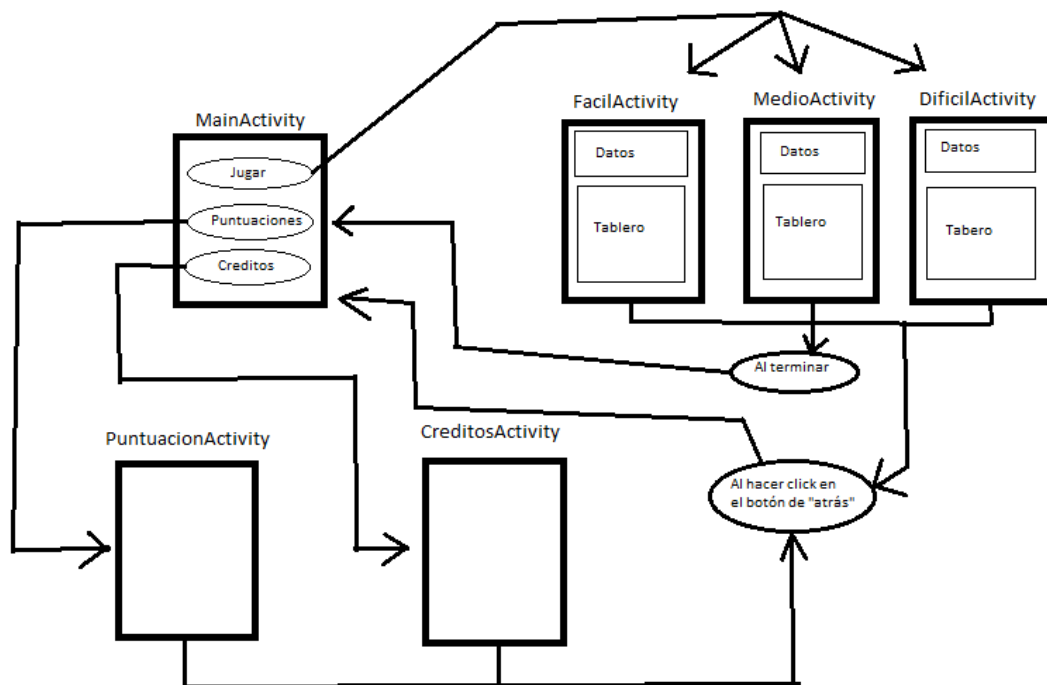


Gráfico ruta Memory

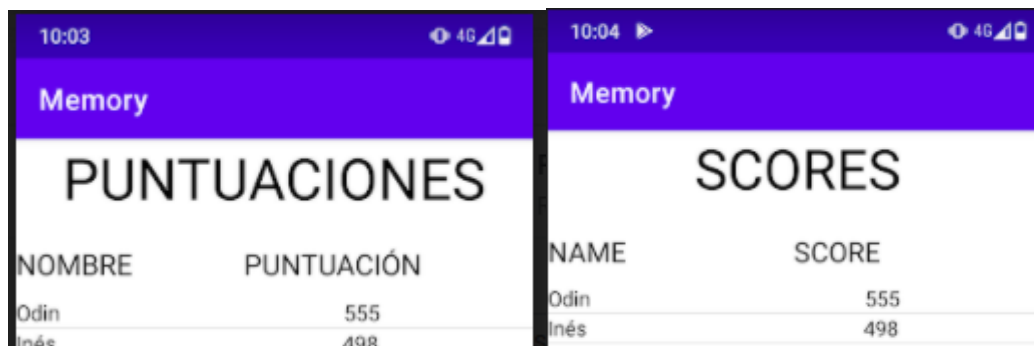
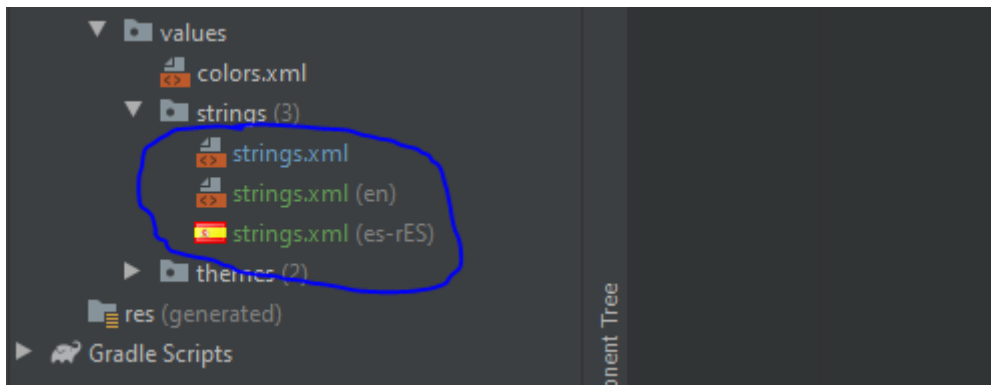
Para poder entender mejor el proyecto Memory, hemos diseñado un gráfico de navegación entre las Activities de la aplicación:



Sesiones 7-16

Traducción del interfaz en inglés

Parte del proyecto era que la aplicación estuviese completamente traducida al inglés. Y se mostraría en inglés o castellano dependiendo del idioma oficial del teléfono móvil. Esto lo hemos realizado mediante el multilinguaje de Android:



Documentación

Este es el archivo de documentación en donde se ve lo que hemos ido haciendo a lo largo de las sesiones.

Programación y pruebas de las aplicaciones cliente y servidor

Aquí es donde hemos programado todo lo necesario para tener nuestro proyecto.

Android

En Android hemos añadido en el proyecto de Memory varias cosas:

Lo primero es la traducción al inglés explicada anteriormente.

Después necesitábamos una biblioteca para realizar comunicaciones con un servidor rest API. Había varias opciones e investigamos [OkHttp](#) y Volley.

Al final nos decantamos por la opción de Volley que es algo más completa y más usada en el entorno.

Para ello es necesario añadir la librería en el graddle de esta manera:

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'com.android.volley:volley:1.1.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
}
```

Lo sincronizamos y por supuesto, no tendremos conexión a internet a no ser que le demos el permiso en el Manifest:

```

1 package="com.josue_dam.consejmemory"
2
3
4
5 <!-- PERMISO PARA USAR INTERNET -->
6 <uses-permission android:name="android.permission.INTERNET"/>
7
8 <application
9     android:name=".app.MyApplication"
10    android:allowBackup="true"
11    android:icon="@mipmap/ic_launcher"

```

Entrando más en el código vamos a distinguir claramente dos acciones, el GET y POST. Desde Volley podemos hacer peticiones StringRequest (únicamente para objetos String) o hacer uso de JSONArray y JSONObject son los más útiles en este tipo de situación. Nosotros hemos usado JSONObject tanto para el GET como para el POST.

La URL a trabajar será la generada por Heroku:

<https://project-memorygame.herokuapp.com/puntuaciones>

```

public class MainActivity extends AppCompatActivity {
    //API
    private static final String URL1 = "https://project-memorygame.herokuapp.com/puntuaciones";
}

```

La petición GET para obtener los datos del servidor API lo tendremos en ActivityPuntuaciones que es donde queremos visualizar los datos:

```

private void jsonArrayRequest(){
    JSONArrayRequest jsonArrayRequest = new JSONArrayRequest(
        Request.Method.GET,
        URL1,
        jsonRequest: null,
        new Response.Listener<JSONArray>() {
            @RequiresApi(api = Build.VERSION_CODES.N)
            @Override
            public void onResponse(JSONArray response) {
                int size = response.length();
                for (int i = 0; i < size; i++){
                    try {
                        JSONObject jsonObject = new JSONObject(response.get(i).toString());
                        String nombre = jsonObject.getString( name: "nombre");
                        Integer puntos = jsonObject.getInt( name: "puntos");
                        //Creo la "puntuacion" temporal
                        Puntuacion puntuacionTmp = new Puntuacion(puntos, nombre);
                        //y la añado a la lista
                        listaPuntuacionesApi.add(puntuacionTmp);
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
                //Por ultimo la añado al recycler
                final PuntuacionAdapter puntuacionAdapter = generarRecycler(listaPuntuacionesApi);
            }
        }
    );
}

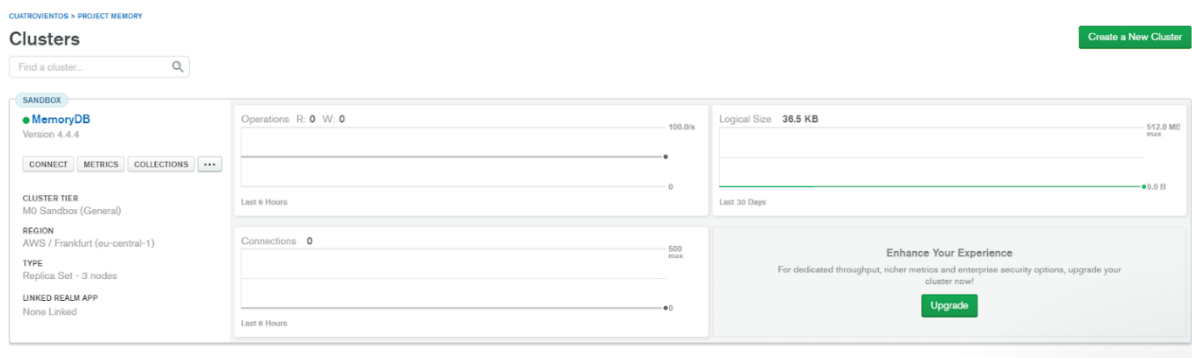
```


La petición POST para enviar datos al servidor API con las “keys” definidas en la base de datos lo tendremos en FacilActivity, MedioActivity y DificilActivity que es donde al finalizar el juego mandará esos datos al servidor:

```
private void postRequest(String nombre, Integer puntos) {  
    RequestQueue requestQueue = Volley.newRequestQueue( context, this);  
    JSONObject jsonObject = new JSONObject();  
    try {  
        //input your API parameters: Aquí puntos y nombre  
        jsonObject.put( name: "puntos", puntos);  
        jsonObject.put( name: "nombre", nombre);  
    } catch (JSONException e) {  
        e.printStackTrace();  
    }  
  
    JSONObjectRequest jsonObjectRequest = new JSONObjectRequest(  
        URL1,  
        jsonObject,  
        new Response.Listener<JSONObject>() {  
            @Override  
            public void onResponse(JSONObject response) {  
                Toast.makeText(getApplicationContext(), response.toString(), Toast.LENGTH_SHORT).show();  
            }  
        },  
        new Response.ErrorListener() {  
            @Override  
            public void onErrorResponse(VolleyError error) {  
                Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();  
            }  
        }  
    );  
    requestQueue.add(jsonObjectRequest);  
}
```

Mongodb (Atlas) + Heroku

Lo primero es crear nuestro clúster en MongoDB Atlas:



Aquí tenemos la colección de datos qué recopilamos del juego con su nombre y su puntuación correspondiente:

DATABASES: 1 COLLECTIONS: 1

+ Create Database

Q NAMESPACES

puntuacionesdb

puntuaciones

puntuacionesdb.puntuaciones

COLLECTION SIZE: 476B TOTAL DOCUMENTS: 8 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

FILTER {"filter": "example"}

QUERY RESULTS 1-8 OF 8

```
{
  "_id": ObjectId("60363e29253e37fd6c7cdf69"),
  "nombre": "Daniel",
  "puntos": 12
}
```

```
{
  "_id": ObjectId("603754e5c81e2d1534a4b56a"),
  "nombre": "Josue",
  "puntos": 25
}
```

```
{
  "_id": ObjectId("6038c774e800a900151ebf68"),
  "nombre": "Ronny",
  "puntos": 277,
  "__v": 0
}
```

Teníamos que conectar la Base de datos con nuestro servidor y para eso el MongoDB Atlas nos proporcionaba la conexión con el código en la función .Connect añadiendo el usuario, la contraseña y la base de datos.

Este es el esquema de puntuaciones que utilizamos que es el nombre de tipo String y los puntos que los ponemos de tipo Number:

```
TS memory-rest-server.ts > ...
1  const mongoose = require('mongoose');
2  const express = require('express');
3  const bodyParser = require('body-parser');
4
5  mongoose.connect('mongodb+srv://projectmemorygame:4Vientos@memorydb.lwrzy.mongodb.net/puntuacionesdb?retryWrites=true&w=majority', {
6    useNewUrlParser: true,
7    useUnifiedTopology: true
8  });
9
10 const app = express();
11
12 let puntuacionesSchema = new mongoose.Schema({
13   nombre: {
14     type: String,
15     required: true,
16     minlength: 1,
17   },
18   puntos: {
19     type: Number,
20     minlength: 1,
21   },
22 });
23
24 let Puntuacion = mongoose.model('puntuaciones', puntuacionesSchema);
25
26 app.use(function (req: any, res: any, next: any) {
27   res.header("Access-Control-Allow-Origin", "*"); // update to match the domain you will make the request from
28   res.header("Access-Control-Allow-Methods", 'GET,PUT,POST,DELETE,OPTIONS');
29   res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
30   next();
31 });
32
33 app.use(
34   bodyParser.urlencoded({
35     extended: true
36   })
37 )
38
39 app.use(bodyParser.json())
40
```

Le añadimos el (GET) para obtener todos los datos y el (PUT) para insertarlos, tenemos también el código de actualizar y eliminar, pero no lo utilizaremos en ningún momento del programa:

```
app.get('/puntuaciones', async (req:any, res:any) => {
  const puntuaciones = await Puntuacion.find({});
  console.log(puntuaciones);

  try {
    res.send(puntuaciones);
  } catch (err) {
    res.status(500).send(err);
  }
});

app.post('/puntuaciones', async (req:any, res:any) => {
  const puntuacion = new Puntuacion({
    nombre: req.body.nombre,
    puntos: req.body.puntos,
  });

  try {
    await puntuacion.save();
    res.send(puntuacion);
  } catch (err) {
    res.status(500).send(err);
  }
});

app.delete('/puntuaciones/:id', async (req:any, res:any) => {
  try {
    const puntuacion = await Puntuacion.findByIdAndDelete(req.params.id)

    if (!puntuacion) res.status(404).send("No item found")
    res.status(200).send()
  } catch (err) {
    res.status(500).send(err)
  }
});

app.put('/puntuacion/:id', async (req:any, res:any)=> {
  try {
    const puntuacion = await Puntuacion.findByIdAndUpdate(req.params.id, req.body)
    await Puntuacion.save()
    res.send(puntuacion)
  } catch (err) {
    res.status(500).send(err)
  }
});
```

Ahora continuamos con la parte del servidor.

Por último, tendremos el código para arrancar el servidor, la primera parte (comentada) fue para probar si funcionaba en la base de datos de MongoDB Atlas, el segundo servirá para iniciar el servidor con Heroku y MongoDB Atlas a la vez:

```
// This is not working with Heroku, IP and PORT are automatically assigned
//const server = app.listen(8000, "localhost", () => {
//  const { address, port } = server.address();

//  console.log('Listening on %s %s', address, port);
// });

// start the server listening for requests
app.listen(process.env.PORT || 3000,
  () => console.log("Server is running..."));
```

Pasamos a la parte del Heroku que sirve para Express, antes de nada, tendremos que instalar el CLI de Heroku. Una vez hecho esto seguiremos los pasos que nos indica la propia página para subir el proyecto a un repositorio logueándonos antes:

Install the Heroku CLI

Download and install the [Heroku CLI](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Clone the repository

Use Git to clone project-memorygame's source code to your local machine.

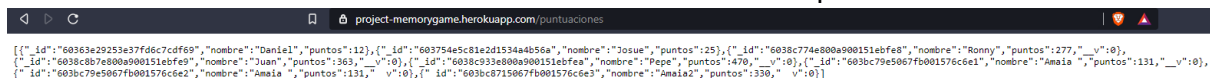
```
$ heroku git:clone -a project-memorygame
$ cd project-memorygame
```

Deploy your changes

Make some changes to the code you just cloned and deploy them to Heroku using Git.

```
$ git add .
$ git commit -am "make it better"
$ git push heroku master
```

Haremos un push y recibiremos la página para poder acceder a nuestros datos que teníamos en la Base de datos añadiendo en nuestro caso en la ruta el “/puntuaciones”.



```
[{"_id":"60363e29253e37fd6c7cdf69","nombre":"Daniel","puntos":12}, {"_id":"603754e5c81e2d1534a4b56a","nombre":"Josue","puntos":25}, {"_id":"6038c774e800a900151ebfe8","nombre":"Ronny","puntos":277,"__v":0}, {"_id":"6038c8b7e800a900151ebfe9","nombre":"Juan","puntos":363,"__v":0}, {"_id":"6038c933e800a900151ebfe9","nombre":"Pepe","puntos":470,"__v":0}, {"_id":"6038c79e5067fb001576c6e1","nombre":"Anaia ","puntos":131,"__v":0}, {"_id":"6038c79e5067fb001576c6e2","nombre":"Anaia ","puntos":131,"__v":0}, {"_id":"6038c715067fb001576c6e3","nombre":"Anaia2","puntos":1330,"__v":0}]
```

Sesiones 17-20

Realización del power point y presentación final del proyecto al resto de equipos y profesorado.

Sesión 21

Coevaluación y autoevaluación

Sesión 22

Feedback final

Este Proyecto nos ha servido para ver como se unen varios módulos y como funciona realmente el trabajo en el mundo laboral.

Hemos ido estudiando y aprendiendo a lo largo del curso las cosas por separado en cada asignatura y ahora al final podemos ver como realmente se junta todo; mezclándose la programación pura de una aplicación (en este caso el juego de móvil Memory que actúa como

cliente), las bases de datos (en nuestro caso hemos usado una sola tabla "Puntuaciones") subido en MongoDB Atlas y por último un servidor, que es el que gestionará esa base de datos, que posteriormente lo subiremos a internet → Mediante Heroku.

Bibliografía

API Rest MongoDB en Android:

https://www.youtube.com/watch?v=E6XicTeBgrs&ab_channel=CodigoInteractivo

API Rest con node:

<https://carlosazaustre.es/como-crear-una-api-rest-usando-node-js>

Creación servidor en la nube:

<https://www.youtube.com/watch?v=OuCrHynro0w>

Solicitudes a API en Android:

<https://developer.android.com/training/volley/request?hl=es-419>

Post request:

<https://stackoverflow.com/questions/29442977/volley-jsonobjectrequest-post-parameters-no-longer-work>

Volley:

https://www.youtube.com/watch?v=mJYwYc51IKI&ab_channel=KikoPalomares → Kotlin

https://www.youtube.com/watch?v=4PljHEv6Xwc&ab_channel=Lacuevadelprogramador