Paradigmas Avanzados de Programación

Azure Candy Crush

Grado en Ingeniería Informática
Universidad de Alcalá

Alexander Brun Guajardo
Pablo García García

12 de mayo de 2023

Índice general

Capítulo 1

Sistema de puntuación

1.1. Introducción

En esta primera parte de la práctica, se solicita llevar a cabo una modificación sobre lo desarrollado en la segunda práctica, implementando un sistema de puntuación. Dicho sistema va a funcionar de la siguiente manera:

1. Por cada bloque borrado, bien sea normal o especial, se sumará un punto
2. Por cada diez caramelos borrados, se sumará un punto adicional
3. Con respecto a los bloques especiales, además de sumar el punto mencionado anteriormente, se sumará también una cantidad de puntos en base a la siguiente ponderación:
   - **Bloque bomba**: en caso de eliminar este bloque especial, se agregarán cinco puntos extras al contador.
   - **Bloque TNT**: en caso de seleccionar este tipo de bloque, la cantidad de puntos extras será de diez.
   - **Bloque rompecabezas**: cuando este tipo de bloque sea seleccionado, puesto que este es el que mayor complejidad tiene a la hora de poder generarlo, la cantidad de puntos extra otorgados al jugador es mayor, siendo esta de quince.
4. Se va a contar también con un multiplicador de puntos, el cual dejará el valor de los puntos en su estado original cuando se esté jugando en la modalidad *Fácil* y los duplicará cuando la modalidad de juego sea *Difícil*.

Esta cantidad de puntos, como veremos más adelante, se irá actualizando en cada jugada hasta que el jugador se quede sin vidas, momento en el que se solicitará su nombre para así mandarlo junto a los puntos totales que ha realizado, la fecha de la partida y su duración, al Cloud. Podemos ver un ejemplo de como se van actualizando los puntos de una jugada a otra en la Figura 1.1.

```
Puntos: 24
Introduzca fila: 8
Introduzca columna: 1

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
  -----------------------------------------
0 | 5 | X | 6 | 3 | 6 | 2 | 2 | 3 | 4 | 5 |
1 | 2 | X | 2 | 3 | 2 | 2 | 4 | 6 | 4 | 4 |
2 | 6 | X | 1 | 5 | 1 | 1 | 5 | 5 | 6 | 1 |
3 | 4 | X | 4 | 1 | 1 | 4 | 1 | 5 | 2 | 4 |
4 | 2 | X | 3 | 3 | 3 | 2 | 4 | 5 | 4 | 4 |
5 | 2 | X | 6 | 2 | 1 | 1 | 3 | 2 | 3 | 5 |
6 | 4 | X | 3 | 5 | 2 | 2 | 3 | 4 | 5 | 3 |
7 | 6 | X | 1 | 5 | 3 | 5 | 4 | 3 | 6 | 1 |
8 | 2 | X | 1 | 4 | 1 | 4 | 2 | 5 | 1 | 6 |
9 | 1 | X | 3 | 6 | 4 | 1 | 1 | 4 | 1 | 1 |

Puntos: 56
```

Figura 1.1: Ejemplo actualización de puntos

1.2. Métodos implementados

Para poder llevar a cabo todo lo mencionado anteriormente, se han seguido varios pasos, entre los que se encuentra la implementación de los siguientes métodos:

- `actualizarPuntos(puntos: Int, borrados: Int, caramelo: Int, modo: Int): Int`: como su propio nombre indica, este es el método que utilizaremos para actualizar los puntos en cada jugada, mediante las condiciones comentadas anteriormente.
- `pedirUsuarioNombre(): String`: se obtiene el nombre del jugador siendo este introducido por teclado.

Además de estos métodos, otros de los aspectos implementados ha sido la obtención de fechas de inicio y finalización, puesto que se obtienen en el momento en el que se inicia una nueva partida, y en el momento en el que se introduce el nombre al terminar.

Dichas fechas se obtienen haciendo uso del método `LocalDateTime.now()`. La fecha de inicio es tratada mediante `DateTimeFormatter.ofPattern("dd/MM/yyyy")`, de forma que se muestre en un formato de día/mes/año. Una vez que tenemos las dos fechas, calculamos la diferencia entre ambas gracias al método `ChronoUnit.SECONDS.between()`, para así obtener la duración de la partida expresada en segundos.

Por último, una vez que tenemos todos los datos necesarios, y una vez finalizada la partida, estos datos son enviados a la API para almacenarlos en la base de datos (la implementación de estos dos aspectos se explicarán detalladamente en el Capítulo 2), mediante una petición HTTP, en la que se pasarán como parámetros, usando el comando:

```
Seq(''cmd'', ''/c'', s''curl ''https://api-postgres.azurewebsites.net/api/api-postgres?
nombre=$nombre&puntos=$puntos&fecha=$fechaInicialFormateada&duracion=$duracionPartida''
                                    --ssl-no-revoke'').!
```

Finalmente, podemos observar un pequeño ejemplo de ejecución del envío de datos al finalizar una partida, en la Figura 1.2.

```
Puntos: 24
Fin del juego, te has quedado sin vidas!
Introduzca su nombre: Alejandro
Alejandro has obtenido un total de 24 puntos!
La duracion de la partida ha sido de: 43 y la partida se ha realizado en la siguiente fecha: 11/05/2023
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   127    0   127    0     0     35      0 --:--:--  0:00:03 --:--:--    35
Alejandro ha jugado durante 43 segundos, obteniendo una puntuacion de 24 a fecha de 11/05/2023
Datos insertados correctamente!
Process finished with exit code 0
```

Figura 1.2: Ejecución de ejemplo

Capítulo 2

Cloud Computing

En este capítulo extenderemos la funcionalidad de nuestro juego *Candy Crush* mediante los servicios que *Azure* nos ofrece en la nube, de forma que podamos llevar un conteo de las partidas, así como poder mostrarlo de forma pública.

2.1. Envío de datos mediante API en Python

La primera de las preguntas a realizarnos es, ¿cómo puedo mandar los datos de mi ordenador a la nube? La respuesta, en nuestro caso, ha sido mediante una API programada en Python, que reciba los datos, y los almacene en una base de datos. De esta manera, el envío de datos desde el juego será totalmente independiente de la base de datos, pues esto se realizará en la API.
Profundizando más, ha sido realizada mediante lo que en Azure es conocido como una *Function App*. Para ello, se ha descargado la extensión correspondiente de Visual Studio Code, y se ha partido de la plantilla de `HTTP trigger`. Dentro de los archivos, en primer lugar nos fijamos en `__init__.py`, que es el código que se ejecutará cuando llamemos a nuestra API. Exigimos que mediante una `request HTTP`, recibamos los parámetros `nombre` y `puntos`. En caso de disponerlos ejecutamos una función que veremos más adelante y devolvemos un mensaje de éxito por respuesta HTTP, y en caso contrario, un mensaje de error.

```
5  def main(req: func.HttpRequest) -> func.HttpResponse:
6      logging.info('Python HTTP trigger function processed a request.')
7      name = req.params.get('nombre')
8      score = req.params.get('puntos')
9      time = req.params.get('duracion')
10     date = req.params.get('fecha')
11     if name and score and time and date:
12         ret = conexionBBDD.query(name, score, time, date)
13         return func.HttpResponse(f"{name} ha jugado durante {time}
               segundos, obteniendo una puntuación de {score} a fecha de {date
               }\n{ret}")
14     else:
15         return func.HttpResponse("Debe enviarse nombre de jugador,
               puntuación, tiempo, y fecha", status_code = 200)
```

Código 2.1: Lógica principal API

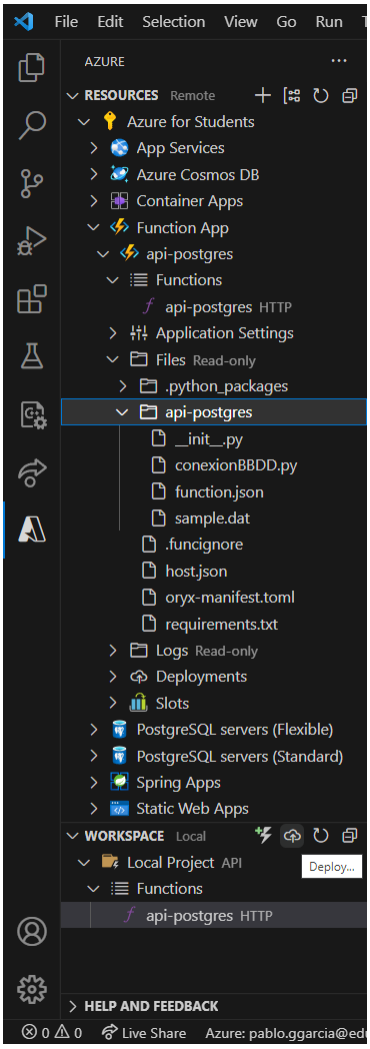

Figura 2.1: Function App mediante extensión de Azure en VSC

Si ahora nos fijamos en el fichero `conexionBBDD.py`, esta es la parte susceptible de cambiar si en un futuro se cambiase de sistema gestor de base de datos, o servidor. Como veremos en la Sección 2.2, se ha elegido PostgreSQL como SGBD, así que haremos uso del módulo psycopg. Lo que hace la función que aquí se encuentra es, de forma resumida, dado el nombre y los puntos de un jugador, los inserta en una tabla mediante la siguiente instrucción:

```
INSERT INTO partidas (nombre, puntos, duracion, fecha) VALUES (%s,%s,%s,%s)
```

Como tenemos esta dependencia, en el archivo `requirements.txt` añadiremos `psycopg2-binary==2.9.1`. Veamos un ejemplo. Supongamos que nuestro juego en Scala ha recogido que el jugador "*Sergio*" ha conseguido 300 puntos, su partida ha durado 60 segundos, y que jugó el 11 de mayo de 2023. Entonces nuestro juego solo tiene que hacer una llamada HTTP a la siguiente URL[1]:

[1]Todas las URL estarán activas hasta la publicación de las calificaciones.

```
https://api-postgres.azurewebsites.net
/api/api-postgres?nombre=Sergio&puntos=300&duracion=60&fecha=11/05/2023
```

Si la llamada se ha realizado correctamente, veremos por pantalla, o en nuestro caso Scala recibirá en el cuerpo de la respuesta:

```
Sergio ha jugado durante 60 segundos, obteniendo una puntuación de 300 a fecha de
11/05/2023
Datos insertados correctamente!
```

Si consultamos la base de datos, veremos que efectivamente existe este nuevo registro. Puede verse el código completo, en los Códigos A.1 y A.2.

2.2. Almacenamiento de datos con PostgreSQL

Para almacenar todos los datos que nuestra API recibe, usaremos una base de datos relacional, aunque no es estrictamente necesario, pues como veremos, vamos a usar una única tabla donde almacenemos nombres y puntuaciones.

Para realizar esto, se ha creado en Azure un servidor de PostgreSQL estándar. Durante su creación, como el objetivo es realizar la práctica, se han elegido las especificaciones mínimas, siendo estas:

- PostgreSQL versión 11
- Servidor básico
- Un núcleo virtual
- 5 GB de almacenamiento

Además, en la configuración de la conexión, permitimos que servicios Azure accedan a él, y desactivamos las conexiones SSL. En este servidor, creamos una base de datos llamada `candycrush`, y dentro de esta, una tabla llamada `partidas`.

```
CREATE TABLE partidas ( id SERIAL PRIMARY KEY, nombre TEXT, puntos INTEGER,
                        duracion INTEGER, fecha TEXT );
```

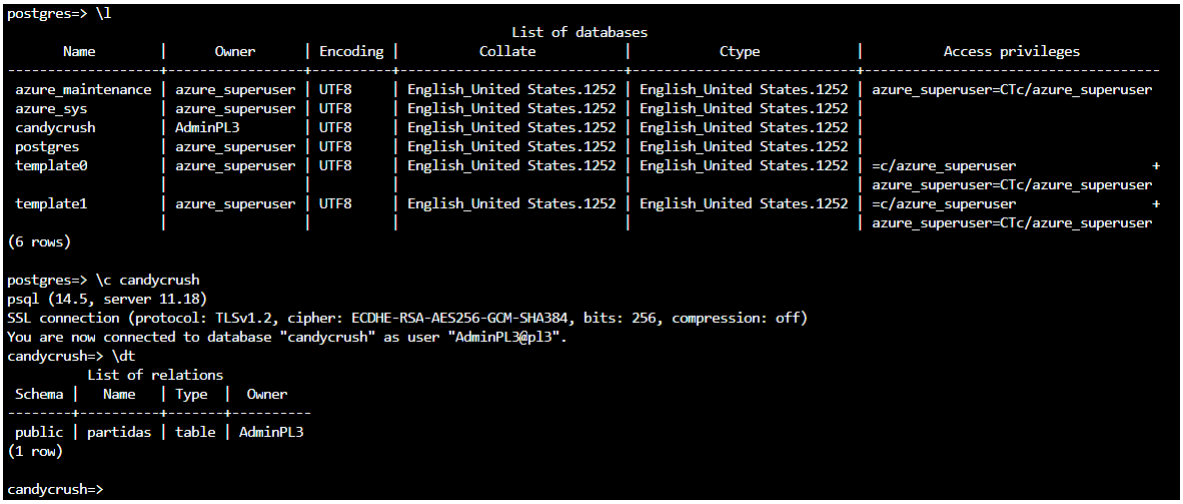

Figura 2.2: Servidor de PostgreSQL desde Azure Shell

2.3. Visualización de datos con PHP

Si bien hasta ahora somos capaces de generar datos, enviarlos, y almacenarlos de manera consistente, sería una gran idea que todos los jugadores pudieran consultar los datos del resto para que intenten competir entre ellos. Para realizar esto, se ha decidido crear una página web con ayuda de HTML y PHP, en la que se muestren las puntuaciones de los jugadores en tiempo real. En Azure se crea como una "*Aplicación web*".

Para realizar esto, creamos en PHP una clase que nos permita preguntar las puntuaciones a la base datos. Esto lo vemos en el fichero `puntuaciones.php` (Código A.5). Tiene dos métodos, el primero de ellos se conecta a la base de datos, y el segundo de ellos, devuelve una tabla en formato HTML con la propia tabla de la base de datos, ordenada por puntuaciones.

Finalmente, desde el archivo `index.php` (Código A.4) damos un poco de estilo a la página e invocamos a los métodos previamente nombrados, obteniendo el siguente resultado en el enlace `https://puntuacionescandy.azurewebsites.net`. Podemos observar cómo aparece el ejemplo de *Sergio* y 300 puntos que habíamos ejecutado antes, así como la fecha y duración. También disponemos de tres botones que nos permiten ordenar la tabla que vemos según puntuación, fecha, o tiempo. Por defecto, se ordena de mayor a menor puntuación. Podemos ver el resultado final en la Figura 2.3.

Para subir nuestra web a Azure, dentro de *Centro de Implementación*, vamos al apartado *Credenciales de FTPS*, y creamos un usuario y contraseña, pues usaremos el protocolo FTPS para subir los archivos. Una vez creado, vamos al apartado de *Introducción*, y descargamos el perfil de publicación. Instalamos en nuestro equipo el software `FileZilla`, y con este deberemos rellenar los campos servidor, usuario, y contraseña. Estos valores se corresponden con los campos `publishUrl`, `userName`, y `userPWD` del fichero que acabamos de descargar.

LEADERBOARD Scala Candy Crush

¡Intenta superar el récord!

Datos consultados a 13:52:13, 11/05/2023

Ordenar por puntuación | Ordenar por fecha | Ordenar por tiempo

Nombre	Puntos	Tiempo	Fecha
Sergio	300	60	11/05/2023
Pablo	210	42	10/05/2023
Cesar	206	68	10/05/2023
AlexanderBrun	86	28	10/05/2023
Alex	84	46	10/05/2023
Eugenio	62	41	10/05/2023
MiguelAngel	30	35	10/05/2023

Figura 2.3: Web con las puntuaciones de los jugadores

Finalmente, podemos ilustrar un resumen de cómo funciona nuestra arquitectura en la Figura 2.4.

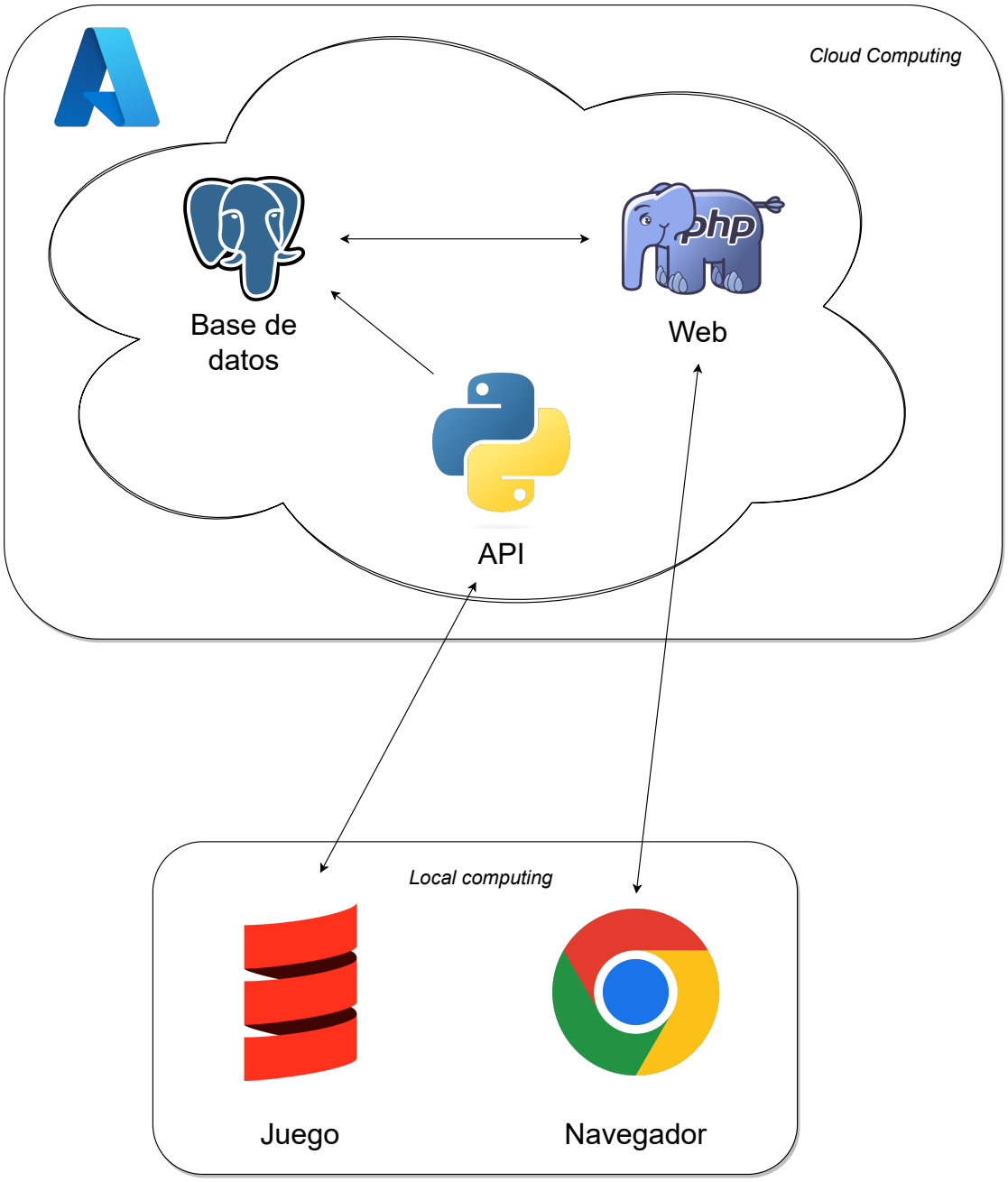

Figura 2.4: Diagrama de la arquitectura

Apéndice A

Código Cloud

A.1. API

```python
import logging
import azure.functions as func
from . import conexionBBDD

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')

    name = req.params.get('nombre')
    score = req.params.get('puntos')
    time = req.params.get('duracion')
    date = req.params.get('fecha')

    if name and score and time and date:
        ret = conexionBBDD.query(name, score, time, date)
        return func.HttpResponse(
            f"{name} ha jugado durante {time} segundos, obteniendo una puntuación de {score} a fecha de {date}\n{ret}")
    else:
        return func.HttpResponse("Debe enviarse nombre de jugador, puntuación, tiempo, y fecha",
                                 status_code = 200)
```

Código A.1: Azure Function (__init__.py)

```python
import psycopg2

def query(nombre, puntos, duracion, fecha):
    try:
        conn = psycopg2.connect(
            host = "pl3.postgres.database.azure.com",
            database = "candycrush",
            user = "AdminPL3@pl3",
            password = "uah_2023",
            port = "5432"
        )
```

```
12          cursor = conn.cursor()
13          consulta = """ INSERT INTO partidas (nombre, puntos, duracion,
                fecha) VALUES (%s, %s, %s, %s)"""
14          datosConsulta = (nombre, puntos, duracion, fecha)
15          cursor.execute(consulta, datosConsulta)
16          conn.commit()
17      except (Exception, psycopg2.Error) as error:
18          return "Error al insertar datos en Postgres"
19      finally:
20          if (conn):
21              cursor.close()
22              conn.close()
23              return "Datos insertados correctamente!"
```

Código A.2: Python ⟷ PostgreSQL (`conexionBBDD.py`)

```
1  # DO NOT include azure-functions-worker in this file
2  # The Python Worker is managed by Azure Functions platform
3  # Manually managing azure-functions-worker may cause unexpected issues
4
5  azure-functions
6  psycopg2-binary==2.9.1
```

Código A.3: Dependencias Python

A.2. Visor web

```
1  <!DOCTYPE html>
2  <html lang="es">
3      <head>
4          <meta charset="UTF-8">
5          <meta name="viewport" content="width=device-width, initial-scale
               =1.0">
6          <title>PL3 - PAP</title>
7          <style>
8          body {
9              font-family: 'Calibri';
10             font-size: 16px;
11             line-height: 1.5;
12             margin: 0;
13             padding: 0;
14         }
15         h1, h2, h3 {
16             text-align: center;
17         }
18         h3 {
19             font-size: 13px;
20         }
21         table {
22             border-collapse: collapse;
23             width: 100%;
24         }
```

```
25          th, td {
26              padding: 8px;
27              text-align: left;
28              border-bottom: 1px solid #ddd;
29          }
30          th {
31              background-color: #DC322F;
32              color: white;
33          }
34  </style>
35      </head>
36      <body>
37          <h1><u>LEADERBOARD Scala Candy Crush</u></h1>
38          <h2> Intenta superar el récord!</h2>
39          <h3>
40              <?php
41                  date_default_timezone_set('Europe/Madrid');
42                  echo "Datos consultados a " . date('H:i:s, d/m/Y ', time()
                        );
43              ?>
44          </h3>
45          <?php
46              include_once("puntuaciones.php");
47
48              if(!isset($_GET["sort"])){
49                  $sort = 0;
50              }
51              else{
52                  $sort = $_GET["sort"];
53              }
54
55              echo '<a href="index.php?sort=0"><button type="button", style
                    ="margin: 10px">Ordenar por puntuación</button></a>';
56              echo '<a href="index.php?sort=1"><button type="button", style
                    ="margin: 10px">Ordenar por fecha</button></a>';
57              echo '<a href="index.php?sort=2"><button type="button", style
                    ="margin: 10px">Ordenar por tiempo</button></a>';
58
59              $pg = new Pg();
60              $c = $pg -> conectar();
61              $pg -> consultar($c, $sort);
62          ?>
63      </body>
64  </html>
```

Código A.4: Web principal (`index.php`)

```
1  <?php
2  class Pg {
3      function conectar(){
4          $host = "pl3.postgres.database.azure.com";
5          $bd = "candycrush";
6          $usuario = "AdminPL3@pl3";
7          $pass = "uah_2023";
```

```
8
9           try{
10              $c = new PDO("pgsql:host = $host; dbname = $bd", $usuario,
                    $pass);
11              return $c;
12          }
13          catch(PDOException $ex){
14              echo "Error al conectarse con Postgres";
15          }
16      }
17
18      function consultar(PDO $c, int $sort){
19          if($sort == 0){
20              $query = "SELECT * FROM partidas ORDER BY puntos DESC;";
21          }
22          else if($sort == 1){
23              $query = "SELECT * FROM partidas ORDER BY id DESC;";
24          }
25          else{
26              $query = "SELECT * FROM partidas ORDER BY duracion DESC;";
27          }
28          $stmt = $c -> query($query);
29          $results = $stmt -> fetchAll(PDO::FETCH_ASSOC);
30          echo '<table>';
31          echo '<tr><th>Nombre</th><th>Puntos</th><th>Tiempo</th><th>Fecha</
                th></tr>';
32          foreach ($results as $row) {
33              echo '<tr>';
34              echo '<td>' . $row['nombre'] . '</td>';
35              echo '<td>' . $row['puntos'] . '</td>';
36              echo '<td>' . $row['duracion'] . '</td>';
37              echo '<td>' . $row['fecha'] . '</td>';
38              echo '</tr>';
39          }
40          echo '</table>';
41      }
42  }
43  ?>
```

Código A.5: PHP ⟷ PostgreSQL (`puntuaciones.php`)