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

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
8      name = req.params.get('nombre')
9      score = req.params.get('puntos')
10
11     if name and score:
12         ret = conexionBBDD.query(name, score)
13         return func.HttpResponse(f"He recibido {name} como nombre, que ha
               obtenido {score} puntos. \npostgres -> {ret}")
14     else:
15         return func.HttpResponse("Debe enviarse un nombre de jugador y
               puntuacion. ", status_code=200)
```

Código 2.1: Lógica principal API

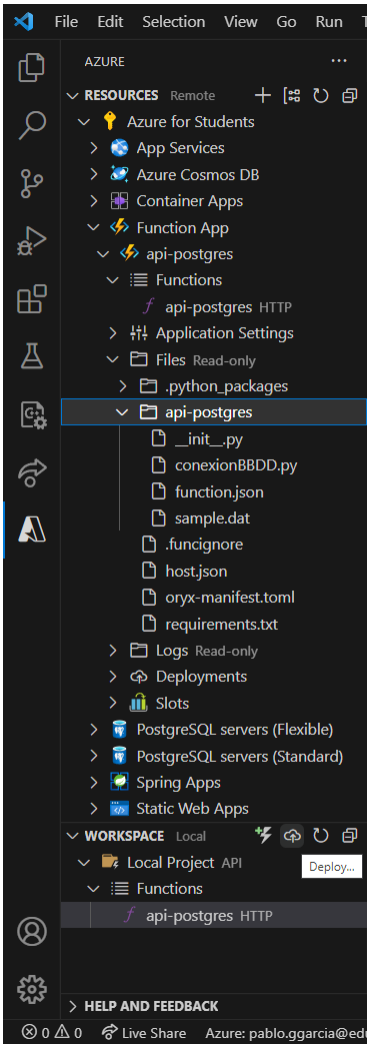


Figura 2.1: Function App mediante extensión de Azure en VSC

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) VALUES (%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 20000 puntos. Entonces nuestro juego solo tiene que hacer una llamada HTTP a la siguiente URL[1]:

```
https://api-postgres.azurewebsites.net/api/api-postgres?nombre=Sergio&puntos=20000
```

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

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

```
He recibido Sergio como nombre, que ha obtenido 20000 puntos.
postgres ->Datos insertados correctamente!
```

Si consultamos la base de datos, veremos que efectivamente existe este nuevo registro.

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 );
```

```
postgres=> \l
                                                  List of databases
      Name       |      Owner      | Encoding |         Collate          |          Ctype           |         Access privileges
-----------------+-----------------+----------+--------------------------+--------------------------+-----------------------------------
 azure_maintenance | azure_superuser | UTF8     | English_United States.1252 | English_United States.1252 | azure_superuser=CTc/azure_superuser
 azure_sys         | azure_superuser | UTF8     | English_United States.1252 | English_United States.1252 |
 candycrush        | AdminPL3        | UTF8     | English_United States.1252 | English_United States.1252 |
 postgres          | azure_superuser | UTF8     | English_United States.1252 | English_United States.1252 |
 template0         | azure_superuser | UTF8     | English_United States.1252 | English_United States.1252 | =c/azure_superuser                +
                   |                 |          |                          |                          | azure_superuser=CTc/azure_superuser
 template1         | azure_superuser | UTF8     | English_United States.1252 | English_United States.1252 | =c/azure_superuser                +
                   |                 |          |                          |                          | azure_superuser=CTc/azure_superuser
(6 rows)

postgres=> \c candycrush
psql (14.5, server 11.18)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "candycrush" as user "AdminPL3@pl3".
candycrush=> \dt
         List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+----------
 public | partidas | table | AdminPL3
(1 row)

candycrush=>
```

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*".

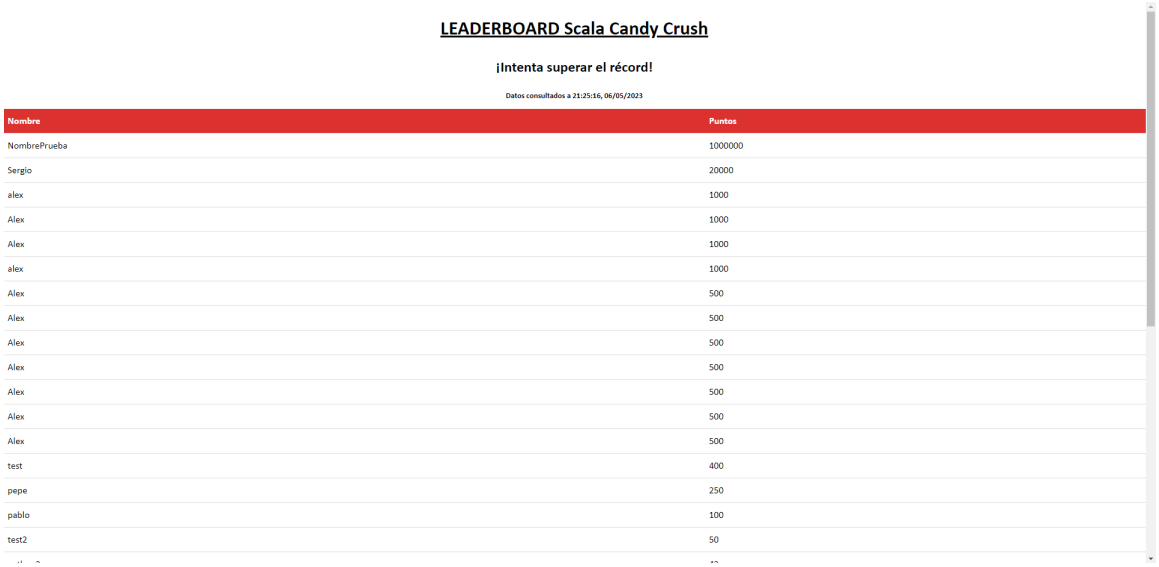
Para realizar esto, creamos en PHP una clase que nos permita preguntar las puntuaciones a la base datos. Esto lo vemos en el fichero `puntuaciones.php`. Tiene dos métodos, el primero de ellos se conecta a la base de datos, y el segundo de ellos, devuelve una tabla en formato HTML con la propia tabla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

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

        try{
            $c = new PDO("pgsql:host = $host; dbname = $bd", $usuario,
                $pass);
            return $c;
        }
        catch(PDOException $ex){
            echo "Error al conectarse con Postgres";
        }
    }

    function consultar(PDO $c){
        $query = "SELECT * FROM partidas ORDER BY puntos DESC;";
        $stmt = $c -> query($query);
        $results = $stmt -> fetchAll(PDO::FETCH_ASSOC);
        echo '<table>';
        echo '<tr><th>Nombre</th><th>Puntos</th></tr>';
        foreach ($results as $row) {
            echo '<tr>';
            echo '<td>' . $row['nombre'] . '</td>';
            echo '<td>' . $row['puntos'] . '</td>';
            echo '</tr>';
        }
        echo '</table>';
    }
}
?>
```

Código 2.2: PHP $\longleftrightarrow$ PostgreSQL

Finalmente, desde el archivo `index.php` damos un poco de estilo a la página e invocamos a los métodos previamente nombrados, obteniendo el siguiente resultado en el enla-

ce `https://puntuacionescandy.azurewebsites.net/index.php`. Podemos observar cómo aparece el ejemplo de *Sergio* y 20000 puntos que habíamos ejecutado antes.



LEADERBOARD Scala Candy Crush

¡Intenta superar el récord!

Datos consultados a 21:25:16, 06/05/2023

Nombre	Puntos
NombrePrueba	1000000
Sergio	20000
alex	1000
Alex	1000
Alex	1000
alex	1000
Alex	500
Alex	500
Alex	500
Alex	500
Alex	500
Alex	500
Alex	500
test	400
pepe	250
pablo	100
test2	50

Figura 2.3: Web con las puntuaciones de los jugadores

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.

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

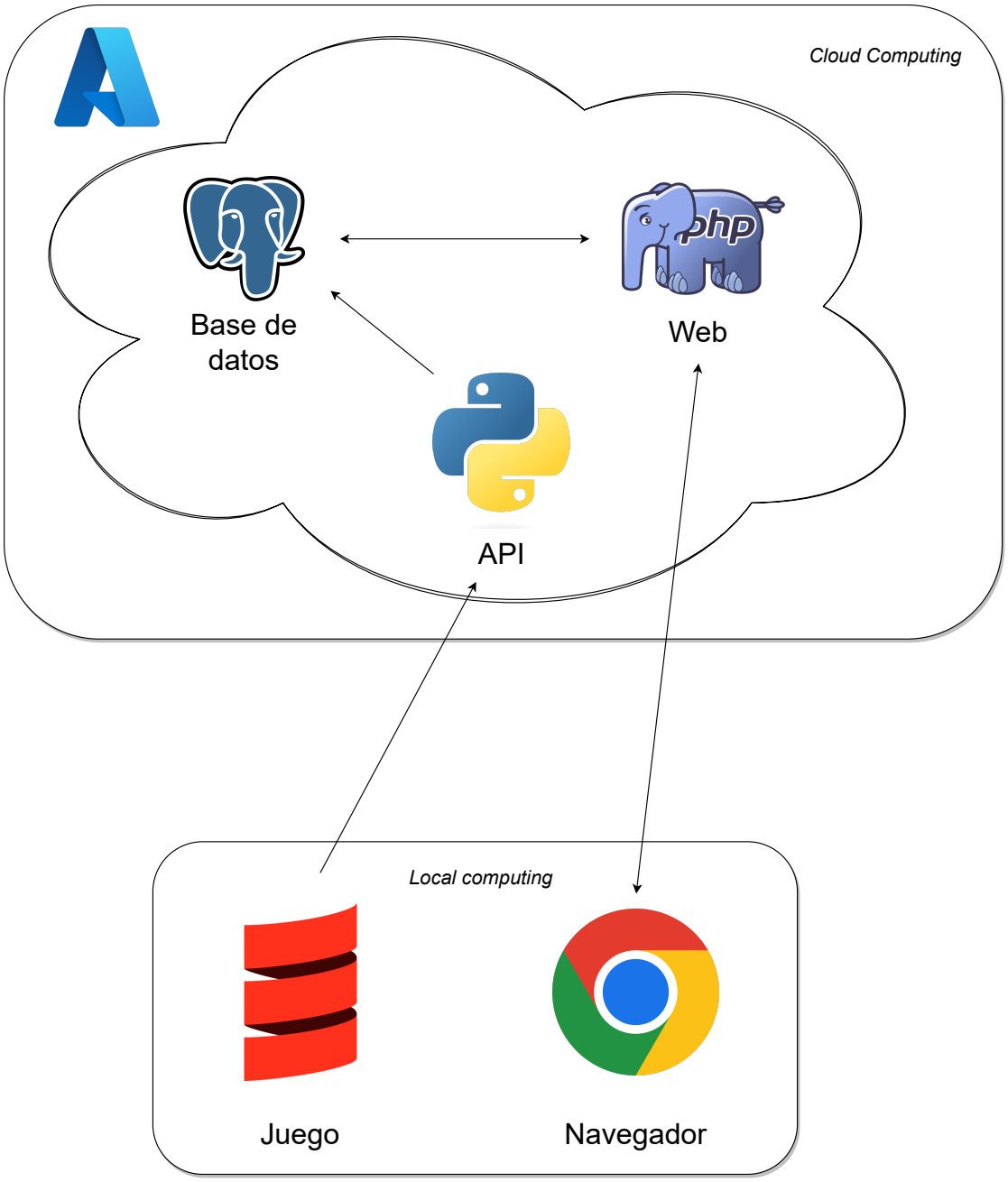


Figura 2.4: Diagrama de la arquitectura