Administración de Sistemas

Proyecto

Titulación:
Grado en Informática de Gestión y Sistemas de Información

4.º curso (1.º cuatrimestre)

Fecha: 23-11-2025

Garcia, Beaskoetxea, Andoni

Índice

1. Descripción de la aplicación

GameSales es una aplicación web para explorar, de forma rápida y visual, las ventas históricas de los videojuegos. Consulta el ranking por año y plataforma, compara títulos y entra al detalle de cada uno.

Además, incluye un apartado de *Arcade* con clásicos para echar una partida y competir por la mejor puntuación. Añade tu récord personal y explora si estás entre los mejores jugadores de Pac-Man, 2048 y/o Tetris.

Los contenedores principales utilizados en este proyecto son para el servidor web (*Node.js*) y para la base de datos (*MongoDB*).

1.1. Repositorios y datos

A continuación, se listan las fuentes de datos y los repositorios empleados para construir los contenedores y la web:

- Datos para el **ranking**: `https://www.kaggle.com/datasets/gregorut/videogamesales`
- Imagen de contenedor **Pac-Man**: `https://hub.docker.com/r/dbafromthecold/pac-man`
- Código fuente de **2048**: `https://github.com/gabrielecirulli/2048`
- Código fuente de **Tetris**: `https://github.com/jakesgordon/javascript-tetris`

1.2. Motivación del trabajo

El objetivo principal de esta web es fomentar la competición amistosa entre amigos por el mejor récord. Mostramos de un vistazo los datos de ventas históricas y buscamos despertar la pregunta: *¿por qué ciertos juegos han vendido tanto?*

Para explorar esa curiosidad, incluimos un apartado de *Arcade* y una *Leaderboard* donde se puede jugar a clásicos y comparar puntuaciones en tiempo real.

2. Enlace al repositorio de código

Consulta el repo: `https://github.com/andonigarcia240/AS_Proyecto`

3. Listado de tareas realizadas

- **Tareas obligatorias:**
  - Desarrollo de una aplicación web. ✓
  - Creación de archivos Dockerfile y otras imágenes. ✓
  - Creación de un entorno Docker Compose. ✓
- **Tareas opcionales:**
  - Creación de un despliegue Kubernetes equivalente. ✓
  - Inclusión de imágenes adicionales. ✓
  - Utilización de funcionalidades de Kubernetes no vistas en clase. ✓

4. Explicación de tareas realizadas

4.1. Docker

El fichero `docker-compose.yml` permite desplegar la aplicación completa con un único comando, enlazando la base de datos, el servidor web, las tareas de carga de datos (*seeders*) y los minijuegos del módulo *Arcade*.

4.1.1. Arquitectura de servicios

- **servidor-bbdd (MongoDB)**: almacena las colecciones de juegos y puntuaciones. Puerto **27017**.
- **servidor-web (Node.js)**: expone la API y la interfaz web de GameSales. Lee de dos bases de datos (*appdb* y *scoresdb*). Puerto **3000**.
- **seed-games** / **seed-scores**: tareas puntuales que importan datos iniciales desde ficheros CSV y finalizan al completarse.
- **pacman**, **juego-2048**, **tetris**: contenedores con los minijuegos del *Arcade*, accesibles por puertos independientes **8080**, **8081** y **8082**, respectivamente.

4.1.2. Persistencia y volúmenes

Se define un volumen nombrado **mongo-data**, montado en `/data/db`, para la persistencia de MongoDB. Además, se montan los ficheros CSV y la carpeta **data** del host dentro de `/home/app/data` del contenedor *servidor-web* para su consumo por la aplicación.

En caso de detectar **datos duplicados** en la base de datos tras ejecutar los seeders, se recomienda ejecutar los siguientes comandos:

```
docker compose run --rm seed-scores node seeds/seed-scores.js --force
docker compose run --rm seed-games node seeds/seed-games.js --force
```

4.1.3. Arranque ordenado y salud

MongoDB define un *healthcheck* con **mongosh**. Los servicios *seed-games*, *seed-scores* y *servidor-web* dependen de que la base de datos esté sana (`depends_on: condition: service_healthy`), lo que evita errores de conexión en el arranque inicial.

En cuanto a los juegos del *Arcade*, se ha omitido la dependencia de MongoDB, ya que no requieren base de datos para ejecutarse.

4.2. Kubernetes

En el despliegue de Kubernetes se han utilizado varias funcionalidades para cumplir la misma función que Docker.

4.2.1. LoadBalancer

El Service `arcade-loadbalancer` es la puerta de entrada pública a la aplicación desplegada en los Pods con la etiqueta `app: servidor-web` dentro del *namespace* `arcade`. Expone el puerto lógico 3000 del clúster y lo redirige al `targetPort`: 3000 de cada contenedor, haciendo balanceo de carga entre los Pods.

4.2.2. ConfigMap

Para poder interactuar correctamente con la imagen de MongoDB es necesario inicializar tanto el usuario *root* como un usuario de aplicación. Para ello se ha definido un `ConfigMap` que contiene la configuración y el script de inicialización. Este `ConfigMap` se monta en el contenedor de la base de datos, de forma que MongoDB crea los usuarios al arrancar y el resto de Pods pueden autenticarse utilizando esas credenciales.

4.2.3. Web

El despliegue de la parte web se divide en dos manifiestos principales: `Service` y `Deployment`.

- **Service**: el Service es de tipo `ClusterIP`, por lo que solo es accesible desde dentro del clúster, y actúa como punto de entrada para el `LoadBalancer` y para otros servicios que necesiten comunicarse con la aplicación.
- **Deployment**: este Deployment define una réplica del servidor web , la imagen `andonigarcia/servidor-web:1.0.4` que se utiliza y las variables de entorno necesarias.

4.2.4. Mongo

El apartado de Mongo en Kubernetes está dividido en cuatro manifiestos: `Service`, `Deployment`, `Secrets` y `PVC`. Aunque en esta sección solo se hace énfasis en dos.

- **Service**: El Service `servidor-bbdd` expone MongoDB dentro del *namespace* `arcade` como `ClusterIP`, accesible para el resto de Pods mediante el nombre DNS `servidor-bbdd` en el puerto 27017.
- **Deployment**: El Deployment `servidor-bbdd` levanta un Pod de MongoDB con credenciales seguras, datos persistentes, scripts de inicialización y una *readiness probe* que comprueba que la base de datos está lista.

4.2.5. Juegos

En este caso se ha decidido agrupar los tres juegos del arcade en la misma sección, ya que, aunque en Kubernetes cada uno se despliega de forma independiente, comparten la misma estructura: un `Deployment` que ejecuta la imagen del juego y un `Service` de tipo `ClusterIP` que lo expone internamente en el puerto 80.

4.2.6. Ingress

El recurso `Ingress arcade-paths` actúa como punto de entrada HTTP al clúster para el host `arcade`, y enruta las peticiones según la ruta: `/` hacia el Service `servidor-web` (puerto 3000), `/db` hacia `servidor-bbdd` (puerto 27017) y `/pacman`, `/2048` y `/tetris` hacia los Services de cada juego (puerto 80), centralizando el acceso a toda la aplicación arcade.

4.2.7. Funcionalidades no vistas en clase

- **Namespace**: Se ha creado un *namespace* para agrupar todos los recursos del proyecto en un entorno ordenado. Proporciona aislamiento lógico de nombres y configuración frente a otros despliegues del clúster. Además, para borrar todo, basta con eliminar el *namespace* `arcade`.
- **Secret**: En cuanto a la accesibilidad de Mongo, se ha puesto en práctica la utilidad *secret*. El único objetivo es almacenar las credenciales y así poder tener un control del acceso a las bases de datos.

  (**Experiencia personal**: He tenido un conflicto entre Kubernetes y Docker ya que uno tenía la contraseña original y el otro decodificada para la misma imagen. Dejando así varios errores a la hora de crear la URL para conectarse a MongoDB)
- **PersistentVolumeClaim (PVC)**: Solicita al clúster un volumen de *storage* persistente sin acoplarse a un disco físico concreto. Así, los ficheros (*seeds*) sobreviven a reinicios/recreaciones de Pods y pueden compartirse entre distintos componentes del proyecto.
- **Job**: Para la ingesta de datos en Mongo, se crearon dos imágenes para llevar a cabo esta tarea. A pesar de poder realizar esta ingesta con un *deployment*, se ha optado por hacer uso de la utilidad *job*, ya que solo nos interesa ejecutar la ingesta una vez.
- **initContainer**: Se ha sustituido el depends_on de Docker Compose por un initContainer en el Deployment de `servidor-web` y en los Job de `seed-games` y `seed-scores` que espera a que MongoDB responda antes de arrancar la aplicación.

4.3. Imágenes adicionales

Para llevar a cabo el objetivo principal de esta web, que es incentivar la competitividad amistosa en los videojuegos, se han utilizado varias imágenes de terceros:

- **Pac-Man**: Si esta web quiere ser un arcade `"oficial"`, es imperdonable olvidar este mítico juego. Esta imagen, creada por el usuario *dbafromthecold*, ha sido extraída de la página oficial de Docker Hub.

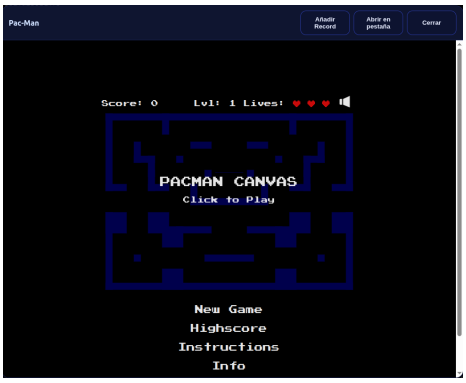

Figura 1: Imagen de Pac-Man

- **2048**: Este juego, menos popular pero adictivo, está orientado a un público menos competitivo. En este caso se ha tenido que crear una imagen propia desde un repositorio de GitHub del usuario *gabrielecirulli*.

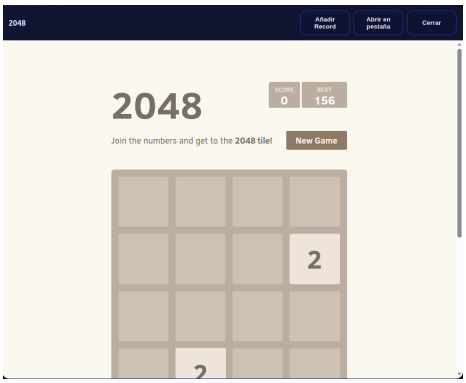

Figura 2: Imagen de 2048

- **Tetris**: Por último, pero no menos importante, el **Goat** de los clásicos. Esta imagen también ha sido creada a partir de un repositorio de GitHub de *jakesgordon*.

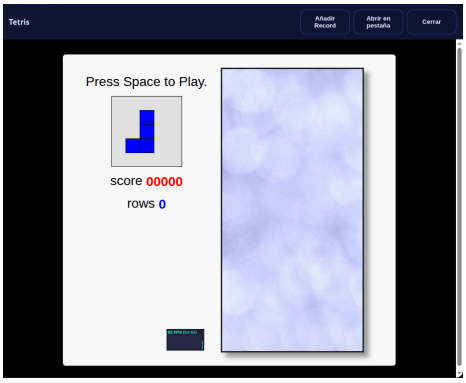

Figura 3: Imagen de Tetris

Todos ellos se controlan con las flechas del teclado y tienen disponible registrar el récord obtenido mediante el botón `Añadir Record`. Si no se incluye el nombre del jugador, se registrará como anónimo.

5. Breve manual de despliegue

5.1. Ejecución y comandos de Docker

1. **Añadir Secrets a env**
   Para tener los usuarios y contraseñas más accesibles desde Docker, se ha optado por crear un archivo `.env` con los valores en la carpeta `secrets`

```
chmod u+x guardar_env.sh
./guardar_env.sh
```

2. **Construir y arrancar**

```
docker compose up --build

# Si sale un error de permision denied por alguna imagen
sudo docker compose up --build
```

   Para acceder a la web hay buscar la IP de la MV junto con el puerto 3000: `http://34.14.113.51:3000`

   Los puertos 3000, 8080, 8081 y 8082 deberán estar habilitados, sino la web no funciona como debe.

3. **Ver registros (otra terminal)**

```
# Solo del servidor web
docker compose logs -f servidor-web

# Todos los servicios
docker compose logs -f

# Servicio concreto
docker compose logs -f servidor-bbdd
```

4. **Detener**

```
# Parar todo y borrar contenedores
docker compose down

# (Opcional) Borrar tambien volumenes
docker compose down -v

# Si no hay permisos como al arrancar
sudo docker compose down -v
```

5.2. Ejecución y comandos de Kubernetes

1. **Construir y arrancar**

```
minikube start
alias kubectl="minikube kubectl --"
kubectl apply -f Kubernetes/ --recursive
```

2. **Ver registros**

```
# Todos los pods, deployment...
kubectl -n arcade get all

# En caso de errores, comprobar logs
kubectl -n arcade logs job/seed-games
```

3. **(Opcional) Dashboard**
   Para poder ver de manera más gráfica los pods, deployments... se ha usado el Dashboard de *minikube*

```
minikube addons enable ingress
minikube addons enable metrics-server

minikube dashboard
```

4. **Abrir en el navegador**

```
minikube tunnel

# Para saber en que url se encuentra la app
minikube service -n arcade arcade-loadbalancer --url
```

5. **Detener**

```
# Parar todo y borrar
kubectl delete namespace arcade
```

6. Breve manual de uso de la aplicación

Como ya hemos mencionado anteriormente, el objetivo principal de esta aplicación es fomentar la competitividad amistosa en los videojuegos. La aplicación se divide en dos secciones: **ventas** y **arcade**.

En la sección de ventas solo está disponible el listado de ventas globales desde los años 1980 hasta 2020. Para hacer una búsqueda más específica, se ofrece la opción de filtrar las ventas por año y plataforma, hasta un máximo de 100 resultados ordenados de manera descendente.

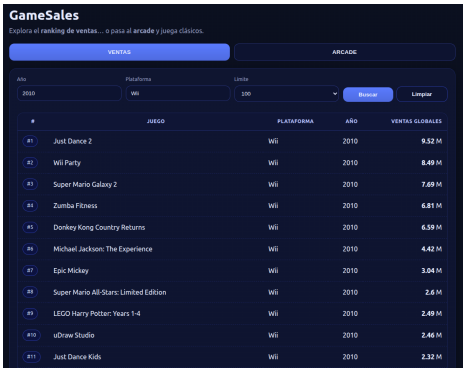

Figura 4: Interfaz de ***Ventas***

Por otro lado, la sección más importante consta de un listado de récords filtrado por los tres juegos disponibles (Pac-Man, 2048 y Tetris). A su vez, permite disfrutar de los tres juegos ya mencionados con solo hacer clic en el botón `jugar`.

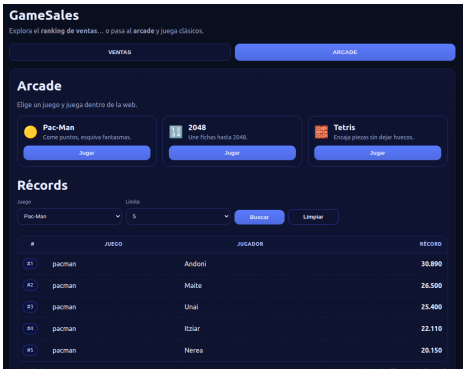

Figura 5: Interfaz de ***Arcade***

Una vez jugada una partida satisfactoria, el jugador tiene la opción de guardar su récord haciendo clic en el botón `Añadir Record`. Si el jugador no escribe su nombre en el campo `Nombre`, el récord se guardará bajo el nombre *Anon* (jugador anónimo).

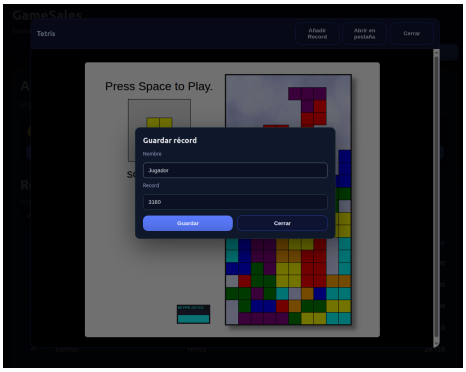

Figura 6: Interfaz de ***Añadir Record***

7. Respuestas a 4 preguntas técnicas

1) Describe un problema concreto que tuviste al construir una de las imágenes Docker y cómo lo resolviste.

Al construir la imagen del servidor web añadí una variable de entorno provisional **MONGO_URL** que apuntaba únicamente a la base de datos `appdb`. Más adelante incorporé una segunda base de datos, `scoresdb`, pero olvidé actualizar dicha variable, de modo que las operaciones relacionadas con las puntuaciones seguían yendo contra `appdb` y fallaban. Detecté el problema al revisar los logs y comprobar que las colecciones de `scoresdb` nunca llegaban a consultarse.

Para solucionarlo, pasé a configurar la URL de MongoDB desde Docker usando la ruta correcta que incluía `scoresdb`, reconstruí la imagen y verifiqué que el servidor web se conectaba a la base de datos adecuada. Al integrar Kubernetes vi que mantener una **MONGO_URL** fija complicaba los manifiestos, así que la eliminé y la reemplacé por dos variables más genéricas, **MONGO_USER** y **MONGO_PWD**, a partir de las cuales la propia imagen construye internamente la URL de conexión.

2) Explica cómo se comunican los contenedores entre sí, y qué configuraciones han sido necesarias. Es suficiente explicarlo para uno de los 2 despliegues (Docker Compose o Kubernetes).

En Kubernetes, los contenedores se comunican mediante *Services* y el DNS interno del clúster. Por ejemplo, MongoDB se expone como un Service `ClusterIP` llamado `servidor-bbdd` en el puerto 27017, y el servidor web se conecta usando la URL `servidor-bbdd:27017` sin necesitar conocer la IP del Pod. Las credenciales de acceso se inyectan como variables de entorno desde un *Secret*, y un recurso `Ingress` (`arcade-paths`) enruta las peticiones externas a los distintos Services (`servidor-web`, `pacman`, `2048`, `tetris`) según la ruta.

3) ¿Qué pasos has seguido para verificar el funcionamiento de la aplicación, antes de entregarla?

Antes de entregar, verifiqué la aplicación en los dos entornos de despliegue.

- En **Docker Compose** levanté todos los servicios y comprobé que el backend era accesible desde el navegador, creando partidas y consultando el listado de puntuaciones. Revisé los logs de la aplicación y de la base de datos para asegurarme de que no había errores ni reconexiones constantes.
- En **Kubernetes** desplegué los manifiestos, comprobé que todo iba en orden tanto con kubectl -n arcade get all, como con el dashboard que incluye minikube.

- En cuanto a la **funcionalidad de la web**, probé tanto buscar las mejores ventas globales, como jugar un par de partidas y añadir los récords obtenidos.

4) ¿Cómo has organizado el repositorio para facilitar la comprensión del proyecto? ¿Has seguido alguna guía o convención?

Para la realización de este proyecto no se ha seguido una guía de organización específica. La estructura del repositorio se ha ido definiendo de forma incremental a medida que avanzaba el desarrollo, dando lugar a la siguiente distribución de **carpetas**:

- `data`: Contiene los archivos CSV utilizados para la ingesta de datos en la base de datos de MongoDB.
- `seeds`: Incluye una carpeta por cada base de datos, cada una con su propio *Dockerfile* y el script en JavaScript encargado de la ingesta de los datos contenidos en `data`.
- `games`: Contiene el *Dockerfile* para los dos juegos incluidos, a partir de sus respectivos repositorios de GitHub.
- `mongo-init`: Contiene el script en JavaScript utilizado para inicializar los usuarios en MongoDB.
- `public`: Incluye los archivos HTML y CSS para la interfaz de la aplicación, junto con un archivo JavaScript que carga los rankings desde el backend (`/ranking`, `/leaderboard`), abre los juegos en un modal dentro de un *iframe*, lee automáticamente el marcador del juego y permite guardar récords mediante peticiones `POST` a `/record`.
- `secrets`: Almacena los usuarios y contraseñas de la aplicación web.
- `Kubernetes`: Carpeta destinada a todos los archivos *YAML* necesarios para el apartado adicional de Kubernetes.

En cuanto a los **archivos** sueltos, los más redundantes son los siguientes:

- `.env`: Archivo creado a partir del archivo `codificar.sh`, contiene los usarios y contraseñas de la carpeta `secrets` codificadas.
- `guardar_env.sh`: Ejecutable para guardar los archivos de `secrets` en un `.env`.
- `docker-compose.yml`: Archivo destinado para completar el apartado principal de Docker.
- `app.js`: Servidor principal en Express que sirve los archivos estáticos de la aplicación, expone los minijuegos mediante proxies y se conecta a MongoDB para ofrecer los endpoints `/ranking` y `/leaderboard`. Además, gestiona el guardado de récords en `/record`, almacenándolos tanto en la base de datos como en un archivo CSV.

8. Bibliografía

- MongoDB: `https://www.mongodb.com/`
- Docker: `https://www.docker.com/`
- Docker Compose (documentación): `https://docs.docker.com/compose/`
- Docker Hub – Imagen Pac-Man (dbafromthecold): `https://hub.docker.com/r/dbafromthecold/pac-man`
- Node.js (runtime servidor web): `https://nodejs.org/`
- Express (framework para Node.js): `https://expressjs.com/`
- Kubernetes (documentación oficial): `https://kubernetes.io/`
- Minikube (entorno local de Kubernetes): `https://minikube.sigs.k8s.io/docs/`
- Dataset de ventas de videojuegos (Kaggle): `https://www.kaggle.com/datasets/gregorut/videogamesales`
- Juego Pac-Man (Chregi Platzhersh): `https://github.com/platzhersh/pacman-canvas`
- Juego 2048 (Gabriele Cirulli): `https://github.com/gabrielecirulli/2048`
- Juego Tetris (Jake Gordon): `https://github.com/jakesgordon/javascript-tetris`