



BrainEx

Alumna: Silvia Condori Chambi

IES COMERCIO
DESARROLLO DE APLICACIONES WEB

Repositorios GitHub: https://github.com/SilviaCC1701/brainex_web
https://github.com/SilviaCC1701/brainex_proxy
https://github.com/SilviaCC1701/brainex_docker-compose

ÍNDICE

INTRODUCCIÓN	2
MOTIVACIÓN.....	2
OBJETIVOS	2
Objetivo general.....	2
Objetivos específicos.....	3
FASES DEL PROYECTO.....	4
ANÁLISIS.....	4
ESQUEMA ENTIDAD-RELACIÓN (UML).....	4
ACCIONES	5
DIAGRAMA DE ACTIVIDADES	6
DISEÑO	7
Nombre web proyecto	7
Logo	7
COLORES	8
FUENTE DE LETRA.....	9
BOCETOS VS RESULTADO	10
HERRAMIENTAS.....	16
ARQUITECTURA DEL PROYECTO.....	19
CONFIGURACIÓN E IMPLEMENTACIÓN	21
Paso a producción: configuración del VPS	22
Preparación del dominio	23
Certificados SSL y configuración de Nginx.....	24
Arquitectura final con GHCR	25
Otros ajustes necesarios	28
AMPLIACIÓN Y POSIBLES MEJORAS.....	28
Mejoras funcionales.....	29
Mejoras arquitectónicas y de sistema:	29
PROBLEMAS ENCONTRADOS.....	30
BIBLIOGRAFÍA	31
PILA DE TAREAS ESTABLECIDA	32

INTRODUCCIÓN

BrainEx es una plataforma web que proporciona una serie de juegos pensados para mejorar habilidades como la memoria, la agilidad mental y el razonamiento lógico.

En la web se pueden encontrar distintos minijuegos a completar, donde los usuarios pueden ejercitar su mente de forma sencilla.

MOTIVACIÓN

Desde el inicio del proyecto, una de las razones y principales motivos ha sido el deseo/voluntad de desarrollar un proyecto web que cumpla con la mayor parte de los requisitos del Trabajo de Fin de Grado, con la posibilidad que este proyecto sea escalable a largo plazo. Por ello este trabajo ha sido desarrollado simulando que fuera un servicio web público.

Sobre la elección temática del proyecto, se debe a preferencias personales. Desde joven me han atraído los juegos relacionados a la estimulación, entretenimiento y entrenamiento cognitivo de la mente. Un claro ejemplo de ello fue Brain Training, un juego que me llamó bastante la atención en su momento y que ofrecía una forma divertida de trabajar habilidades como la memoria, el cálculo o la lógica. Más recientemente, juegos del LinkedIn (Zip, Queens o Tango) o “La palabra del día” han reforzado esa misma idea, ya que es una pequeña dosis diaria de agilidad mental que es entretenida y estimulante al mismo tiempo.

OBJETIVOS

Objetivo general

El objetivo general de este proyecto web se centra en el desarrollo de una página web que ofrezca a los usuarios la posibilidad de ejercitar la agilidad mental mediante minijuegos. Por lo que he considerado que el proyecto web debe estar desplegado para añadir un toque de realidad.

Objetivos específicos

Entrando en detalles sobre mis objetivos a cerca de este TFG, me propuse lo siguiente:

- Tratar de implementar un sistema que permita a los usuarios realizar ejercicios cognitivos y lógicos para su agilidad mental. Con la posibilidad de calcular una estimación de la edad mental del usuario.
- Implementar y diseñar una estructura/arquitectura web escalable mediante el uso de contenedores Docker.
- Aprender a crear y configurar un archivo docker-compose desde cero, con el objetivo de gestionar los distintos servicios necesarios del proyecto web (como el servidor web, la base de datos o el backend) de una forma organizada.
- Hacer uso de un controlador de versiones de las imágenes Docker (github).
- Realizar el apartado del backend con una Aplicación web ASP.NET Core (Model-Vista-Controlador), con el lenguaje C# y tratando de hacer uso de las buenas prácticas de desarrollo aprendidas durante el curso.
- Desarrollar un diseño de interfaz responsive en el frontend.

FASES DEL PROYECTO

ANÁLISIS

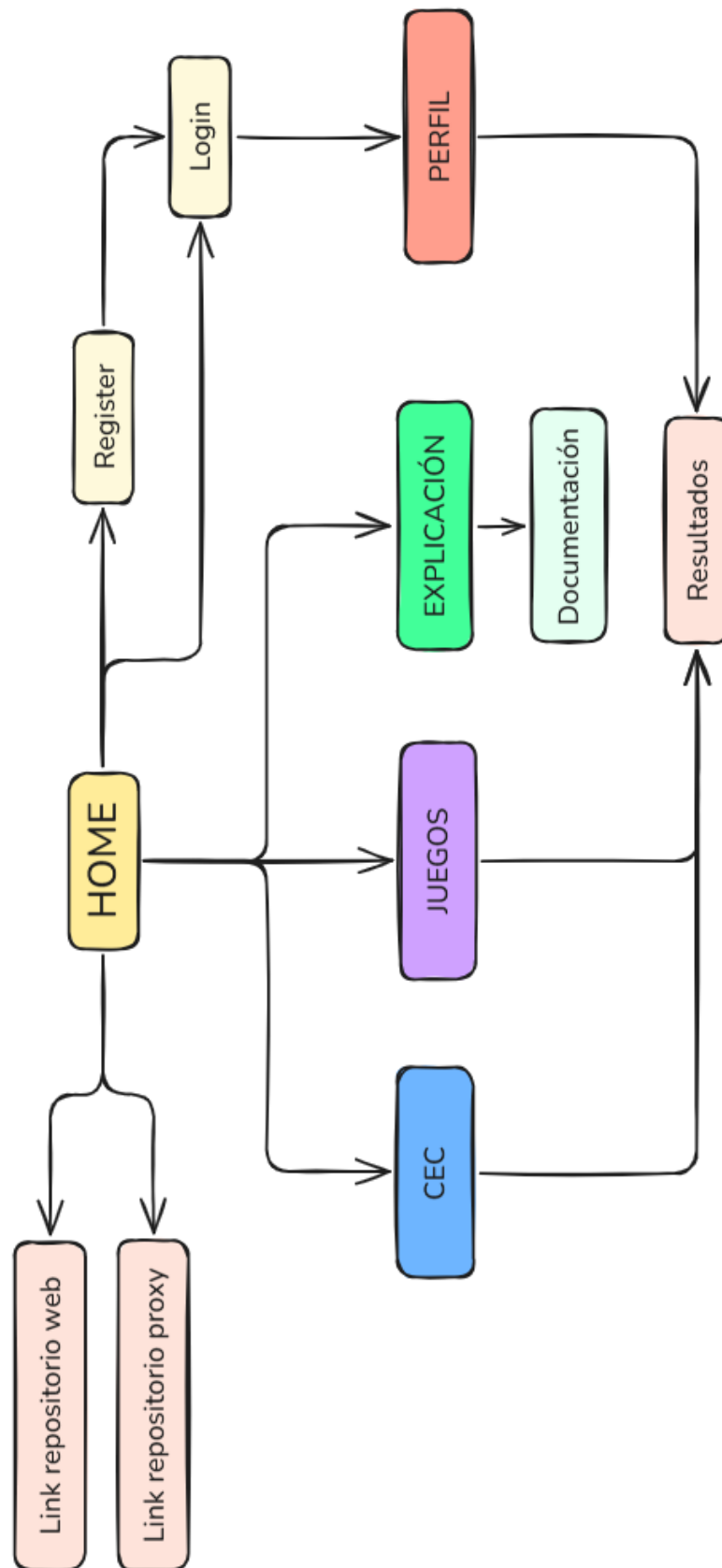
ESQUEMA ENTIDAD-RELACIÓN (UML)



ACCIONES

- **Home (sin logearse):**
 - Click Registrarse (mostrar modal)
 - Click Iniciar Sesión (mostrar modal)
 - Click Calcula tu edad cerebral (redirige a vista para iniciar una serie de juegos para calcular la edad cerebral).
 - Click Catálogo de juegos (redirige a vista Juegos)
- **Home (logueado):**
 - Click Perfil (redirige a la vista Perfil y muestra los datos y partidas del usuario).
 - Click Cerrar Sesión (cierra sesión y redirige Home)
- **Modal – Login/Register:**
 - Click botón “Registrarse” (registro en BBDD si no existe)
 - Click botón “Entrar” (inicio de sesión si existe en BBDD)
- **Perfil:**
 - Visualización de datos de usuario.
 - Visualización de una tabla con los datos de partida del usuario
 - Click en botón “Detalles” de una partida (redirige a vista de Detalles)
- **Detalles:** muestra los detalles de una partida en específica.
- **Calcula tu edad cerebral:**
 - Click Empezar (inicio de juegos).
 - Click Ver Resultados (visualización de resultados)
- **Juegos:**
 - Click Calculo Rápido (redirección al juego y al finalizar boton Resultados)
 - Click Completa Operación (redirección al juego y al finalizar boton Resultados)
 - Click Encuentra Patrón (redirección al juego y al finalizar boton Resultados)
 - Click Sigue Secuencia (redirección al juego y al finalizar boton Resultados)
 - Click Memory Game (redirección al juego y al finalizar boton Resultados)
 - Click Torre Hanoi (redirección al juego y al finalizar boton Resultados)
- **Resultados:**
 - Visualización de datos de la partida.
- **Explicación del proyecto:**
 - Visualización de la documentación del TFG.

DIAGRAMA DE ACTIVIDADES



DISEÑO

Nombre web proyecto

El nombre BrainEx está formado por dos partes en inglés: “Brain”, que significa cerebro, y “Ex”, una abreviatura que puede tener varios significados dependiendo del contexto. Justamente esa ambigüedad es lo que me pareció interesante y útil a la hora de nombrar el proyecto.

Por un lado, Ex puede venir de Exercise, es decir, ejercicio. Esto encaja perfectamente con la idea principal de la web: una plataforma con juegos que ejercitan el cerebro y ayudan a mejorar distintas habilidades mentales como la memoria, la atención o el razonamiento lógico.

Por otro lado, Ex también puede hacer referencia a Exam, es decir, examen. Esto tiene sentido especialmente porque uno de los objetivos de BrainEx es calcular la edad cerebral del usuario. A través de los diferentes juegos, se hace una especie de “evaluación” de la edad cerebral, de una forma divertida.

Además de estos dos significados principales, me gustó que el nombre suene moderno, corto y fácil de recordar. Quería un nombre que tuviera relación directa con el cerebro, pero que también diera pie a distintas interpretaciones.

En resumen, BrainEx une dos ideas clave del proyecto: entrenar la mente y evaluarla.

Logo

El logo de BrainEx es uno de los elementos clave más importantes de la identidad visual del proyecto. Desde el principio quise que fuera simple, reconocible y que representara bien la idea principal de la plataforma: juegos que entrenan la mente, pero de una forma accesible y divertida. Por eso elegí basarlo en tres figuras geométricas básicas: un círculo, un triángulo y un cuadrado.



Estas figuras no han sido elegidas al azar. Tienen un significado relacionado con el aprendizaje desde pequeños. Si pensamos en los primeros juegos educativos que usan los niños, como las cajas donde hay que encajar piezas en sus huecos correspondientes, estas formas siempre están presentes. Son de lo primero que se aprende: las figuras básicas, los colores, cómo encajan... y esa idea me parecía perfecta e idónea para reflejar

lo que quiero que transmita BrainEx. Al final, la plataforma también busca entrenar el cerebro desde lo más básico, pero en un entorno digital.

Además, todas las figuras tienen los bordes redondeados. Esto lo hice a propósito para que el logo se vea más amable y moderno. Las formas redondeadas dan una sensación más fluida, menos rígida, y encajan mejor con el estilo de la web educativa que quiere ser cercana y motivadora, no fría ni demasiado seria. No quería que pareciera una web académica o técnica, sino algo que anime a los usuarios a entrar y probar los juegos.

En cuanto a la disposición de las figuras, también está pensada: el círculo aparece primero porque es la única que no tiene vértices. Después viene el triángulo, que es la figura más simple con vértices (tiene tres), y por último el cuadrado, que ya tiene cuatro vértices. De esta forma se crea una especie de secuencia lógica, de menos a más complejidad. Esta estructura transmite orden y equilibrio, y además se reutiliza en otras partes de la web como iconos, decoraciones, botones e incluso en algunos minijuegos, para mantener una coherencia visual en toda la plataforma.

En resumen, el logo representa las ideas principales del proyecto: el aprendizaje progresivo, el juego como forma de entrenar el cerebro, y una estética sencilla pero cuidada. Quería que todo esto se viera reflejado solo con tres formas, y creo que lo he conseguido de una forma que refleja el objetivo de la página web BrainEx.

COLORES

La elección de los colores ha sido una de las partes clave a la hora de definir la identidad visual y estética de BrainEx. Desde el principio tenía la idea de que quería una paleta que transmitiera profesionalidad, pero que también fuera alegre.

Paleta de colores principal

La paleta principal está formada por colores vivos y agradables, destacando especialmente el azul, el morado y el verde, ya que son los tres colores principales que aparecen en las figuras del logo. A estos colores les acompañan otros más cálidos, como el amarillo o el coral, que ayudan a marcar elementos importantes de la interfaz, aportando contraste o destacando acciones del usuario.

#FFFFFF	#1E1E1E	#6EB5FF	#CFA1FF	#42FF99	#FFE866	#FF9E8D	#FCB45E
---------	---------	---------	---------	---------	---------	---------	---------

Paleta de colores secundaria

Se ha definido una paleta secundaria con colores pastel que funcionan como apoyo auxiliar visual en distintas secciones de la web. Estos tonos más suaves sirven para equilibrar la intensidad de la paleta principal, ayudando a que la interfaz no resulte demasiado saturada o pesada a la vista. Personalmente considero esta paleta de colores una derivación de la paleta principal.

#1E1E1E	#EAF4FF	#F1EAF4	#E4FFF1	#FEF9DB	#FEE3DA	#FEE3C3	#E8D1FF
---------	---------	---------	---------	---------	---------	---------	---------

FUENTE DE LETRA

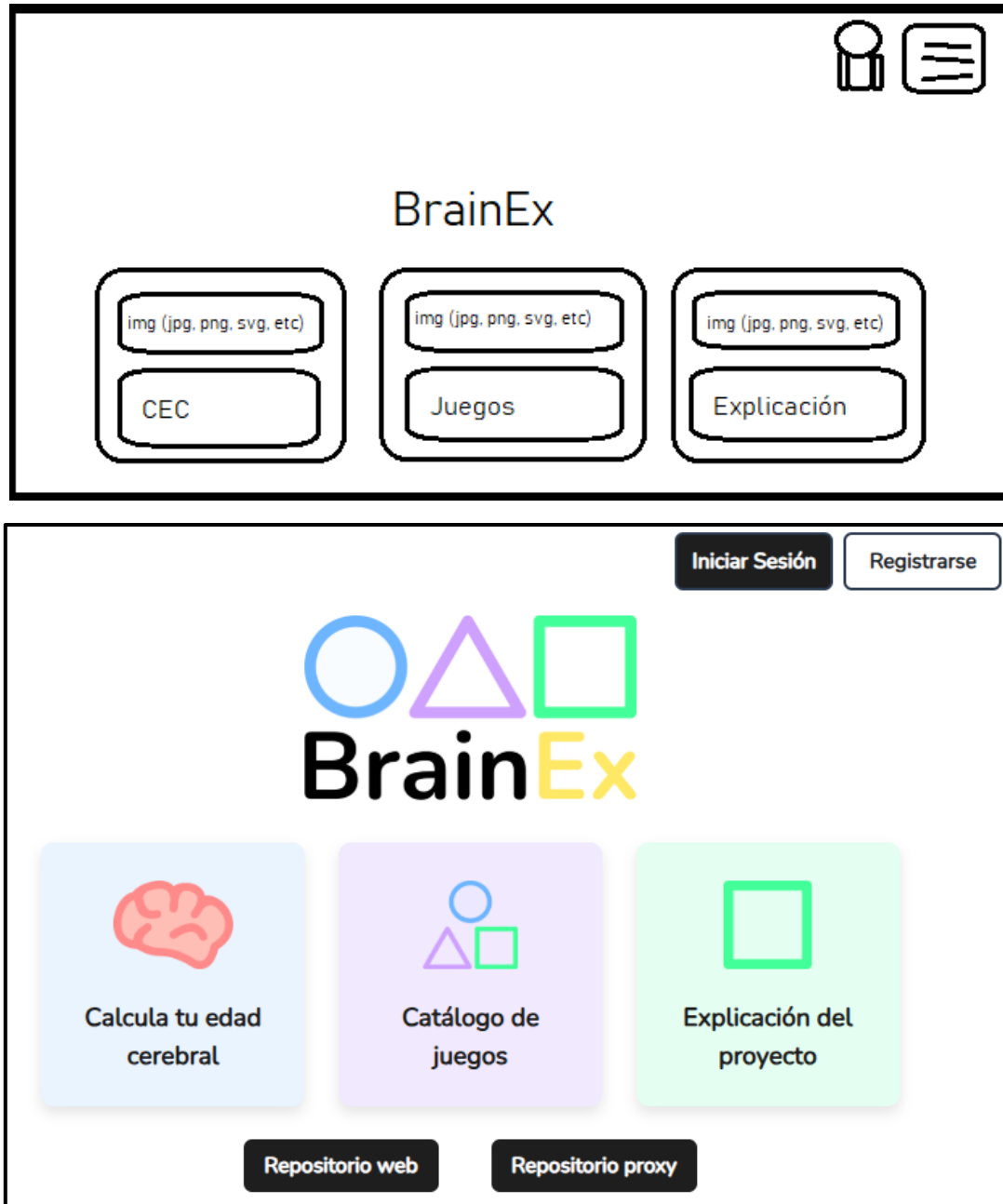
Para la tipografía principal de la plataforma he elegido Nunito, una fuente sans-serif que descubrí a través de varias recomendaciones en redes sociales, especialmente en vídeos de TikTok donde diseñadores gráficos la mencionaban como una de sus favoritas para proyectos modernos y bien cuidados. Después de probar varias opciones, me gustó tanto su estilo como su legibilidad, por lo que decidí utilizarlo en el proyecto BrainEx como fuente única y principal de la página web.

Otro punto a favor es que Nunito forma parte de Google Fonts, lo que facilita mucho su integración en la web, garantiza buena compatibilidad entre navegadores y además permite mantener el rendimiento sin tener que cargar fuentes externas.

BOCETOS VS RESULTADO

Los siguientes bocetos mockup fueron una de las referencias en cuanto a la estructura de la aplicación.

HOME



LOGIN y REGISTRO



Iniciar Sesión

Registrarse

×

Correo electrónico

Contraseña

Entrar

Iniciar Sesión

Registrarse

×

Nombre completo

Nombre de usuario

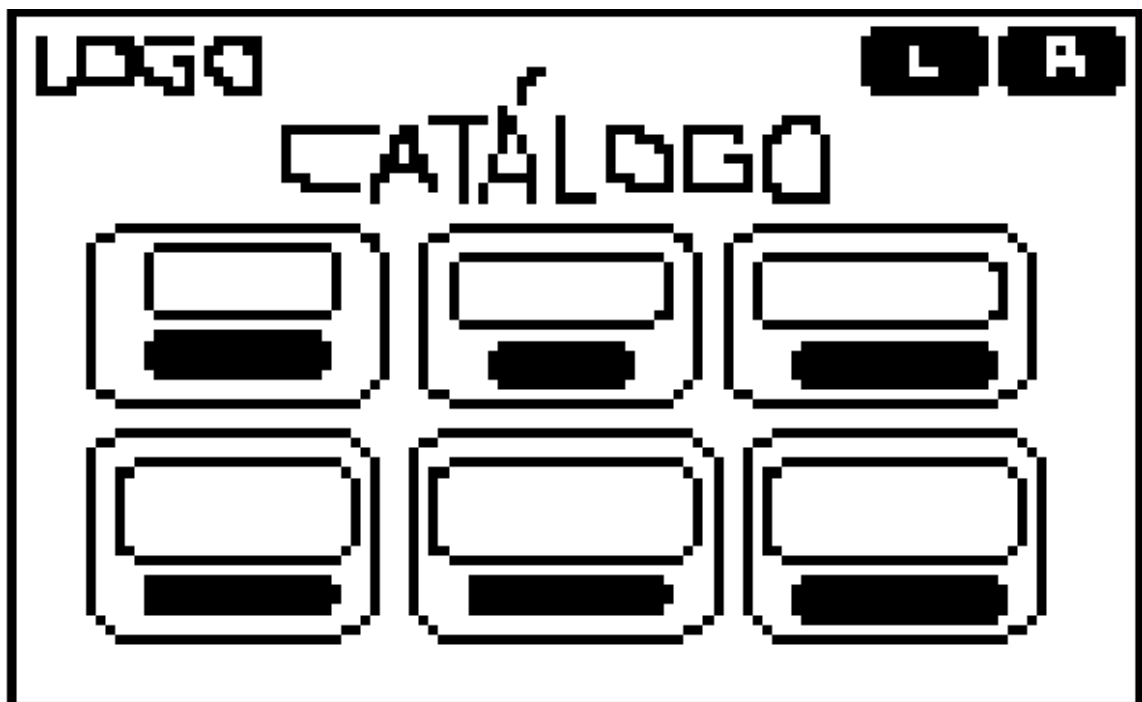
Correo electrónico

Contraseña

Confirmar contraseña

Registrarse

CATÁLOGO DE JUEGOS



BrainEx

Iniciar Sesión

Registrarse

Catálogo de Juegos

$$7+8 \\ = ?$$

Calculo Rápido

$$?+4 \\ = 6$$

Completa
Operación

$$1 \rightarrow 3 \rightarrow 5 \\ 7 \rightarrow 9 \rightarrow ?$$

Encuentra Patrón



Sigue Secuencia

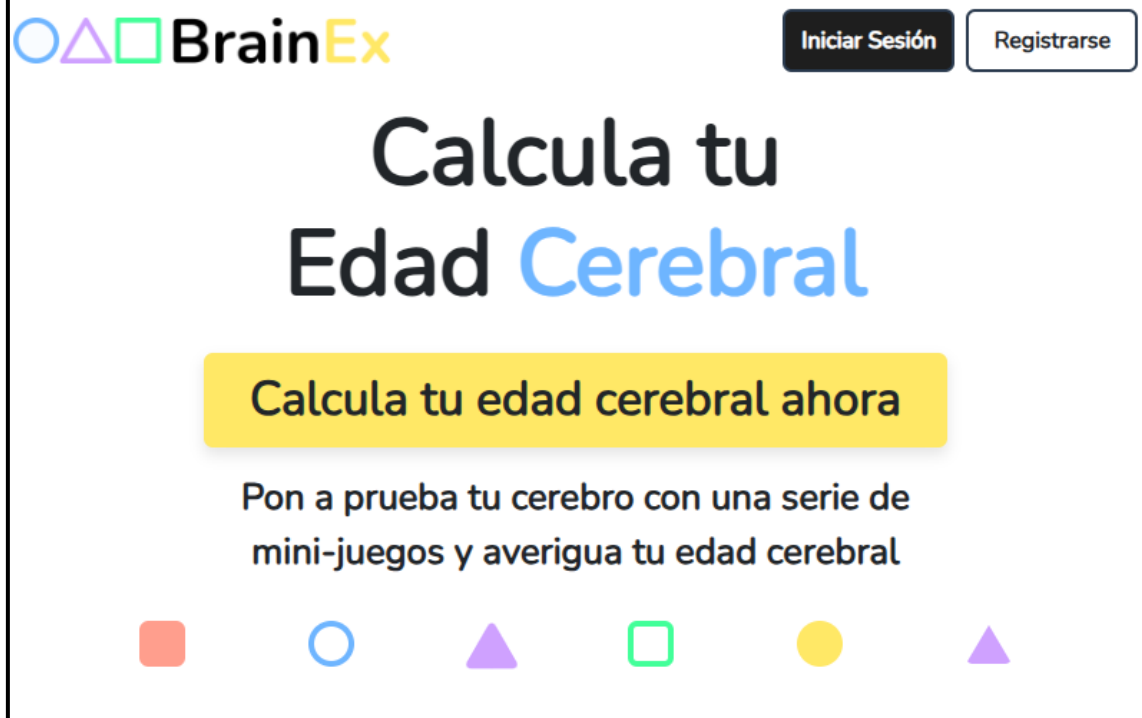
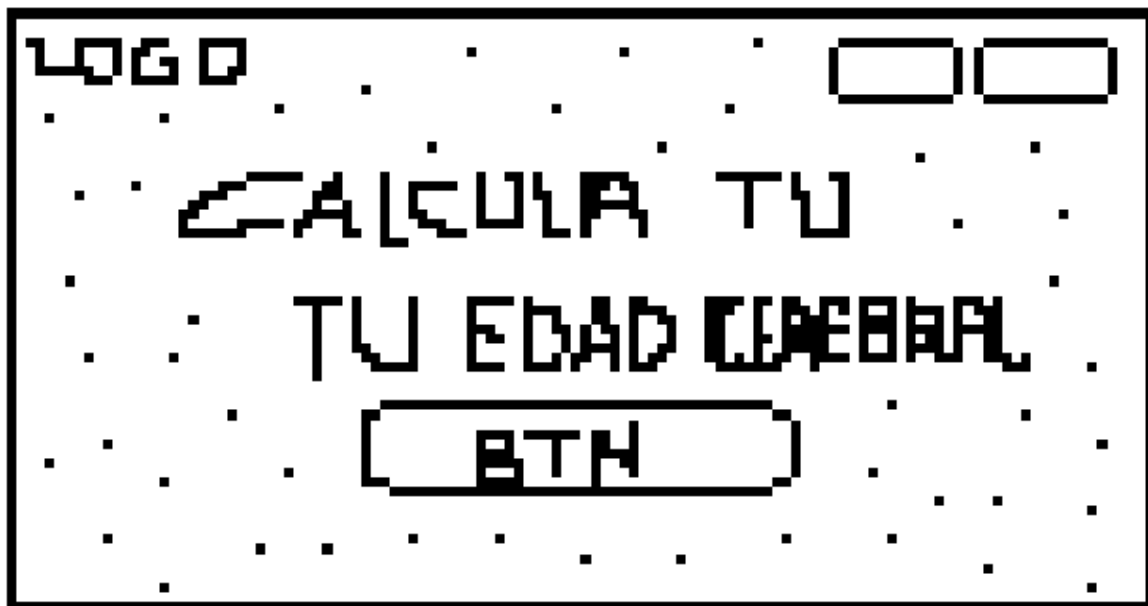


Memory Game

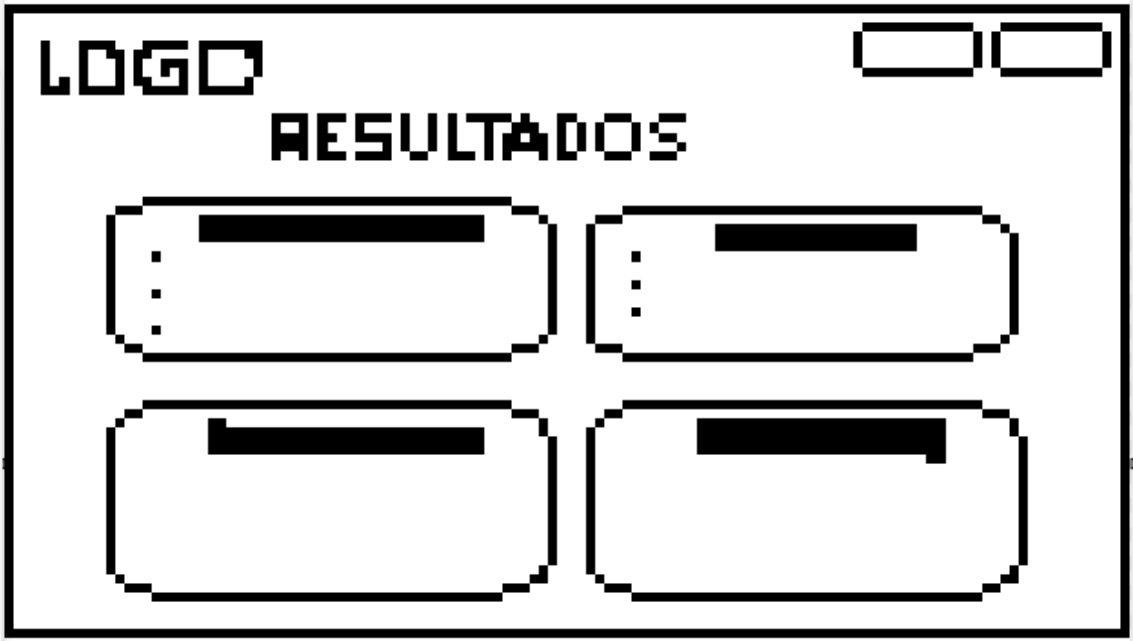


Torre Hanoi

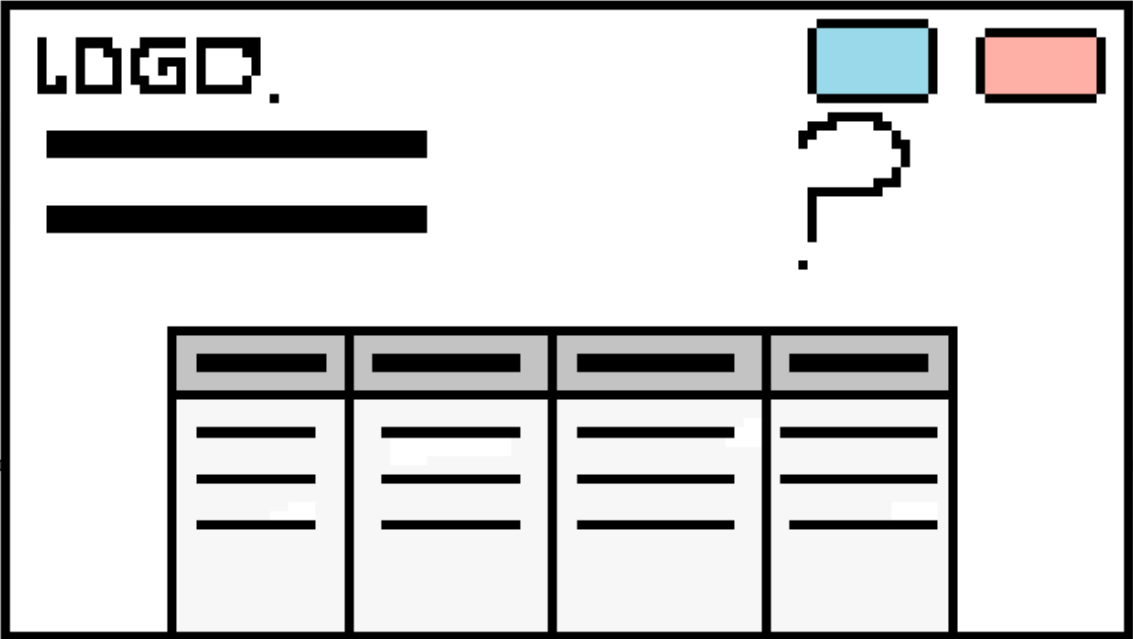
CÁLCULO EDAD CEREBRAL



RESULTADOS Y DETALLES



PERFIL





Perfil

Cerrar Sesión

Perfil de Usuario

Nombre completo: silvia

Nombre usuario: silvia

Email: silvia@email.com

Edad Cerebral

30

Partidas

Fecha	Tipo Partida	Tiempo	Acciones
03-06-2025	Cálculo Rápido	13.45 sg	<div>Detalles</div>
03-06-2025	Cálculo Rápido	32.12 sg	<div>Detalles</div>
03-06-2025	Cálculo Rápido	21.84 sg	<div>Detalles</div>

HERRAMIENTAS

.NET

Ha sido utilizado como base para desarrollar tanto el backend de la aplicación como los servicios que componen el sistema. Gracias a su versatilidad, permite crear aplicaciones web robustas, escalables y de alto rendimiento. En este proyecto, la utilizó para implementar los dos microservicios principales: la web (interfaz de usuario) y el API proxy (encargado de manejar las llamadas del cliente a la lógica del servidor).

ASP.NET

Es el framework web dentro de .NET que permite desarrollar aplicaciones web modernas utilizando el patrón Modelo-Vista-Controlador (MVC). Se ha utilizado para construir tanto la parte del backend de la web como la API, que ha sido útil por su integración con servicios y soporte para REST APIs.

C#

Es el lenguaje de programación que se ha utilizado para desarrollar toda la lógica del servidor. Su sintaxis clara y su integración con .NET lo hacen ideal para el desarrollo de servicios web escalables y mantenibles. Toda la aplicación ha sido desarrollada usando C#.

Visual Studio

Ha sido el IDE principal para el desarrollo del proyecto ya que ofrece herramientas muy completas para depurar, compilar y publicar aplicaciones ASP.NET. Además, facilita el manejo de soluciones con múltiples proyectos, como en este caso, donde se puede trabajar con varios servicios.

Paquetes NuGet

Durante el desarrollo, se han utilizado varios paquetes de NuGet para facilitar el desarrollo del proyecto y de los servicios:

- **MySql.Data:** para conectar el API proxy con la base de datos MySQL.
- **Dapper:** un micro ORM ligero para ejecutar queries de forma rápida y sencilla.
- **Microsoft.VisualStudio.Azure.Containers.Tools.Targets:** facilita el despliegue y contenedorización de aplicaciones .NET en Docker, permitiendo depurar la ejecución de un programa dockerizado.

Docker

Ha sido una herramienta clave en este proyecto, ya que ha permitido contenerizar cada uno de los servicios del sistema (web, API proxy, base de datos, y Nginx), asegurando que el entorno de desarrollo sea el mismo que el de producción, facilitando el despliegue y reduciendo errores.

GitHub

Se ha usado GitHub como sistema de control de versiones para llevar un seguimiento del código fuente. Además, se ha configurado un repositorio remoto con GitHub Actions para automatizar el build de las imágenes Docker.

GitHub Container Registry (GHCR)

En lugar de usar DockerHub, se ha utilizado GHCR para almacenar y distribuir las imágenes Docker, ya que permite integrar directamente el flujo de trabajo con GitHub y gestionar versiones de imágenes de forma segura.

Docker Compose

Docker Compose ha sido fundamental para orquestar los distintos contenedores del proyecto. Gracias a este archivo, se pueden lanzar todos los servicios del sistema con un solo comando, definiendo sus dependencias y redes internas.

Nginx

Se ha utilizado como proxy inverso. Su principal función en este proyecto ha sido redirigir el tráfico entrante del dominio hacia los servicios correspondientes, tanto para el frontend como para el API. Además, ha sido configurado para trabajar con HTTPS mediante Certbot.

MySQL

Ha sido el sistema gestor de bases de datos utilizado para almacenar toda la información de los usuarios y sus resultados. Se ejecuta en un contenedor Docker en conjunto del resto de contenedores y se inicializa automáticamente con los scripts definidos en el proyecto.

CertBot

Es la herramienta utilizada para generar certificados SSL gratuitos a través de Let's Encrypt. Esto ha permitido que los dominios personalizados funcionen con HTTPS, asegurando la comunicación entre cliente y servidor.

Namecheap

Es el proveedor donde compré el dominio personalizado “tfg-brainex.com”. Desde su panel de configuración se gestionaron los registros DNS para apuntar hacia la IP del VPS.

Hetzner

Ha sido el proveedor de hosting utilizado para desplegar el proyecto. Se contrató un VPS con Ubuntu, y allí se instalaron Docker, Docker Compose, y todos los servicios necesarios para correr la aplicación.

MobaXterm

Se utilizó este cliente SSH con interfaz gráfica para conectarse de forma remota al servidor Hetzner. Que ha facilitado la transferencia de archivos y el acceso a la consola del servidor.

HeidiSQL

Una herramienta de escritorio que se usó para conectarse a la base de datos MySQL de forma remota. Ha sido de gran ayuda para comprobar que las tablas y datos se estaban generando correctamente y para hacer pruebas directas sobre la base de datos.

GitHub Desktop

Interfaz gráfica de Git utilizada para realizar comandos git con más comodidad al momento de ver el historial o realizar commits o push y conectarse con el repositorio de Github.

Draw.io (Diagrams.net)

Es una herramienta online para crear diagramas de forma sencilla. Se ha utilizado para realizar los distintos esquemas y diagramas necesarios durante la documentación del proyecto.

Inkscape

Es un editor de gráficos vectoriales gratuito y de código abierto, que se utiliza principalmente para la creación de logotipos, iconos, ilustraciones y todo tipo de elementos gráficos escalables, utilizado aquí para diseñar el logotipo principal del proyecto BrainEx, así como los logotipos y elementos visuales de la web.

mp3cut.net

Es una página web, que permite el recorte y ajuste de audios .mp3 online y gratuito. Se ha utilizado para el ajuste de sonidos utilizados en la web, concretamente de los juegos.

ARQUITECTURA DEL PROYECTO

La arquitectura del proyecto BrainEx ha sido pensada desde el inicio para simular un sistema real, desplegado y accesible públicamente a través de Internet. Uno de los principales objetivos era no solo desarrollar la web como tal, sino montar todo el entorno de producción en un servidor propio, lo que me ha permitido aprender y aplicar conceptos relacionados con el despliegue de aplicaciones modernas basadas en contenedores (Docker).

Todo el sistema está alojado en un servidor VPS (Servidor Privado Virtual) proporcionado por Hetzner, con sistema operativo Ubuntu, donde se instalaron y configuraron las herramientas necesarias para que la aplicación pueda funcionar correctamente: Docker, Docker Compose, Nginx, Certbot, entre otras.

La arquitectura general se basa en el uso de contenedores Docker, una de las piezas clave del proyecto. Cada servicio corre en su propio contenedor de forma independiente, pero están conectados entre sí mediante una red interna definida en el archivo docker-compose.yml.

El archivo docker-compose.yml es el encargado de orquestar todos los servicios. Gracias a él, puedo iniciar toda la infraestructura del proyecto con un solo comando. En él se definen los siguientes servicios:

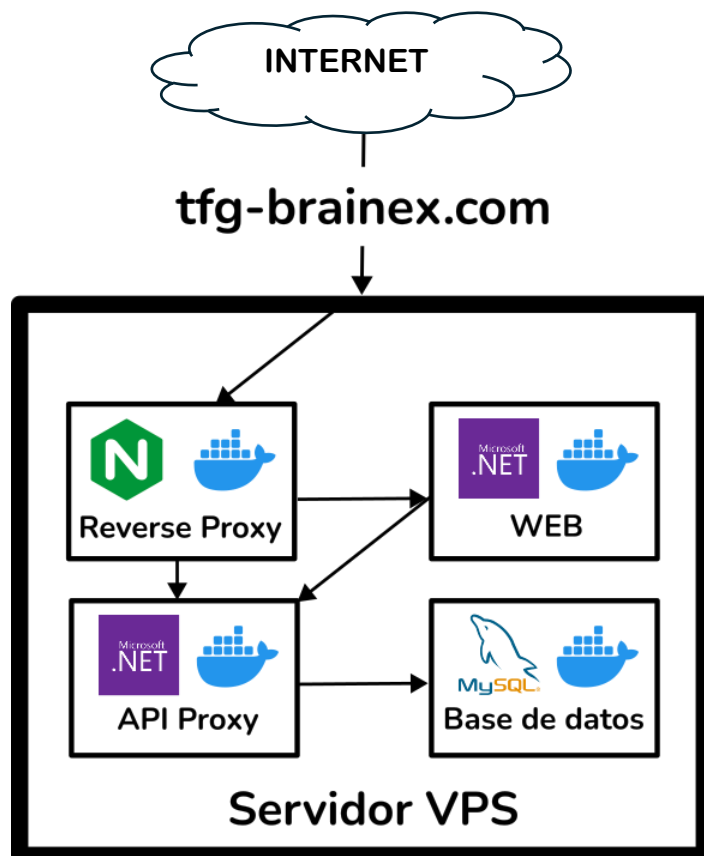
- **Web:** Es el frontend del proyecto, desarrollado con ASP.NET. Aquí es donde se encuentra la interfaz que los usuarios utilizan para interactuar con los juegos, visualizar resultados y navegar por la plataforma.
- **API Proxy:** Es el backend de la aplicación, también desarrollado con ASP.NET Core. Actúa como intermediario entre la web y la base de datos. Desde aquí se gestionan todas las peticiones relacionadas con los usuarios, puntuaciones, estadísticas, etc.
- **MySQL:** Es la base de datos que almacena toda la información relacionada con el sistema: usuarios, partidas, resultados, etc. Este contenedor también se inicializa automáticamente con scripts personalizados para crear la estructura necesaria.

- **Nginx:** Es el proxy inverso que se encarga de recibir todas las peticiones externas hacia el servidor (HTTP/HTTPS) y redirigirlas al contenedor correspondiente (ya sea el de la web o el de la API). También se ha configurado para trabajar con certificados SSL generados por Certbot para asegurar las conexiones con HTTPS.

Gracias a esta separación en contenedores, cada parte del sistema puede gestionarse de forma independiente, lo cual facilita mucho las tareas de mantenimiento, actualización y escalabilidad.

Además, para que la aplicación sea accesible desde cualquier lugar, compré el dominio tfg-brainex.com a través de Namecheap, y lo configuré para que apunte a la IP del servidor VPS. Esto se hizo mediante la edición de los registros DNS en el panel de control de Namecheap. Posteriormente, utilicé Certbot para generar los certificados SSL de los subdominios configurados (tfg-brainex.com, www.tfg-brainex.com y proxy.tfg-brainex.com).

En resumen, la arquitectura de BrainEx está compuesta por varios servicios contenedorizados que interactúan entre sí dentro de un mismo servidor, y que se exponen al exterior mediante un dominio personalizado y con conexión segura. Esta estructura no solo cumple los objetivos del proyecto, sino que además permite que BrainEx esté montado de forma similar a cómo lo haría una aplicación real en producción.



CONFIGURACIÓN E IMPLEMENTACIÓN

Desde el principio quise trabajar en local con un entorno que se acercara lo máximo posible al entorno final de producción. Por eso, una de las primeras cosas que hice fue instalar Docker y Docker Compose en mi propio equipo.

Para hacer las primeras pruebas, Visual Studio me generó automáticamente un archivo Dockerfile en cada uno de los proyectos al activar la opción de compatibilidad con contenedores. Esto me permitió crear contenedores tanto para la web como para la API proxy, y lanzarlos individualmente. Al mismo tiempo, levantaba un contenedor de MySQL de forma local. Así pude hacer pruebas de forma simultánea entre servicios y verificar que todos se comunicaban correctamente entre sí.

Gracias a trabajar con contenedores desde el principio, el número de problemas se redujo considerablemente, ya que todo se ejecutaba de una forma muy parecida a cómo se haría en el servidor. Esto me dio seguridad y confianza de que lo que desarrollaba en local iba a funcionar después en producción.

En lugar de utilizar el archivo launchSettings.json, me centré en configurar correctamente los archivos appsettings.json y en declarar variables de entorno, especialmente la cadena de conexión a la base de datos (DB_CONNECTION_STRING). Estas variables se pasaban directamente a los contenedores y permitían que el servicio supiera a qué base de datos conectarse.

Una vez tenía los servicios funcionando de forma individual, el siguiente paso fue “orquestrarlos” todos desde un único lugar. Para ello creé un archivo docker-compose.yml, donde definía los tres servicios (web, API proxy y base de datos) junto a sus configuraciones. En esta primera fase no usaba imágenes externas, sino que el docker-compose usaba la opción build, apuntando directamente al código fuente local. Para que esto funcionara, tuve que indicar la ruta al Dockerfile, y dentro del Dockerfile, la ruta correcta hacia el .csproj, que es el punto de entrada necesario para compilar la aplicación.

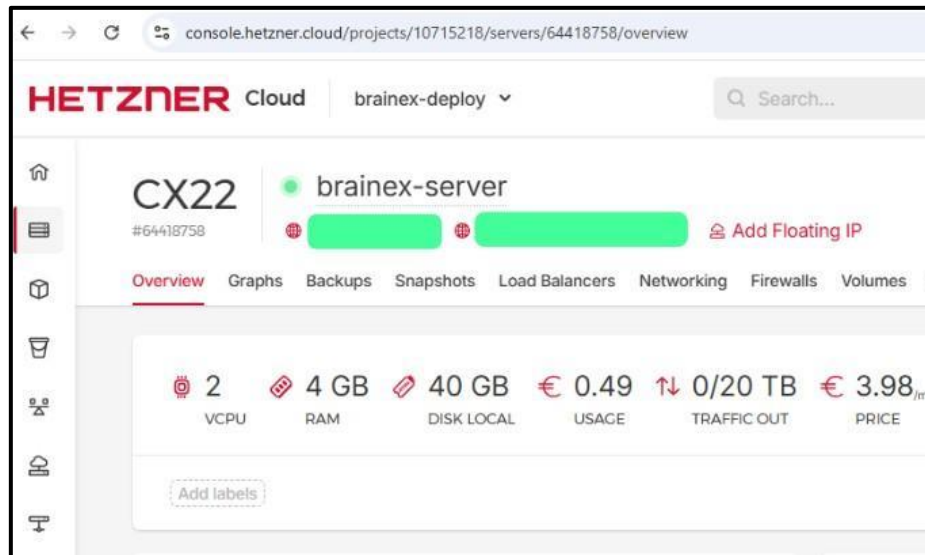
Además, para evitar tener que modificar constantemente el docker-compose.yml, creé un archivo .env en la raíz del proyecto. En él declaré todas las variables de entorno necesarias para el funcionamiento de los servicios, incluyendo contraseñas, cadenas de conexión y valores del entorno.

Gracias a toda esta configuración, pude levantar todo el entorno de desarrollo con un solo comando (docker-compose up -d) y empezar a hacer pruebas de integración de forma mucho más ágil. Esta fase me permitió validar que los servicios se conectaban correctamente, que la base de datos funcionaba y que todo el sistema respondía como se esperaba.

Paso a producción: configuración del VPS

Una vez que todo funcionaba bien en local, pasé a desplegar el proyecto en un entorno real. Para ello, contraté un servidor VPS en Hetzner, proveedor que elegí por su buena relación calidad-precio, rendimiento y facilidad de gestión.

Concretamente contraté el modelo CX22, con un coste mensual de 4,69 €, ubicado en el centro de datos de Helsinki. Este servidor incluye 2 vCPU, 4 GB de RAM y 40 GB de disco SSD, lo cual me pareció suficiente para las necesidades del proyecto.



Desde el panel de Hetzner desplegué una máquina virtual con Ubuntu, y una vez obtenida la IP pública, me conecté al servidor mediante MobaXterm, un cliente SSH con interfaz gráfica que me facilitó mucho la conexión remota, transferencia de archivos y gestión del sistema.

Nada más acceder al VPS, actualicé los paquetes e instalé Docker y Docker Compose. Después, subí todos los archivos del proyecto al servidor: el docker-compose.yml, los Dockerfile, las carpetas de configuración como nginx, el .env, y los scripts de base de datos iniciales.

Preparación del dominio

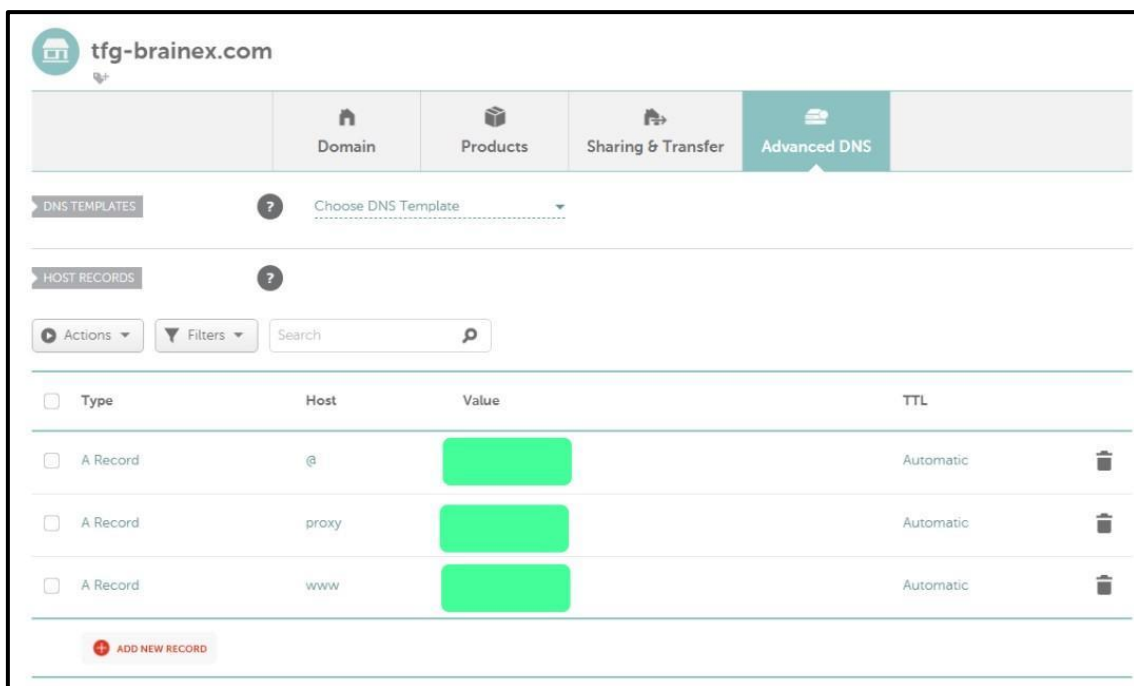
Para el dominio, elegí Namecheap por ser un proveedor económico, fácil de usar y por permitir registrar dominios .com. Compré el dominio tfg-brainex.com, que sería la entrada principal a la plataforma.

Desde el panel de control de Namecheap configuré los siguientes registros DNS:

- Un registro A para @ apuntando a la IP del VPS
- Un registro A para www apuntando a la misma IP
- Un registro A para proxy también apuntando a la IP del servidor

De esta manera, los subdominios quedaron asignados así:

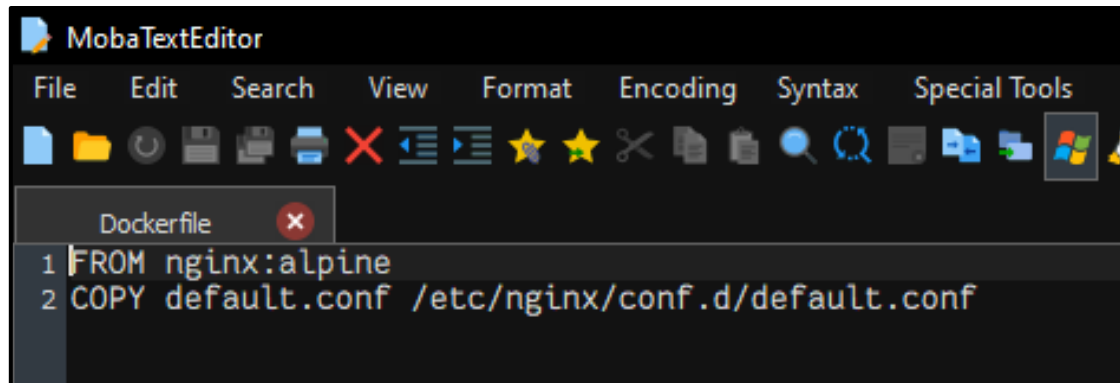
- tfg-brainex.com y www.tfg-brainex.com → web
- proxy.tfg-brainex.com → API proxy



Certificados SSL y configuración de Nginx

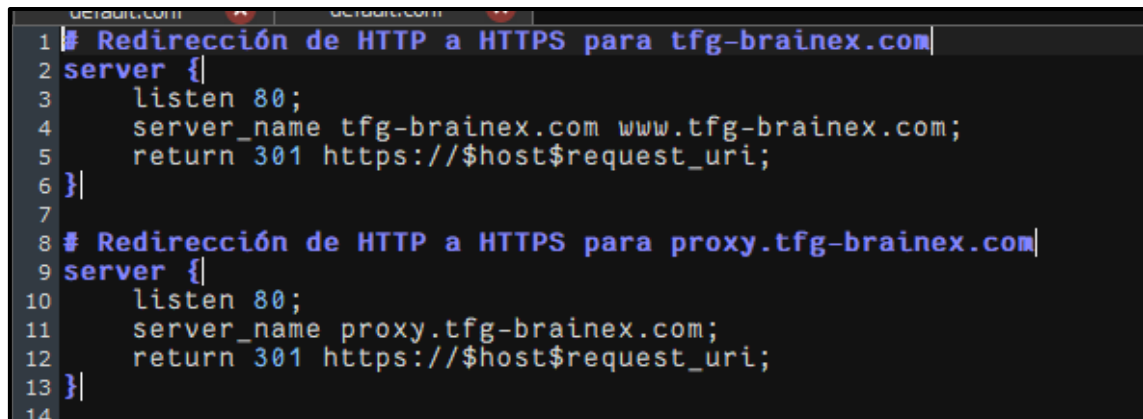
Para habilitar el uso de HTTPS en el servidor, instalé Certbot y utilicé Let's Encrypt para generar certificados SSL gratuitos. Ejecuté el comando `certbot --nginx`, el cual detectó automáticamente los dominios configurados y generó los certificados correspondientes, almacenándolos en la ruta `/etc/letsencrypt`.

El contenedor de Nginx que utilicé es personalizado. A partir de la imagen base `nginx:alpine`, copié un archivo `default.conf` que define los distintos servidores virtuales y redirecciones.



```
Dockerfile
1 FROM nginx:alpine
2 COPY default.conf /etc/nginx/conf.d/default.conf
```

El archivo de configuración `default.conf` está preparado para redirigir automáticamente el tráfico HTTP a HTTPS y para enrutar correctamente las peticiones entrantes según el dominio.



```
1 # Redirección de HTTP a HTTPS para tfg-brainex.com
2 server {
3     listen 80;
4     server_name tfg-brainex.com www.tfg-brainex.com;
5     return 301 https://$host$request_uri;
6 }
7
8 # Redirección de HTTP a HTTPS para proxy.tfg-brainex.com
9 server {
10    listen 80;
11    server_name proxy.tfg-brainex.com;
12    return 301 https://$host$request_uri;
13 }
14
```

Y para el tráfico seguro:

```
14
15 # HTTPS para web (tfg-brainex.com)|
16 server {
17     listen 443 ssl;
18     server_name tfg-brainex.com www.tfg-brainex.com;
19
20     ssl_certificate /etc/letsencrypt/live/tfg-brainex.com/fullchain.pem;
21     ssl_certificate_key /etc/letsencrypt/live/tfg-brainex.com/privkey.pem;
22
23     location / {
24         proxy_pass http://web:8080;
25         proxy_set_header Host $host;
26         proxy_set_header X-Real-IP $remote_addr;
27     }
28 }|
29
30 # HTTPS para proxy (api)|
31 server {
32     listen 443 ssl;
33     server_name proxy.tfg-brainex.com;
34
35     ssl_certificate /etc/letsencrypt/live/tfg-brainex.com/fullchain.pem;
36     ssl_certificate_key /etc/letsencrypt/live/tfg-brainex.com/privkey.pem;
37
38     location / {
39         proxy_pass http://proxy:8080;
40         proxy_set_header Host $host;
41         proxy_set_header X-Real-IP $remote_addr;
42     }
43 }|
44
```

Arquitectura final con GHCR

Una vez validado el funcionamiento del sistema en local, decidí preparar una arquitectura de despliegue más profesional, basada en imágenes Docker versionadas y almacenadas en un registro de contenedores. Para ello opté por utilizar GitHub Container Registry (GHCR), que me permitía aprovechar el mismo entorno donde ya tenía mis repositorios y mantener todo el flujo de trabajo centralizado.

Para automatizar la generación de imágenes Docker y su publicación en GHCR, tuve que crear una carpeta especial dentro de cada uno de los repositorios de mis servicios (brainex_web y brainex_proxy), llamada .github/workflows. Dentro de esa carpeta creé un archivo YAML llamado docker-build.yml, que es el archivo donde se define el workflow que ejecuta la construcción y publicación de las imágenes.

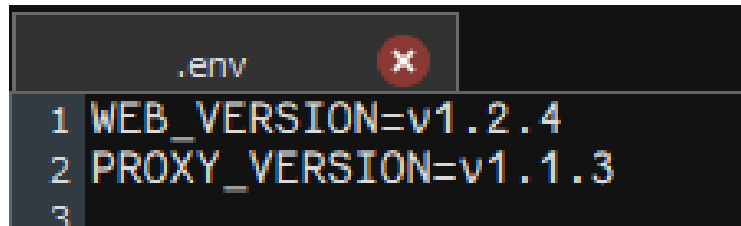
Este archivo docker-build.yml contiene las instrucciones necesarias para:

1. Detectar cuándo se realiza un nuevo release con una etiqueta tipo -v1.0.0
2. Realizar un docker build del proyecto, indicando exactamente el Dockerfile que debe usar.
3. Especificar el context y el path correcto al .csproj dentro del Dockerfile, para compilar el proyecto correctamente.
4. Etiquetar la imagen con el nombre del release (v*).
5. Iniciar sesión en GHCR.
6. Publicar automáticamente la imagen en el registro de GitHub

En el caso del proyecto de la web la imagen se publica como: ghcr.io/silviacc1701/brainex_web:v1.0.1

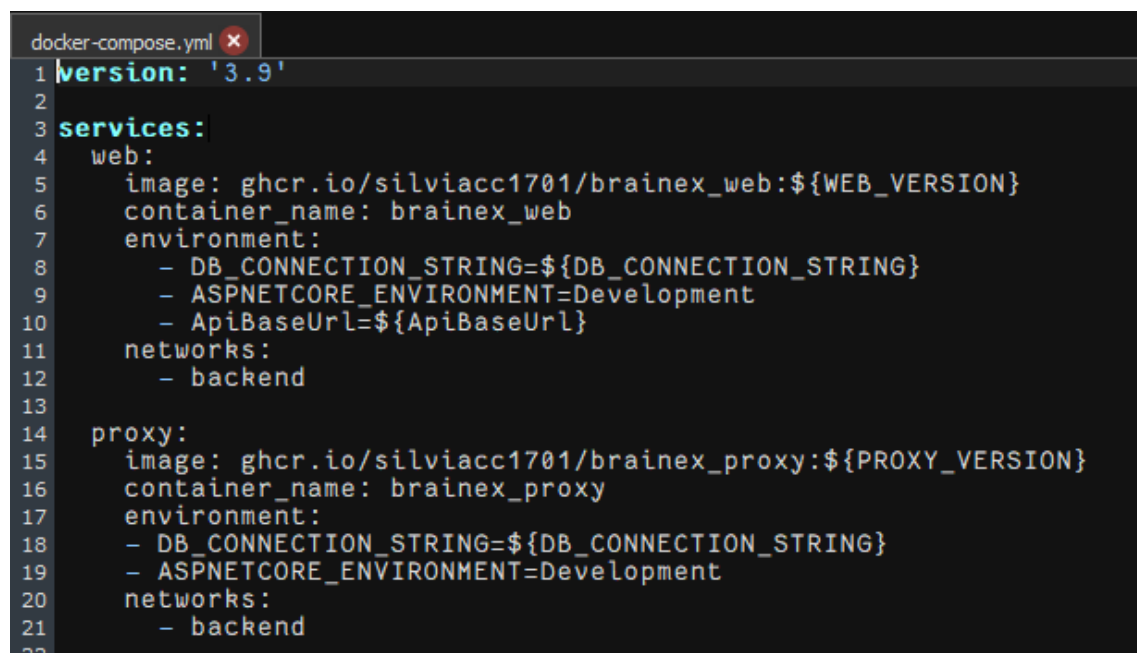
Este flujo de trabajo se activa automáticamente con cada release y no requiere intervención manual. Una vez la imagen está publicada en GHCR, ya está lista para ser usada en el docker-compose.yml del servidor.

Para poder utilizar estas imágenes desde el servidor VPS, en el archivo .env del proyecto indico las versiones que quiero ejecutar:



```
.env
1 WEB_VERSION=v1.2.4
2 PROXY_VERSION=v1.1.3
3
```

Y en el docker-compose.yml, cada servicio hace referencia a esas variables, lo que me permite cambiar la versión de una imagen fácilmente sin tener que modificar el archivo principal:



```
docker-compose.yml
1 version: '3.9'
2
3 services:
4   web:
5     image: ghcr.io/silviacc1701/brainex_web:${WEB_VERSION}
6     container_name: brainex_web
7     environment:
8       - DB_CONNECTION_STRING=${DB_CONNECTION_STRING}
9       - ASPNETCORE_ENVIRONMENT=Development
10      - ApiBaseUrl=${ApiBaseUrl}
11     networks:
12       - backend
13
14   proxy:
15     image: ghcr.io/silviacc1701/brainex_proxy:${PROXY_VERSION}
16     container_name: brainex_proxy
17     environment:
18       - DB_CONNECTION_STRING=${DB_CONNECTION_STRING}
19       - ASPNETCORE_ENVIRONMENT=Development
20     networks:
21       - backend
22
```

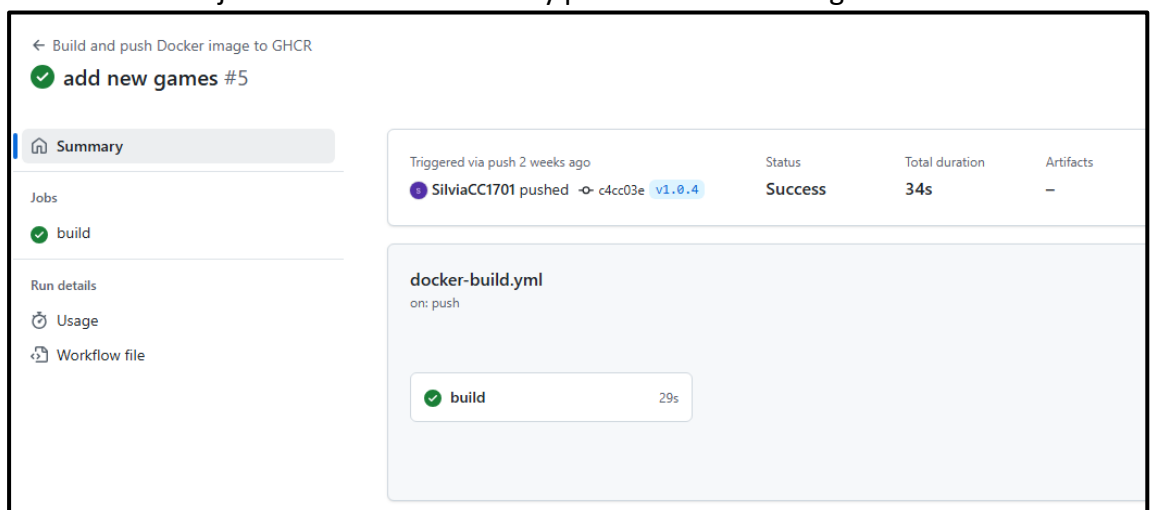
Este sistema de versionado me ha permitido tener un control total sobre qué versión del código está ejecutándose en producción, facilitando actualizaciones, pruebas y mantenibilidad.

Cuando quiero desplegar una nueva versión de alguno de los servicios, simplemente:

1. Hago los cambios en el código.
2. Hago commit y push a GitHub.
3. Creo un nuevo release con la etiqueta correspondiente (por ejemplo v1.0.4).



4. El workflow se ejecuta automáticamente y publica la nueva imagen en GHCR.



5. En el servidor, actualizo el archivo “.env” con la nueva versión.
6. Y finalmente ejecuto:

```
root@brainex-server:~# cd brainex-deploy/
root@brainex-server:~/brainex-deploy# docker-compose up -d web
WARN[0000] /root/brainex-deploy/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 10/10
✔ web Pulled
✔ 3dae71ba6b94 Already exists
✔ 0022984bc3f7 Already exists
✔ 152867e67c59 Already exists
✔ 6a8899a0ba52 Already exists
✔ 79714b2f18d5 Already exists
✔ 54a79b02e795 Already exists
✔ 6f5d0a9ecc0a Pull complete
✔ 4fa7b70def54 Pull complete
✔ 2897f08f2b2f Pull complete
[+] Running 1/1
✔ Container brainex_web Started
root@brainex-server:~/brainex-deploy#
```

Con este comando, solo se actualiza ese servicio en concreto sin afectar al resto del sistema.

Gracias a esta arquitectura, el despliegue de nuevas versiones es limpio, rápido y predecible, y puedo estar segura de que el código en producción es exactamente el que corresponde a cada release publicado en GitHub. Además, todo el sistema está preparado para escalar y adaptarse en el futuro a nuevas necesidades o servicios adicionales.

```
root@brainex-server:~/brainex-deploy# docker-compose ps
WARN[0000] /root/brainex-deploy/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
NAME                IMAGE                                COMMAND                                SERVICE    CREATED   STATUS    PORTS
brainex_proxy        ghcr.io/silviacci1701/brainex_proxy:v1.1.3    "dotnet ProxyBrainEx..."    proxy      2 days ago    Up 2 days    8080-8081/tcp
brainex_web          ghcr.io/silviacci1701/brainex_web:v1.2.5      "dotnet BrainEx.dll"         web        2 minutes ago    Up 2 minutes    8080-8081/tcp
mysql_db             mysql:8.0                                     "docker-entrypoint.s..."    db         2 weeks ago    Up 2 weeks    0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp
reverse_proxy       brainex-deploy-nginx                        "/docker-entrypoint..."    nginx      2 weeks ago    Up 2 weeks    0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0
```

Otros ajustes necesarios

Durante el desarrollo, para facilitar la resolución de nombres de dominio internos, edité el archivo `/etc/hosts` del servidor para que `web.localhost` y `proxy.localhost` apuntaran a la IP del VPS.

También aseguré que el volumen de la base de datos estuviera correctamente montado, para que los datos se conservaran entre reinicios. Además, incluí un script de inicialización SQL que se ejecuta automáticamente al levantar el contenedor de MySQL por primera vez.

```
db:
  image: mysql:8.0
  container_name: mysql_db
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
    MYSQL_DATABASE: ${MYSQL_DATABASE}
  ports:
    - "3306:3306"
  volumes:|
    - dbdata:/var/lib/mysql
    - ./mysql-init:/docker-entrypoint-initdb.d
  networks:
    - backend
```

En resumen, todo el proceso de configuración e implementación ha sido diseñado para imitar una infraestructura real: servicios contenedorizados, automatización de versiones, uso de dominios públicos, certificados SSL y un despliegue reproducible en cualquier entorno. Gracias a Docker y Docker Compose, el sistema es escalable, modular y fácil de mantener.

AMPLIACIÓN Y POSIBLES MEJORAS

Aunque el proyecto BrainEx se encuentra completamente funcional y desplegado, desde el inicio he tenido en mente que esta plataforma pueda seguir creciendo y

evolucionando con el tiempo. A continuación, mencionaré alguna de las mejoras y ampliaciones que me gustaría implementar en el futuro, tanto a nivel funcional (orientado al usuario) como técnico (a nivel de arquitectura y despliegue).

Mejoras funcionales

- **Más juegos:** Uno de los objetivos principales es ampliar la cantidad de minijuegos disponibles. Esto no solo ofrecería más variedad al usuario, sino que permitiría trabajar diferentes áreas cognitivas con más precisión.
- **Añadir niveles de dificultad:** Añadir distintos niveles de dificultad en los juegos permitiría adaptar la experiencia al nivel del usuario, desde principiantes hasta usuarios avanzados, haciendo la plataforma más desafiante y entretenida.
- **Mejoras en el perfil de usuario:** Actualmente, la funcionalidad del perfil es básica. Se podrían añadir opciones como: subir foto de perfil, editar datos personales o ver evolución histórica.
- **Compartir resultados:** Incluir la opción de compartir los resultados de los juegos en redes sociales (X, Instagram, WhatsApp...) para fomentar la participación y viralidad de la plataforma.
- **Modo claro y oscuro:** Implementar un sistema de temas personalizables, con opción para elegir entre modo claro u oscuro, mejorando la accesibilidad visual y adaptabilidad.
- **Sistema de rankings:** Crear un modo ranking donde los usuarios puedan ver su posición diaria, semanal o general, motivando así la competición sana y la retención.
- **Juego diario tipo Wordle:** Un único reto mental diario, con puntuación especial y posibilidad de compartir resultados, puede fomentar el hábito de visitar la web a diario, como lo ha hecho Wordle.
- **Versión en inglés:** Traducir toda la interfaz al inglés para hacer la plataforma accesible a un público más amplio y global.

Mejoras arquitectónicas y de sistema:

Servidor BACK: Separar el backend y el frontend en servidores distintos permite mejorar el rendimiento, la seguridad y la escalabilidad del sistema. Esta división reduce la carga en cada servidor y facilita el mantenimiento independiente de cada servicio.

Escalado horizontal y balanceador de carga: Si el tráfico crece, una posible evolución sería separar los servicios en varias instancias y balancear el tráfico entre ellas mediante Nginx u otro sistema (HAProxy, Traefik, etc.).

Monitorización del sistema: Añadir herramientas de monitorización como Prometheus + Grafana o simplemente UptimeRobot, para detectar caídas o errores en tiempo real.

Backups automáticos: Automatizar copias de seguridad de la base de datos y ficheros importantes para asegurar la integridad de los datos frente a posibles errores o pérdidas.

PROBLEMAS ENCONTRADOS

Durante el desarrollo de BrainEx me encontré con varios problemas técnicos que me obligaron a investigar, probar y buscar soluciones. Muchos de estos errores se debieron al uso de herramientas nuevas como Docker, GitHub Actions o Certbot, pero gracias a ellos también aprendí cosas nuevas y gané experiencia en entornos reales. A continuación, enumero los más relevantes:

- Uno de los primeros problemas fue conectarme a la base de datos cuando lanzaba contenedores Docker de forma independiente en la primera fase de desarrollo. La API no lograba encontrar el contenedor de MySQL. Después de investigar, descubrí que cuando los contenedores no están en la misma red o no están orquestados con Docker Compose, hay que usar `host.docker.internal` como host en la cadena de conexión. Esa fue la solución: `Server=host.docker.internal;`...
- Tuve un error parecido al anterior al desplegar los servicios, ya que especifica `localhost` y el puerto en la cadena de conexión a la base de datos, al final descubrí que dentro de una red Docker hay que usar el nombre del servicio (`db`) como host en la cadena de conexión, ya que Docker crea un DNS interno entre contenedores.
- Me confundí con las rutas relativas dentro del Dockerfile, sobre todo al indicar el path al `.csproj`. Un pequeño error en la ruta hacía que el build fallara completamente.
- Otro error que me costó detectar fue que al crear imágenes para GHCR, el nombre de la imagen no puede contener mayúsculas. Al principio, en el `docker-build.yml`, estaba utilizando el nombre de mi usuario con mayúsculas (como aparecía en GitHub), lo cual hacía que el proceso fallara. La solución fue poner todo en minúsculas, tanto en el nombre del usuario como en el del repositorio, siguiendo la convención de GHCR.
- Un problema inesperado me ocurrió justo después de contratar el VPS: al arrancar el sistema por primera vez, me pedía que estableciera una contraseña de root. Elegí una contraseña segura, pero con caracteres que requerían usar la tecla Mayús para escribirlos. Como estaba usando la consola web integrada del panel de Hetzner, descubrí más tarde que esa consola no siempre detecta correctamente las pulsaciones de mayúsculas. Esto provocaba que se guardara una contraseña distinta a la que yo pensaba que había puesto. Como no conseguía entrar ni recuperar el acceso, tuve que eliminar y reinstalar el sistema del VPS un par de veces hasta que me di cuenta de que el error venía del uso de esa tecla en la consola web.

BIBLIOGRAFÍA

ASP.NET Core Documentation (Microsoft): <https://learn.microsoft.com/es-es/aspnet/core/>

NGINX Documentation (Oficial): <https://nginx.org/en/docs/>

Certbot (Let's Encrypt): <https://certbot.eff.org/>

GitHub Actions Documentation: <https://docs.github.com/en/actions>

GitHub Container Registry (GHC): <https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry>

.NET y DOCKER! - Cómo desplegar tus aplicaciones// Containers, .NET, Dockerfiles y MUCHO MÁS!: <https://www.youtube.com/watch?v=8WO-WIZ9NoA>

Aprende Docker ahora! curso completo gratis desde cero!:
<https://www.youtube.com/watch?v=4Dko5W96WHg>

Hetzner tutorial - Deploy Ubuntu on Hetzner cloud and log in via SSH:
<https://www.youtube.com/watch?v=mnFQ2mGJnXI>

Cómo crear un Servidor Web Compartido con Docker y Nginx Proxy Manager (NPM):
<https://www.youtube.com/watch?v=WCSdh37Z6Wk>

How to Enable HTTPS Using a Free SSL Certificate from Certbot:
<https://www.youtube.com/watch?v=WPPBO-QpiJO&t>

Bootstrap: <https://getbootstrap.com/>

PILA DE TAREAS ESTABLECIDA

Tareas	Tiempo estimado	Tiempo realizado
CREACIÓN DE LOS PROYECTOS	15 min.	15 min.
CONFIGURACIÓN DE LOS PROYECTOS	30 min.	1 hr.
CONFIGURACIÓN DEL ENTORNO	30 min.	45 min.
ESTRUCTURACIÓN DE LOS PROYECTOS	1 hr 30 min	1 hr
CREACIÓN DE PÁGINAS	45 min.	30 min.
BOCETOS DE PÁGINAS	45 min.	30 min.
DISEÑO SIMPLE Y LÓGICA PÁGINAS	3 hr.	3 hr.
LÓGICA DE LOS JUEGOS	6 hr.	12 hr
IMPLEMENTACIÓN DE ESTILOS EN PÁGINAS	2 hr.	2 hr. 15 min
DISEÑO Y CREACIÓN DE LOS LOGOS	1 hr.	40 min.
CONFIGURACIÓN GITHUB Y GHCR	30 min.	45 min.
CONFIGURACIÓN ENTORNO DE PRODUCCIÓN	2 hr.	2 hr. 30 min.
DESARROLLO PETICIONES BBDD PROXY	3 hr.	3 hr. 30 min.
TOTAL	21 hr. 45 min.	28 hr. 40 min.

CONCLUSIONES

Realizar el proyecto BrainEx ha sido una experiencia algo compleja, pero enriquecedora a nivel personal en el ámbito del aprendizaje. Desde el inicio del desarrollo del proyecto tenía una idea clara de cómo desarrollar pero con muchas dudas sobre mis capacidades sobre la viabilidad de finalizar el proyecto completo como lo tenía idealizado.

A lo largo del proceso he trabajado con tecnologías y herramientas que no había utilizado hasta ahora, como Docker, GitHub Actions o Nginx, y he conseguido integrarlas en un flujo de desarrollo estable y profesional. Me he sorprendido ver lo mucho que se puede automatizar y optimizar cuando las herramientas están bien configuradas.

Estoy especialmente satisfecha con haber logrado desplegar BrainEx en un servidor real, accesible desde cualquier navegador, con su dominio propio. Ver la plataforma funcionando públicamente ha sido una de las partes más gratificantes del proyecto.

En resumen, este TFG no solo me ha servido para consolidar mis conocimientos en desarrollo web, sino también para ampliar mis habilidades en los distintos ámbitos como en sistemas, despliegue, arquitectura y algunas buenas prácticas de programación. Me voy con la sensación de haber creado algo sólido y útil, y con muchas ideas para seguir mejorándolo en el futuro.