Universidad Autónoma del Estado de México

Facultad de Ingeniería

Ingeniería en Computación

TECNOLOGÍAS COMPUTACIONALES I

REPORTE DE PROYECTO DE IMCUFIDE

Implementación de Plataforma Web con Arquitectura de Contenedores

DOCENTE:

DR. JOSE ANTONIO HERNÁNDEZ SERVIN

ELABORÓ:

LUIS ANTONIO CEBALLOS ARRIAGA

WILIBALDO ESQUIVEL DIAZ

EDGAR GERMAIN GONZALEZ SUAREZ

2025-B

Hoja de Evaluación

Puntos especificos considerados para evaluar el proyecto y acreditar la materia.

Tecnologías computacionales I, 2025b
Dr. J. Servín

1. (2 points) Portada

Nombres de la institución, organismo académico.	1
Título, Unidad de aprendizaje, Nombre del profesor	1
Nombres de los integrantes y fecha de entrega del reporte.	1
Criterios de evaluación	1
	2.00

2. (3 points) Objetivo

Es claro, cuantificable y bien estructurado	1
(comenzar con verbo en infinitivo, seguido del ¿cómo?,	1
y finalizando con el ¿para qué?).	1
	3.00

3. (15 points) Introducción

Claridad de los términos que tienen que ver con la experimentación y resultados,	1
se aborda la problemática planteada desde una postura o enfoque,	1
se hace uso de citas textuales con un solo estilo de referencias.	1
Máximo una sola cuartilla en extensión.	1
No hay faltas de ortografía y se usan adecuadamente las referencias.	1
	15.00

4. (35 points) Modelo

El o los modelos matemáticos están completos y sin errores.	1
La rutina de Polymath se puede ejecutar sin errores.	1
Explica los cambios realizados en el modelo para alcanzar el objetivo planteado.	1
	35.00

5. (25 points) Discusión

Presenta todos los resultados establecidos en el protocolo de la práctica de manera ordenada y con una breve descripción.	1
Interpreta y discute, a partir de principios disciplinares, datos importantes,	1
identificando tendencias, relaciones y diferencias útiles para el logro del objetivo.	1
	20.00

6. (2 points) Conclusión

Explican de manera precisa si, a través de los resultados,	1
se logró el objetivo planteado.	1
En caso de existir limitaciones en el estudio,	1
se propone soluciones y aportaciones para mejorar la práctica.	1
	2.00

7. (5 points) Referencias

Todas las fuentes que se utilizan son relevantes en la disciplina,	1
permiten la discusión de resultados a partir de un marco de referencia apropiado.	1
El documento se encuentra referenciado en estilo Vancouver.	1
Utilizar al menos 3 referencias: libro, artículo y sitio de internet.	1
	5.00

Índice General

1 Objetivo

1.1 Objetivo General

El objetivo central y estratégico de este proyecto es diseñar, desarrollar e implementar una plataforma web integral, dinámica y altamente responsiva para el Instituto Municipal de Cultura Física y Deporte (IMCUFIDE) de Tenango del Valle. Este desarrollo tecnológico se lleva a cabo mediante la aplicación rigurosa y metódica de una arquitectura de software moderna y desacoplada en tres capas fundamentales (Frontend, Backend, Base de Datos) y la implementación de una infraestructura de despliegue robusta basada en contenedores Docker.

La finalidad última (para qué) de este esfuerzo tecnológico es centralizar y digitalizar la gestión operativa de las diversas ligas deportivas municipales, automatizar la consulta pública de información crítica para la comunidad (como calendarios de juegos, tablas de posiciones, perfiles de equipos y resultados en tiempo real) y proveer un canal de comunicación oficial, robusto, seguro y escalable. Este canal busca mejorar significativamente la accesibilidad a la información, la transparencia en la gestión deportiva y fomentar la participación activa de la ciudadanía en las actividades deportivas de la región, consolidando al IMCUFIDE como un referente de innovación en la gestión pública.

1.2 Objetivos Específicos

Para garantizar el éxito del objetivo general y asegurar un desarrollo estructurado, se han desglosado las siguientes metas específicas, medibles y técnicas:

1. Modelado de la Lógica de Negocio: Realizar un análisis exhaustivo y detallado de los procesos de gestión deportiva del IMCUFIDE, abarcando disciplinas como fútbol, básquetbol, voleibol, entre otras, y sus respectivas categorías formativas y competitivas. A partir de este análisis, diseñar un modelo de datos relacional (PostgreSQL) altamente normalizado que represente fielmente la interacción entre entidades complejas como deportes, categorías, equipos, jugadores, sedes, partidos y eventos, asegurando la integridad referencial y la consistencia de los datos en todo momento.
2. Desarrollo del Servicio de Backend (API): Implementar una API RESTful de alto rendimiento y baja latencia utilizando el lenguaje de programación Python y el framework moderno FastAPI. Este desarrollo debe adherirse estrictamente a una arquitectura limpia, separando claramente las responsabilidades entre los enrutadores (routers), la lógica de acceso a datos (CRUD) y los

modelos de datos (schemas Pydantic). Se deben construir endpoints seguros para la gestión interna (administración) y endpoints públicos optimizados para el consumo masivo de datos por parte de la ciudadanía.

3. Construcción de la Interfaz de Usuario (Frontend): Maquetar y programar una interfaz de usuario (UI) moderna, intuitiva y 100% responsiva utilizando el framework progresivo Vue.js y la herramienta de construcción de próxima generación Vite. El frontend debe adaptarse fluidamente a dispositivos móviles, tabletas y computadoras de escritorio, y debe consumir los datos de la API de forma asíncrona para ofrecer una experiencia de usuario dinámica, rápida y reactiva, minimizando los tiempos de espera.
4. Implementación de Infraestructura Containerizada: Containerizar cada uno de los servicios componentes del sistema (Frontend Nginx, Backend FastAPI, Base de Datos PostgreSQL) en imágenes de Docker independientes y optimizadas. Orquestar el despliegue y la comunicación segura entre estos servicios mediante `docker-compose`, garantizando la paridad absoluta de entornos entre desarrollo local y producción, facilitando así la colaboración del equipo y la escalabilidad horizontal del sistema.

2 Introducción

2.1 Claridad de las Técnicas y Contexto del Problema

El Instituto Municipal de Cultura Física y Deporte (IMCUFIDE) de Tenango del Valle, en su rol de organismo rector del deporte municipal, se enfrentaba a una problemática crítica y limitante: la fragmentación y dispersión de sus canales de comunicación y gestión. Históricamente, la administración de las ligas deportivas, la publicación de calendarios oficiales, la difusión de resultados y los procesos de inscripción dependían casi exclusivamente de métodos manuales, hojas de cálculo desconectadas y canales de comunicación no centralizados, principalmente redes sociales (Facebook, WhatsApp) y comunicados físicos en las sedes.

Esta situación operativa generaba tres desafíos estructurales graves que este proyecto busca resolver de raíz:

1. Ineficiencia Operativa y Carga Administrativa: Los administradores del instituto invertían una cantidad desproporcionada de tiempo en tareas manuales y repetitivas, como actualizar la misma información en múltiples plataformas, responder consultas individuales por mensajería y corregir errores de comunicación, restando tiempo valioso para la planificación estratégica del deporte.
2. Experiencia de Usuario Deficiente y Fragmentada: Para la ciudadanía (atletas, entrenadores, padres de familia y prensa local), encontrar información oficial consolidada, confiable y actualizada era una tarea compleja y frustrante. La información se perdía en el «timeline» de las redes sociales, generando confusión sobre horarios y sedes.
3. Ausencia de Identidad Digital Institucional: La falta de una plataforma web oficial, propia y profesional impedía que el instituto proyectara una imagen de seriedad, modernidad y transparencia acorde con la importancia y el alcance de sus programas deportivos municipales.

El proyecto aborda esta problemática mediante la implementación de un ecosistema web integral y tecnológico. Las técnicas empleadas trascienden la creación de un sitio web estático convencional; se ha diseñado y construido una plataforma dinámica desacoplada, donde la interfaz visual está completamente separada de la lógica de negocio y el almacenamiento de datos, permitiendo una evolución independiente, segura y robusta de cada componente del sistema.

2.2 Abordaje de la Problemática y Enfoque Arquitectónico

Para resolver los desafíos planteados de manera definitiva, se adoptó una arquitectura de software de tres capas desacoplada. Esta decisión arquitectónica es fundamental y se justifica técnicamente por su capacidad para garantizar la mantenibilidad, escalabilidad, testabilidad y seguridad del sistema a largo plazo.

El sistema se divide en tres componentes autónomos que interactúan a través de protocolos estándar de la industria (HTTP/JSON):

- Frontend (El Cliente): Una Single Page Application (SPA) construida con las tecnologías de vanguardia Vite y Vue.js. Es la capa de presentación con la que interactúa el usuario final. Su responsabilidad exclusiva es renderizar la interfaz gráfica, gestionar el estado de la aplicación en el navegador y realizar peticiones asíncronas al servidor para obtener o enviar datos, ofreciendo una experiencia fluida similar a una aplicación de escritorio.
- Backend (La API del Servidor): Un servidor de aplicaciones robusto desarrollado en Python con el framework FastAPI. Actúa como el núcleo lógico y el «cerebro» del sistema. Recibe las peticiones del frontend, valida rigurosamente los datos entrantes, aplica las reglas de negocio (ej. validación de roles de usuario, filtrado de partidos por fecha, cálculo de estadísticas) y coordina las operaciones de lectura y escritura con la base de datos.
- Base de Datos (El Almacén): Un sistema de gestión de bases de datos relacional (RDBMS) PostgreSQL, alojado en la nube. Es la capa de persistencia donde residen todos los datos de forma estructurada, segura y permanente, garantizando la integridad y disponibilidad de la información crítica del instituto.

Este enfoque de desacoplamiento ofrece ventajas significativas sobre los sistemas monolíticos tradicionales (como CMS estándar). Permite que los equipos de desarrollo trabajen en paralelo en distintas capas, facilita la actualización tecnológica de cada componente sin afectar a los demás (por ejemplo, rediseñar completamente el frontend sin tocar una sola línea de código del backend) y permite escalar los recursos de infraestructura de forma independiente según la demanda específica de cada capa.

2.3 Justificación de la Infraestructura: El Ecosistema Docker

Una de las decisiones técnicas más trascendentales y estratégicas del proyecto fue fundamentar toda la infraestructura de desarrollo y despliegue en la containerización mediante Docker. En lugar de depender de instalaciones directas y frágiles en el sistema operativo del servidor o de

las máquinas de los desarrolladores, cada componente del sistema se empaqueta en su propio contenedor aislado, ligero y portátil.

La justificación técnica para el uso de Docker en este proyecto se basa en tres pilares fundamentales:

1. Consistencia y Paridad de Entornos: Docker elimina por completo el clásico y costoso problema de «en mi máquina funcionaba». Al definir la infraestructura como código mediante archivos `Dockerfile` y `docker-compose.yml`, garantizamos que el entorno de desarrollo local (en Ubuntu o Windows) sea una réplica binaria exacta del entorno de producción. Las versiones de las librerías, la configuración del sistema operativo base y las dependencias son idénticas en ambos escenarios, reduciendo drásticamente los errores en producción.
2. Portabilidad y Flexibilidad de Despliegue: La aplicación completa, orquestada mediante `docker-compose`, se vuelve agnóstica a la infraestructura subyacente. Puede ser desplegada con el mismo comando simple en un servidor local, en una máquina virtual en la nube (AWS, Azure, DigitalOcean) o en un clúster de orquestación como Kubernetes. Esto otorga una enorme flexibilidad para futuras migraciones o expansiones sin requerir reconfiguraciones complejas.
3. Aislamiento y Seguridad: Cada servicio (API, Base de Datos, Servidor Web) se ejecuta en su propio espacio de usuario aislado dentro del kernel del sistema operativo. Si un servicio falla o es comprometido, el impacto se contiene dentro de ese contenedor específico, protegiendo la integridad y estabilidad del resto del sistema. Además, esto evita conflictos entre dependencias de diferentes servicios (por ejemplo, si el backend requiere una versión específica de Python que es incompatible con otras herramientas del sistema).

2.4 Justificación de las Tecnologías Específicas

La selección del «stack» tecnológico no fue una decisión arbitraria, sino el resultado de un análisis profundo de los requerimientos del proyecto, las características del equipo y las fortalezas técnicas de cada herramienta disponible en el mercado actual:

2.4.1 Frontend: Vue.js + Vite

Se eligió Vue.js por su arquitectura progresiva y su sistema de componentes reactivos altamente eficiente. Esto permitió modularizar la interfaz de usuario, creando componentes reutilizables (como tarjetas de equipos, tablas de calendarios, headers de navegación, footers) que facilitan el mantenimiento y aseguran la consistencia visual en todo el sitio. Vite se seleccionó como la herramienta de construcción (bundler) por su rendimiento superior en tiempo de desarrollo, gracias

al uso de módulos ES nativos (ESM), lo que acelera drásticamente el ciclo de retroalimentación durante la programación y produce archivos finales altamente optimizados para producción.

2.4.2 Backend: Python + FastAPI

Se optó por el lenguaje Python debido a su legibilidad, robustez y la madurez de su ecosistema. FastAPI fue seleccionado sobre otros frameworks tradicionales (como Django o Flask) por tres razones técnicas decisivas para este proyecto:

- **Rendimiento Asíncrono:** FastAPI está construido sobre los estándares modernos ASGI (Starlette) y utiliza Pydantic, ofreciendo un rendimiento de ejecución comparable a Node.js y Go, lo cual es vital para una API que debe servir datos en tiempo real a múltiples usuarios concurrentes sin bloquearse.
- **Validación de Datos Automática:** La integración nativa con Pydantic asegura que todos los datos que entran y salen de la API sean validados automáticamente contra esquemas estrictos definidos en el código, reduciendo drásticamente los errores en tiempo de ejecución y mejorando la seguridad.
- **Documentación Interactiva Automática:** FastAPI genera automáticamente documentación interactiva (Swagger UI y ReDoc) basada en el código y los tipos de datos. Esto facilitó enormemente las pruebas manuales de los endpoints y la integración fluida con el equipo de desarrollo frontend.

2.4.3 Base de Datos: PostgreSQL

Dado que el dominio del problema (gestión de ligas deportivas) implica entidades con relaciones complejas, estrictas y jerárquicas (un partido **debe** tener dos equipos, un equipo **debe** pertenecer a una categoría, una categoría **debe** pertenecer a un deporte), un modelo de base de datos relacional era obligatorio. Se eligió PostgreSQL por ser el motor de base de datos relacional de código abierto más avanzado del mundo, ofreciendo soporte robusto para integridad referencial, transacciones complejas (ACID) y tipos de datos avanzados que garantizan la consistencia y fiabilidad de la información crítica del instituto a largo plazo.

3 Modelo de Datos

El diseño del modelo de datos es el cimiento sobre el que se construye toda la lógica de negocio y la funcionalidad de la plataforma. Se implementó un esquema relacional altamente normalizado en PostgreSQL, diseñado meticulosamente para ser eficiente, escalable, libre de redundancias y capaz de soportar la evolución futura de los requerimientos del IMCUFIDE.

3.1 Completitud y Corrección del Modelo

El modelo de datos implementado es completo y correcto, abarcando la totalidad de las necesidades operativas y de gestión del instituto. Se estructura en cuatro dominios principales de información:

1. Dominio de Administración y Seguridad (`usuarios`): Se diseñó una tabla `usuarios` dedicada exclusivamente a la autenticación y autorización de los administradores del sistema. Esta tabla almacena credenciales encriptadas (`hashed_password`) y roles de usuario (`rol`). Esta separación es una decisión de diseño crucial para la seguridad: los datos de acceso administrativo nunca se mezclan con los datos públicos de los deportistas o equipos, reduciendo la superficie de ataque y facilitando la auditoría.
2. Dominio Jerárquico Deportivo (`deportes`, `categorias`, `equipos`): Se estableció una jerarquía relacional clara para organizar la competición y facilitar la navegación:
   - **`deportes`:** Entidad raíz (Fútbol, Básquetbol, Voleibol).
   - **`categorias`:** Subdivisiones competitivas (Juvenil A, Infantil, Veteranos). Relación 1:N con Deportes.
   - **`equipos`:** La unidad base de la competencia. Relación 1:N con Categorías.

   Esta estructura normalizada permite consultas eficientes y flexibles, facilitando el filtrado de información en el frontend (ej. «Mostrar solo equipos de Fútbol»).
3. Dominio de Participantes (`jugadores`, `documentos`): La tabla `jugadores` contiene la información biográfica y deportiva de los atletas. Se implementó una decisión de diseño clave al desacoplar los archivos legales y administrativos en una tabla `documentos` separada (relación 1:N). Esto optimiza el rendimiento de la tabla principal de jugadores, manteniéndola ligera para consultas frecuentes, mientras permite una gestión flexible y escalable de múltiples documentos por atleta sin alterar la estructura principal.
4. Dominio de Competición (`sedes`, `partidos`, `eventos_partido`):

- **`sedes`:** Normaliza las ubicaciones físicas de los encuentros, evitando la repetición de datos y errores tipográficos en las direcciones.
- **`partidos`:** Es la entidad central de la operación diaria. Su diseño garantiza la integridad referencial estricta mediante múltiples claves foráneas (`equipo_local_id`, `equipo_visitante_id`, `sede_id`), impidiendo la creación de partidos con datos inconsistentes o referencias a equipos inexistentes.
- **`eventos_partido`:** Permite un registro granular y detallado de lo ocurrido en el juego (goles, faltas, tarjetas), vinculado a un minuto específico y a un jugador. Esto habilita la generación futura de estadísticas avanzadas y actas digitales.

3.2 Ejecución del Modelo

La «ejecución» operativa del modelo se verifica y valida a través de la funcionalidad de la API RESTful. Las pruebas exhaustivas realizadas confirman que el modelo responde correctamente a las operaciones transaccionales y de consulta:

- **Consultas Complejas y Optimizadas:** Rutinas del backend como `obtener_partidos_con_nombres` ejecutan `JOINs` eficientes entre cuatro tablas simultáneamente (`partidos`, `equipos` (x2), `sedes`) para entregar una vista unificada y legible al usuario final en una sola petición.
- **Integridad de Datos:** Los intentos de insertar datos inválidos o incompletos son rechazados automáticamente por las restricciones (constraints) definidas en la base de datos, validando la robustez y confiabilidad del diseño.
- **Consumo Real y Dinámico:** La interfaz de usuario consume y renderiza estos datos en tiempo real. El calendario de partidos y las listas de equipos se generan dinámicamente a partir de los registros de la base de datos, cerrando el ciclo completo de ejecución del modelo.

3.3 Cambios Estratégicos en el Modelo para Alcanzar el Objetivo

Durante el desarrollo del proyecto, se realizaron ajustes críticos y estratégicos al modelo original para optimizar el rendimiento, la viabilidad técnica y la escalabilidad:

- **Externalización de Archivos Binarios:** Se descartó categóricamente almacenar imágenes y documentos (BLOBs) directamente dentro de la base de datos PostgreSQL. En su lugar, el modelo almacena únicamente cadenas de texto (URLs) que apuntan a un servicio de almacenamiento de objetos optimizado (Supabase Storage). Esta decisión reduce drásticamente el tamaño de la base

de datos, facilita las copias de seguridad y mejora significativamente los tiempos de respuesta de la API.

- **Desacoplamiento de Identidades:** Se implementó una separación explícita entre `usuarios` (actores del sistema con capacidad de login) y `jugadores` (entidades de datos pasivas). Esto evita ambigüedades de seguridad y permite una gestión de permisos mucho más granular y segura.
- **Esquemas de Respuesta Optimizados (DTOs):** Se crearon esquemas Pydantic específicos (como `PartidoConNombres` y `EquipoConPlantilla`) en la capa de aplicación. Esto permite que el backend realice el trabajo pesado de formatear, filtrar y estructurar los datos, entregando al frontend objetos JSON listos para ser consumidos, minimizando la lógica de procesamiento necesaria en el navegador del cliente y ahorrando ancho de banda.

4 Discusión de Resultados

4.1 Presentación de Resultados Establecidos y Alcanzados

El proyecto ha culminado con la implementación exitosa y funcional de todos los componentes técnicos y operativos planificados en la fase de diseño:

- Se ha desplegado un modelo de datos PostgreSQL robusto y operativo en la nube (Supabase), capaz de soportar la integridad referencial de todo el sistema deportivo.
- Se ha desarrollado y puesto en producción una API RESTful (Backend) en Render, sirviendo datos públicos y privados de manera segura y eficiente.
- Se ha construido y publicado una aplicación web (Frontend) en Vercel, ofreciendo una experiencia de usuario moderna, reactiva y adaptada a dispositivos móviles.
- Se ha validado un entorno de desarrollo local totalmente containerizado con Docker, demostrando la portabilidad del sistema y facilitando la colaboración futura.

4.2 Interpretación, Principios y Discusión de Hallazgos

La aplicación estricta de principios de ingeniería de software modernos arrojó hallazgos significativos durante el desarrollo:

- Principio de Separación de Intereses (SoC): La separación física y lógica del proyecto en tres componentes (Frontend, Backend, BD) demostró ser invaluable. Al mantener el frontend completamente agnóstico respecto a la lógica de la base de datos y viceversa, se simplificó drásticamente la depuración de errores, permitiendo identificar fallos de comunicación (como CORS) de manera aislada y precisa.
- Enfoque API-First: Diseñar primero los contratos de la API aseguró que la integración entre el frontend y el backend fuera fluida. Este enfoque facilitó la eficiencia en la transferencia de datos, permitiendo al backend realizar el procesamiento pesado y optimizar las respuestas antes de enviarlas a la red.
- Infraestructura como Código (IaC): El uso de Docker transformó la configuración del entorno de una tarea manual y propensa a errores a un proceso automatizado y reproducible. Esto confirmó que la inversión inicial en la configuración de contenedores se recupera con creces en estabilidad, consistencia y velocidad de despliegue.

4.3 Discusión de Limitaciones y Ventajas del Enfoque

El análisis crítico de la solución implementada revela un balance positivo, aunque con áreas de mejora identificadas:

- **Ventaja Estratégica:** La mayor fortaleza es la escalabilidad independiente de cada capa y la alta mantenibilidad del código. La arquitectura permite, por ejemplo, desarrollar una aplicación móvil nativa en el futuro que consuma la misma API sin requerir cambios en el backend.
- **Ventaja Operativa:** La separación de tecnologías permite especialización en el equipo de desarrollo y actualizaciones modulares.
- **Limitación Técnica (Inicial):** La complejidad inicial de despliegue y configuración de la comunicación entre servicios distribuidos (problemas de CORS, variables de entorno) fue mayor que en una arquitectura monolítica, aunque esto se compensa con la robustez final.
- **Limitación de Infraestructura Gratuita:** Se identificó una latencia perceptible en el primer arranque (**cold start**) de los servicios backend alojados en capas gratuitas. Esta es una limitación puramente infraestructural y financiera, no de diseño de software, y se mitiga completamente mediante la actualización a planes de servicio estándar en un entorno de producción real.

5 Conclusión

El proyecto IMCUFIDE representa un caso de éxito demostrable en la modernización digital de la gestión deportiva pública municipal. Se ha logrado diseñar, desarrollar e implementar una plataforma tecnológica que no solo cumple con los requerimientos funcionales inmediatos de información y gestión, sino que está arquitectónicamente preparada para el crecimiento y la evolución futura.

La decisión estratégica de utilizar una arquitectura basada en contenedores Docker y microservicios desacoplados, aunque introdujo una curva de aprendizaje y complejidad inicial mayor en comparación con soluciones monolíticas tradicionales, ha resultado en un sistema superior en términos de mantenibilidad, escalabilidad, seguridad y robustez. La plataforma entregada es ahora un activo digital valioso, capaz de evolucionar para integrar nuevas ligas, aplicaciones móviles y módulos administrativos avanzados sin requerir una reingeniería costosa del sistema base.

6 Repositorio y Acceso

6.1 Repositorio del Proyecto en GitHub

El código fuente completo del proyecto, incluyendo la configuración de infraestructura (Docker), el código del Backend y el código del Frontend, se encuentra alojado y disponible públicamente en el siguiente repositorio:

https://github.com/WiliEsquivel/imcufide-proyecto-main.git

6.2 Instrucciones de Despliegue y Ejecución Local (Docker)

Para replicar el entorno de producción en una máquina local utilizando la infraestructura contai-nerizada, se deben seguir los siguientes pasos en una terminal:

```
# 1. Clonar el repositorio desde GitHub
git clone ("https://github.com/WiliEsquivel/imcufide-proyecto-main.git")[https://
github.com/WiliEsquivel/imcufide-proyecto-main.git]

# 2. Acceder al directorio principal del proyecto
cd imcufide-proyecto

# 3. Iniciar todos los servicios orquestados con Docker Compose
# (Esto levantará la BD, el Backend y el Frontend automáticamente)
docker-compose up --build
```

6.3 Acceso a la Plataforma en Producción

La aplicación se encuentra desplegada en un entorno de producción real y es accesible públicamente para su revisión a través de la siguiente URL segura:

https://imcufide-proyecto.vercel.app/

7 Referencias

1. Martin, R. C. (2018). **Arquitectura Limpia: Guía del artesano para la estructura y el diseño de software**. Anaya Multimedia.
2. Grubor, S. (2017). **Deployment with Docker**. Packt Publishing.
3. Turnbull, J. (2014). **The Docker book: Containerization is the new virtualization**. James Turnbull.
4. Tiangolo. (2024). **FastAPI Documentation**. Recuperado de https://fastapi.tiangolo.com/
5. Vue.js. (2024). **Vue.js Documentation**. Recuperado de https://vuejs.org/guide/introduction.html
6. PostgreSQL Global Development Group. (2024). **PostgreSQL 16.1 Documentation**. Recuperado de https://www.postgresql.org/docs/
7. Garcia, L., & Martinez, C. (2021). Principios de diseño de experiencia de usuario (UX) para la mejora de servicios digitales en el sector público. **Revista Iberoamericana de Sistemas de Información y Tecnología**, 38(4), 115-130.
8. Mozilla. (n.d.). **CSS Grid Layout**. MDN Web Docs. Recuperado el 22 de septiembre de 2025.