Revista Innovación y Software
Vol. 6, No. 2, Mes Septiembre - Febrero, 2025
ISSN: 2708-0935
Pág. 40-57
https://revistas.ulasalle.edu.pe/innosoft

Tipo de artículo: Artículos originales
Temática: Tecnologías de la información y las comunicaciones
Recibido: 22/10/2024 | Aceptado: 29/11/2024 | Publicado: 30/09/2025

Identificadores persistentes:
DOI: 10.48168/innosoft.s24.a299
ARK: 42411/s24.a299
PURL: ark:/42411/s24.a299

Diseño de una arquitectura de microservicios en contenedores virtuales
Design of a microservices architecture in virtual containers

Samantha Yazmin Elizalde Valencia[1][0009-0001-8412-3176]*, **José Juan Hernández Mora**[2][0000-0003-2878-7290], **María Guadalupe Medina Barrera**[3][0000-0003-3074-0029], **Juan Ramos Ramos**[4][0009-0004-7440-7257]

[1]Instituto Tecnológico de Apizaco. Dirección postal. m23370041@apizaco.tecnm.mx
[2]Instituto Tecnológico de Apizaco. Dirección postal. juan.hm@apizaco.tecnm.com
[3]Instituto Tecnológico de Apizaco. Dirección postal. guadalupe.mb@apizaco.tecnm.com
[4]Instituto Tecnológico de Apizaco. Dirección postal. juan.rr@apizaco.tecnm.com

*Autor para correspondencia: m23370041@apizaco.tecnm.mx

Resumen
En este artículo se presenta la propuesta metodológica para el diseño e implementación de una arquitectura de microservicios en contenedores. La propuesta abarca desde las metodologías utilizadas para la fase de investigación de las primeras fases de desarrollo del proyecto hasta la parte de la metodología utilizada para la creación de un sistema informático de prueba. Se propone una arquitectura basada en arquitecturas de dominio, así como también se expone el diseño de la arquitectura física del sistema, basado en los requerimientos expuestos por la institución colaboradora para el caso de prueba. Como parte de los resultados, se describe la configuración de los microservicios y contenedores, así como su integración en una red común de servicios en ejecución.

Palabras claves: Arquitectura, Contenerización, Docker, Microservicios

Abstract
This article presents the methodological proposal for the design and implementation of a containerized microservices architecture. The proposal covers the methodologies used for the research phase of the initial stages of project development, as well as the methodology used to create a test computing system. An architecture based on domain architectures is proposed, as well as the design of the system's physical architecture, based on the requirements presented by the collaborating institution for the test case. The results describe the configuration of the microservices and containers, as well as their integration into a common network of running services.

Keywords: *Architecture, Containerization, Docker, Microservices*

Introducción

En la actualidad, incontables organizaciones siguen atrapadas con software construido sobre bases obsolestas y poco robustas, la magnitud de este problema es bastante amplia pues resulta que aproximadamente el 65 % de aplicaciones empresariales son sistemas legacy. Lo más frustrante es ver cómo estas organizaciones desperdician entre el 60 % y 80 % de sus presupuestos de TI simplemente manteniendo estos sistemas totalmente obsoletos, en lugar de invertir en innovaciones que realmente podrían transformar sus negocios [1].

Estos sistemas legacy funcionan, pero con muchas dificultades, creados a partir de arquitecturas monolíticas difícil de sobre llevar, documentación pobre o inexistente, y dependencia de tecnologías que muchas veces ya se encuentran obsoletas y el intentar integrar con plataformas modernas resulta una tarea aun mas compleja [2].

Las arquitecturas de microservicios emergen en este panorama como una alternativa alentadora. Estudios sugieren que implementar microservicios en contenedores puede mejorar el rendimiento hasta en un 15 %, gracias a que cada componente ocupa su propio espacio y se despliega casi instantáneamente [3]. Este artículo muestra las metodologías propuestas para diseñar arquitecturas basadas en microservicios usando contenedores Docker.

Metodologías propuestas para el desarrollo del proyecto

En esta sección se presentan las metodologías sugeridas para el desarrollo del proyecto, desde la parte del estudio y análisis de la información, hasta la parte donde detalla el proceso de como se genera el desarrollo del sistema de software.

Metodología de investigación

Después de analizar algunas de las metodologias de invetigación se opto por dos enfoques que cumplian con ciertas caracteristicas importantes para llevar acabo el proyecto: la investigación tecnológica y la investigación tecnológica en ingeniería. Estas metodologias resultan adecuados para proyectos donde el objetivo principal no es establecer nuevas teorías sino la de reconstruir procesos mediante la adaptación y mejora de soluciones existentes.

La investigación tecnológica permitira el examinar soluciones existentes, tomar sus mejores elementos y adaptarlos a las necesidades del proyecto [4]. La investigación tecnológica en ingeniería añade un componente de suma importancia: el proyecto de intervención [5]. Como se puede ver en la figura 1, este componente va más allá del análisis teórico, requiriendo la implementación de una solución que pueda ser evaluada. En este caso, esto se traduce en el desarrollo de un sistema funcional en base a la arquitectura propuesta. Esto permitirá

verificar si se cumple con los requisitos planteados en la pregunta de investigación. Además, incorpora un proceso iterativo que permite retroalimentación constante durante las diferentes etapas del proyecto.

Figura 1. Metodología de investigación

A continuación, se proponen las siguientes etapas para realizar la investigación:

1. **Identificación del problema:** Esta etapa se centra en identificar y formular el problema a investigar, en esta parte es donde se establecen los objetivos específicos y se encarga de delimitar el alcance de la investigación.
2. **Revisión bibliográfica:** Realizar una revisión de la literatura vinculada con el tema de investigación, identificando antecedentes y metodologías utilizadas en otros trabajos de investigación.

3. **Análisis de la información:** En esta fase se elaboran algunas soluciones al problema o se plantean nuevas dudas sobre el tema, utilizando la información recopilada en la fase anterior.

4. **Elaboración del proyecto de intervención:** En esta parte se lleva a cabo el desarrollo e implementación del sistema utilizando una metodología de desarrollo de software.

5. **Validación y verificación:** Esta etapa se realiza mediante la aplicación de casos de prueba.

6. **Resultados:** Interpretar los resultados obtenidos en base a los objetivos de la investigación, así como señalando posibles áreas de mejora.

7. **Conclusiones:** Elaborar conclusiones fundamentadas en los hallazgos obtenidos, además de sugerir recomendaciones para futuras investigaciones.

Metodología de desarrollo del sistema informático

Para la gestión del proyecto, se propone una metodología ágil que fusiona elementos de Scrum y Kanban. Como se muestra en la figura 2, Scrum proporciona la estructura necesaria esto a base de sprints con entregables y revisiones periódicas que permiten ajustar el rumbo del proyecto [6]. Kanban proporciona un esquema visual a través de sus tableros lo que permite mejorar la planificación de los sprints [7].

Dado que el equipo de desarrollo está conformado por un grupo reducido (tres integrantes), se implementa la variante Small-Scale Scrum [8], adecuada por promover la auto organización, la comunicación constante y responsabilidades compartidas en equipos compactos. Esta combinación metodológica permite tener una mejor planificación de las iteraciones con retroalimentación continua. La metodología se segmentará en Sprints con el objetivo de mantener un ritmo de entregas progresivas y adaptarse a las demandas del cliente.

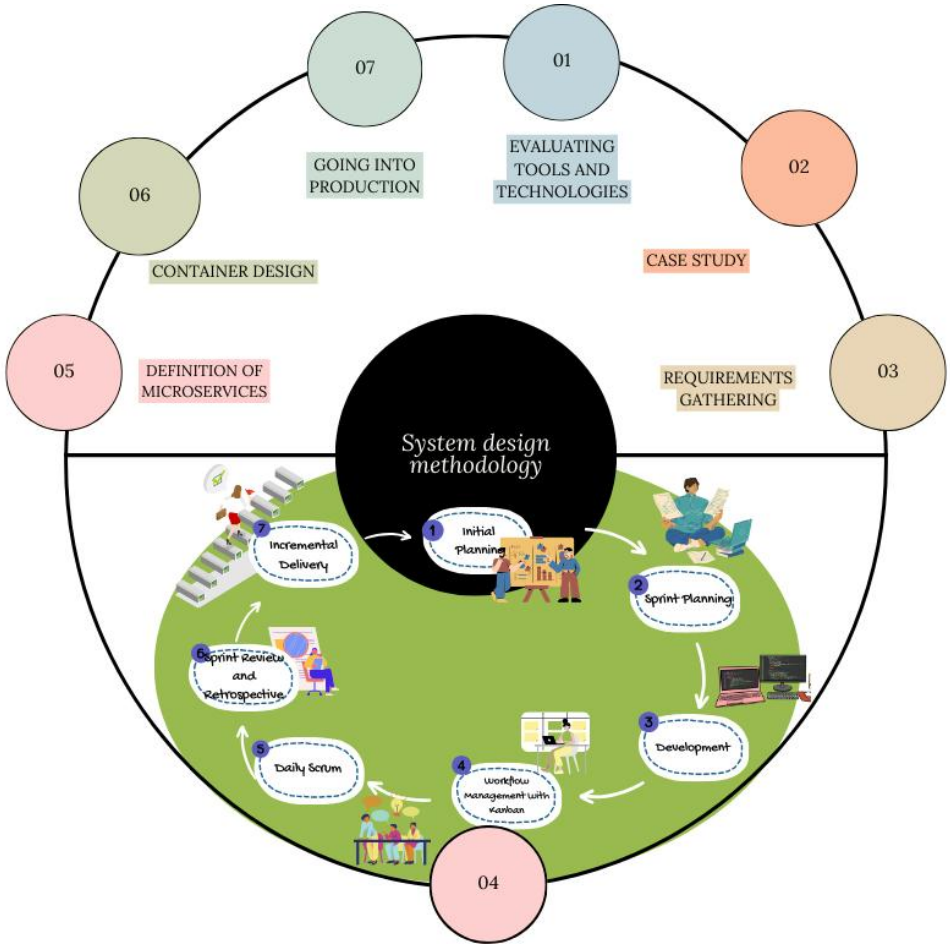

Figura 2. Metodología de desarrollo del sistema

1. **Evaluación de herramientas y tecnologías:** Comparar herramientas de contenedores y APIs.
2. **Estudio de casos:** Realizar el análisis de casos de estudio con atributos parecidos para identificar prácticas sugeridas y retos.
3. **Recolección de requisitos:** En esta etapa, se llevarán a cabo entrevistas con interesados para identificar las necesidades del sistema.
4. **Planificación de sprints:**
   - **Planificación Inicial.** En la siguiente etapa se busca establecer el ámbito general del proyecto, los objetivos, los roles y el backlog inicial.

- **Planificación de Sprint.** Se organiza el trabajo que se llevará a cabo durante el sprint.
- **Gestión del Flujo de Trabajo utilizando Kanban.** En esta etapa se garantiza un flujo constante de trabajo y eliminar cuellos de botella.
- **Daily Scrum.** El propósito de esta etapa es supervisar el avance y solucionar bloqueos de forma rápida.
- **Revisión y Retrospectiva de Sprint.** Se muestra al Product Owner el avance funcional y se recibe retroalimentación.
- **Entrega Incremental.** Al concluir cada sprint, entregar versiones del producto.
- **Repetición del Ciclo.** Explorar e incorporar mejoras o nuevas características a través de iteraciones continuas.

5. **Definición de microservicios.** En este segmento se seleccionarán los microservicios que se desempeñarán en el sistema.
6. **Diseño de Contenedores:** Elaborar la estructura que incorporarán cada contenedor.
7. **Puesta en marcha en producción:** Implementar la arquitectura en el entorno de producción.

Arquitectura del sistema

Esta sección expone el diseño arquitectónico sugerido para el sistema, detallando tanto su estructura lógica en capas como su implementación física. La arquitectura establece la estructura modular del sistema, detallando las distintas capas funcionales y sus interrelaciones, mientras que la arquitectura física ilustra la disposición precisa de los elementos de infraestructura, ajustada específicamente a las necesidades técnicas y operativas de la institución colaboradora.

Arquitectura del sistema

La arquitectura de software mostrada se basa en principios de diseño orientado al dominio, fusionados con una perspectiva de microservicios (ver figura 3). El eje de esta perspectiva se basa en un entendimiento detallado del ámbito empresarial, lo que demanda una colaboración cercana entre desarrolladores y expertos del área [9]. Esta estructura diferencia la lógica central (núcleo) de los componetes tecnológicos por medio de capas, mientras incorpora microservicios para elementos funcionales especificos. Lo que permite simplificar significativamente el mantenimiento y la evolución del sistema [10]. Esta arquitectura proporciona un entorno de trabajo flexible, escalable y modular, creado específicamente para respaldar la evolución constante de sistemas complejos y su

posible traslado a ambientes cloud. La solución implementa una estructura por capas independientes que aísla las reglas de negocio centrales de las dependencias externas (frameworks, bases de datos e interfaces) [11].

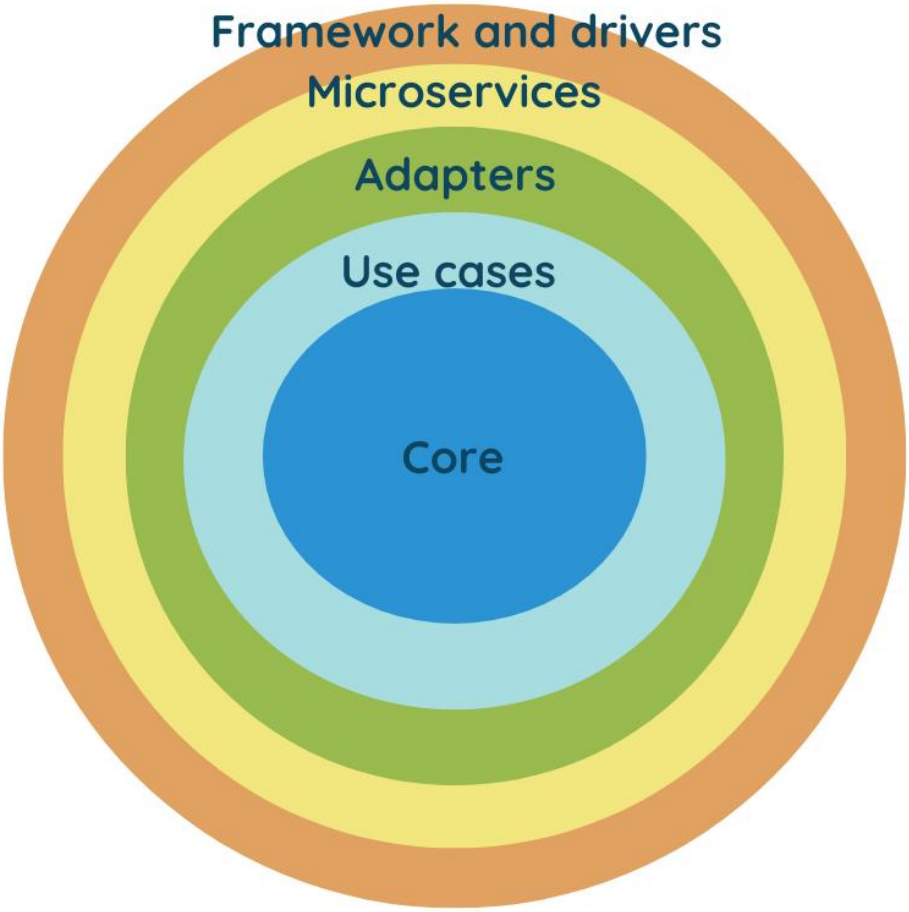

Figura 3. Metodología de desarrollo del sistema

- **Núcleo (Lógica de negocio):** El núcleo alberga las entidades y el pensamiento empresarial, basándose en el principio de la arquitectura de cebolla y limpia. Esto garantiza que la lógica empresarial esté separada y sea autónoma de la infraestructura.
- **Casos de uso:** Implementan casos de uso específicos y se comunican con el núcleo a través de interfaces. Esto está inspirado en la arquitectura hexagonal, en la que los servicios funcionan como mediadores.
- **Adaptadores (Interfaces):** Los puertos y adaptadores enlazan el núcleo y los servicios con sistemas

externos como bases de datos, APIs externas, entre otros. Esto garantiza que el núcleo se mantenga autónomo respecto a los detalles de implementación de la infraestructura.

- **Microservicios:** Cada característica del sistema se pone en marcha como un microservicio autónomo, desplegado en contenedores. Esto permite la escalabilidad y el despliegue constante, conforme a las prácticas óptimas de la arquitectura de microservicios.
- **Marcos y conductores:** Considerando la arquitectura limpia, esta capa está situada en el exterior, donde puede provocar un daño mínimo. Suele incluir marcos y herramientas como la base de datos y el marco web.

Vista física del Sistema

Para probar la metodología propuesta se desarrollara el sistema de gestión deportiva del Instituto del Deporte del Estado de Tlaxcala (IDET). El IDET, es una institución cuyo objetivo estratégico es transformar el deporte local mediante cuatro ejes principales: masificación, deporte social, deporte popular y deporte escolar [12].

Mediante el cual se identificaron tanto los requisitos funcionales como no funcionales orientados a: gestión de citas médicas deportivas, administración de historiales clínicos de atletas, generación de reportes estadísticos y de monitoreo, control completo del ciclo de consultas médicas. Este análisis posibilitó la creación de los siguientes artefactos esenciales: diagramas de casos de uso, casos de abuso, diagrama de clases para entidades médicas-deportivas, flujos de procesos clínicos y administrativos, especificaciones de comportamiento para módulos clave.

Los hallazgos de este estudio establecieron la base del diseño físico del sistema como se exhibe en la figura 4, garantizando que la arquitectura sugerida cumpla con las necesidades detectadas y los criterios de calidad establecidos.

Dentro de la arquitectura, los usuarios pueden acceder a la aplicación desde navegadores en computadoras, tabletas y teléfonos, estableciendo comunicación mediante solicitudes HTTP. A partir de estas peticiones, el frontend, desarrollado en React ofrece una interfaz visual. React gestiona los estilos del sistema a través de archivos CSS y utiliza JavaScript para mejorar la interactividad y la experiencia del usuario. Cuando un usuario interactúa con la interfaz, el frontend envía una solicitud a Django, que en este caso actúa como middleware a través de Django REST Framework. Este último factúa como un enlace entre la capa visual y los microservicios. En el backend, los modelos de datos juegan un rol crucial, debido a que representan la estructura de la información y su almacenamiento en la base de datos.

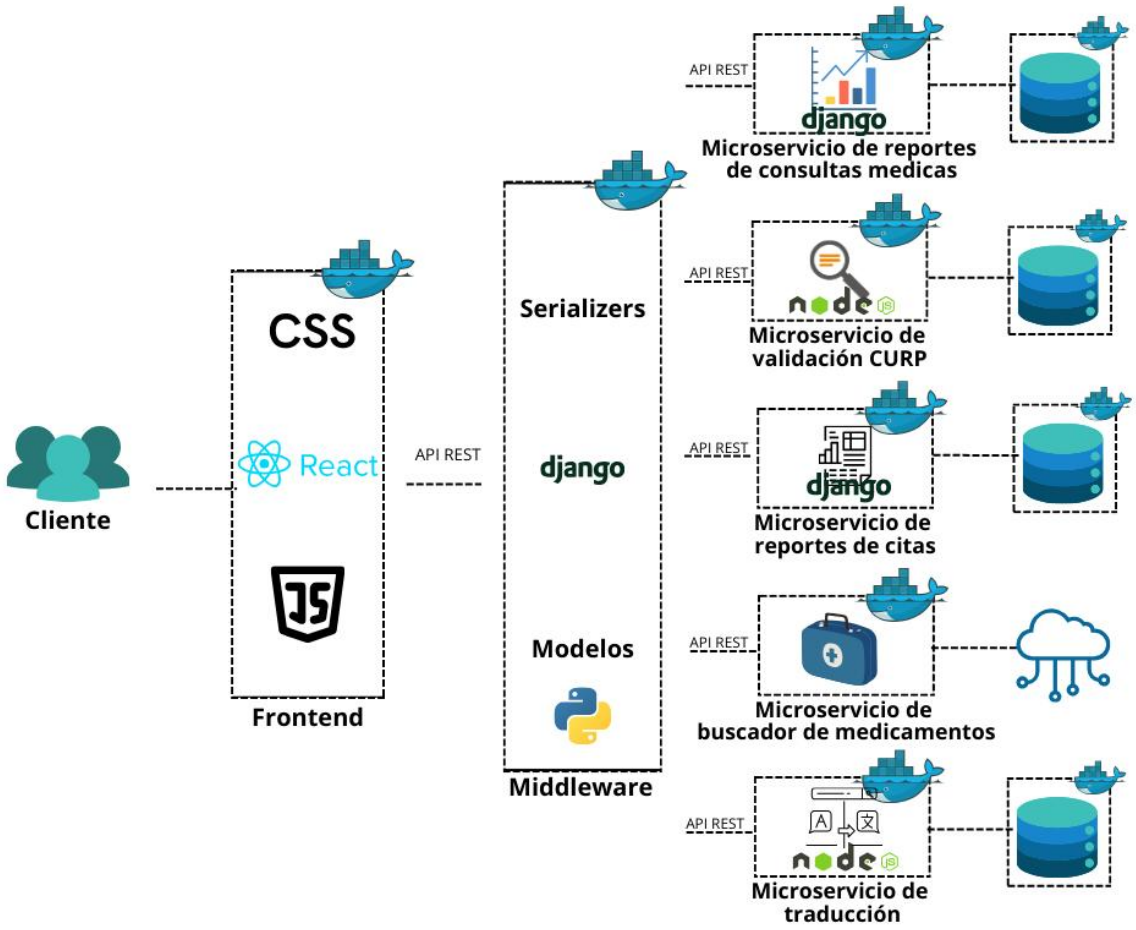

Figura 4. Vista física del sistema

El sistema cuenta con varios microservicios, creados para mejorar y verificar la información en los formularios. El sistema incorpora los siguientes microservicios:

- **Microservicio de Validación de Credenciales:** Validación de CURP de atletas mediante un servicio Node.js alojado en servidor la implementación estara basada en un repositorio Git clonado.
- **Microservicios de Analítica y Reportes:** Generación de diagramas estadísticos, elaboración de reportes personalizados en PDF con parámetros ajustables por el usuario y acceso seguro a la base de datos a través de ORM.
- **Microservicio de buscador de medicamentos:** El cual se pretende integrar a la parte de consultas

médicas, permite realizar la toma de decisiones más acertada para el profesional de la salud, en cuanto a medicamentos se refiere, y proporciona información sobre dosis recomendadas, reacciones adversas y otros datos relevantes. Este es un microservicio que pertenece al Departamento de Salud y Servicios Humanos de EE. UU., el cual se encuentra en la nube, y se accederá a través de una API. Además, requiere un servicio adicional para su funcionamiento.

- **Microservicio de traducción:** Ya que toda la información del buscador de medicamentos estaba disponible en inglés, se requiere la implementación de un traductor que permita mostrar la información en el idioma adecuado según el contexto de uso; este microservicio pertenece a lingva-translate. El cual permite la traducción tanto para hacer la petición como también para recibir la información, esto mediante un servicio Node.js.

Para asegurar la escalabilidad y portabilidad, cada elemento esencial del sistema está alojado en contenedores Docker, lo que permite su despliegue tanto en servidores locales como en la nube, dependiendo del requisito del cliente. La interacción entre diversos contenedores y microservicios se lleva a cabo mediante las API REST.

Resultados y discusión

En esta sección se expondrá parte de la implementación del proyecto de intervención. Se mostrará desde la configuración del frontend, backend y microservicios, así como también cómo se configuraron los microservicios dentro de los contenedores y cómo se comunican entre sí.

Configuración del frontend y backend

El sistema busca trabajar de forma modular, lo cual permite agregar nuevas funcionalidades sin comprometer demasiado el desempeño del sistema; es por esto que se necesita separar las funcionalidades de frontend y backend.

Para trabajar el frontend, se trabajó a través del framework de desarrollo React, alojado en el puerto 5173, el cual realiza todo lo relacionado a la parte visual del sistema. En cuanto al backend, se trabajó en base al framework de desarrollo Django, que permitió realizar toda la parte de los modelos de datos y manejar la parte de la lógica central de la aplicación. Este servicio se ejecuta en el puerto 8000. Para lograr la comunicación de ambas partes, se utilizo con la herramienta Django Rest Framework, que permitió la comunicación convirtiendo los modelos de datos a un formato JSON, esto a través de las vistas y archivos serializadores.

La siguiente figura muestra un ejemplo de la vista DeportesViewSet, la cual se trata de una vista basada en

ViewSet de Django REST Framework. La cual incluye las operaciones básicas de API REST.

Dentro de esta vista se define el conjunto de datos (Deporte), que será utilizado por la vista, lo que permite seleccionar todos los registros pertenecientes a dicho modelo. También, al hacer referencia al serializador (DeporteSerializer), indica que los datos del modelo (Deporte) deberán ser convertidos a JSON usando el serializador.

```
class DeporteViewSet(viewsets.ModelViewSet):
    queryset = Deporte.objects.all()
    serializer_class = DeporteSerializer
```

Figura 5. Vista basada en ViewSet

Mientras que en la figura 6, se muestra el archivo serializador perteneciente al modelo de datos de Deporte. Esta es la parte encargada de convertir los datos que se envían en formato JSON y también es el encargado de convertir los datos recibidos por la API. Lo cual permite crear, eliminar o actualizar registros. Dentro del serializador se utiliza la clase ModelSerializer, la cual permite la creación de serializadores a partir de un modelo de Django.

```
class DeporteSerializer(serializers.ModelSerializer):
    class Meta:
        model = Deporte
        fields = ['id', 'nombre']
```

Figura 6. Serializer basado en ModelSerializer

Ahora, en la parte del frontend, será necesario configurar una API cliente como se observa en la figura 7, que permita la interacción con el backend. Con base en una URL, permite la conexión con el servidor, lo que ayuda a obtener la información de los deportes. Y por lo tanto tener a las funciones CRUD, pero esta vez desde el frontend.

```
import axios from "axios";

const deportesApi = axios.create({
    baseURL: 'http://localhost:8000/Catalogos/Deportes/'
});

// Obtener todos los deportes
export const getAllDeportes = () => deportesApi.get('/');

// Crear nuevo deporte
export const createDeporte = (deporte) => deportesApi.post('/', deporte);

// Obtener un deporte por ID
export const getDeporteById = (id) => deportesApi.get(`/${id}/`);

// Actualizar un deporte
export const updateDeporte = (id, deporte) => deportesApi.put(`/${id}/`, deporte);

// Eliminar un deporte
export const deleteDeporte = (id) => deportesApi.delete(`/${id}/`);
```

Figura 7. API cliente

De esta manera, los datos de los modelos podrán ser gestionados por el backend a través de la API.

Configuración de los microservicios

En cuanto a los servicios creados por el equipo de desarrollo, la configuración es muy similar, ya que estos dos microservicios de generación de reportes y estadísticas de citas y consultas se encuentran en dos proyectos Django diferentes, alojados en los puertos 8001 y 8002. Para realizar la comunicación, se obtiene la información desde las URLs pertenecientes al microservicio backend (Figura 8). Lo que permite que estos microservicios procesen y filtren los datos relevantes para ser presentados en el frontend, ya sea mediante gráficos o para enriquecer la información contenida en los archivos PDF generados.

```python
class EstadisticasCitasView(APIView):
    def get(self, request):
        try:
            # URLs de los servicios
            API_CITAS = 'http://backend:8000/Modulos/Citas/'
            API_PROFESIONALES = 'http://backend:8000/Catalogos/Profesionales-Salud/'
            API_ATLETAS = 'http://backend:8000/Catalogos/Atletas/'
            API_AREAS = 'http://backend:8000/Catalogos/Areas/'

            # 1. Obtener todas las citas
            logger.info(f"Consultando citas en: {API_CITAS}")
            citas_response = requests.get(API_CITAS, timeout=10)
            citas_response.raise_for_status()
            todas_citas = citas_response.json()

            # 2. Filtrar citas del mes actual
            mes_actual = datetime.now().month
            año_actual = datetime.now().year
```

Figura 8. Vista de filtrado de microservicio de estadisticas

Los microservicios clonados de repositorios git corresponden al servicio de traducción y de validación de CURP, están desarrollados en Node.js y se encuentran alojados en los puertos 3000 y 3001. Para su ejecución fue necesario instalar los requerimientos especificados en su documentación. El manejo de su información se realiza mediante el consumo de sus respectivas APIs (Figura 9). Las APIs se encargarán de gestionar las peticiones necesarias para lograr la comunicación del frontend con alguno de los dos servicios externos.

```
import axios from "axios";
// Configuración para la API de Node.js
const nodeApi = axios.create({
  baseURL: 'http://localhost:3000',
  timeout: 5000
});
/**
 * Valida una CURP usando el servicio Node.js
 * @param {string} curp - La CURP a validar
 * @returns {Promise<{valid: boolean, data: {sexo: string, fecha_nacimiento: string}|null}>}
 */
export const validateCurp = async (curp) => {
  try {
    const response = await nodeApi.post('/curp/validate', { curp });

    if (!response.data) {
      throw new Error('Respuesta vacía del servidor');
    }

    return {
      valid: response.data.valid,
      data: {
        sexo: response.data.sex === 'Hombre' ? 'H' : 'M',
        fecha_nacimiento: response.data.birthDate
      }
    };
  } catch (error) {
```

Figura 9. API servicio de validación de CURP

En el caso del microservicio externo que se encuentra alojado en la nube, se define una función, la cual consulta la API de FDA para obtener la información de los medicamentos, se realiza la búsqueda de información y devuelve una lista de los objetos procesados.

```
// Función para buscar en OpenFDA
const buscarEnOpenFDA = async (terminoBusqueda) => {

  let url = `https://api.fda.gov/drug/label.json?search=openfda.brand_name:"${encodeURIComponent(terminoBusqueda)}"&limit=5`;

  let respuesta = await fetch(url);
  let datos = await respuesta.json();


  if (!datos.results || datos.results.length === 0) {
    url = `https://api.fda.gov/drug/label.json?search=openfda.brand_name:${encodeURIComponent(terminoBusqueda)}&limit=5`;
    respuesta = await fetch(url);
    datos = await respuesta.json();
  }
```

Figura 10. Función para acceder a OPEN FDA

Configuración de contenedores (Docker)

Para contenerizar los microservicios, fue necesario crear un archivo Dockerfile dentro de la carpeta correspondiente de cada uno, el cual permitió crear la imagen de cada microservicio, en el cual se especifica el entorno base sobre el que se construirá la imagen, directorio de trabajo, instalación de dependencias, exposición de dependencias y comando de arranque.

Por otro lado el archivo docker compose permitió administrar los múltiples contenedores donde se asignó una red en la cual se estarán comunicando, esta opción es especialmente útil para aplicaciones que cuentan con múltiples servicios.

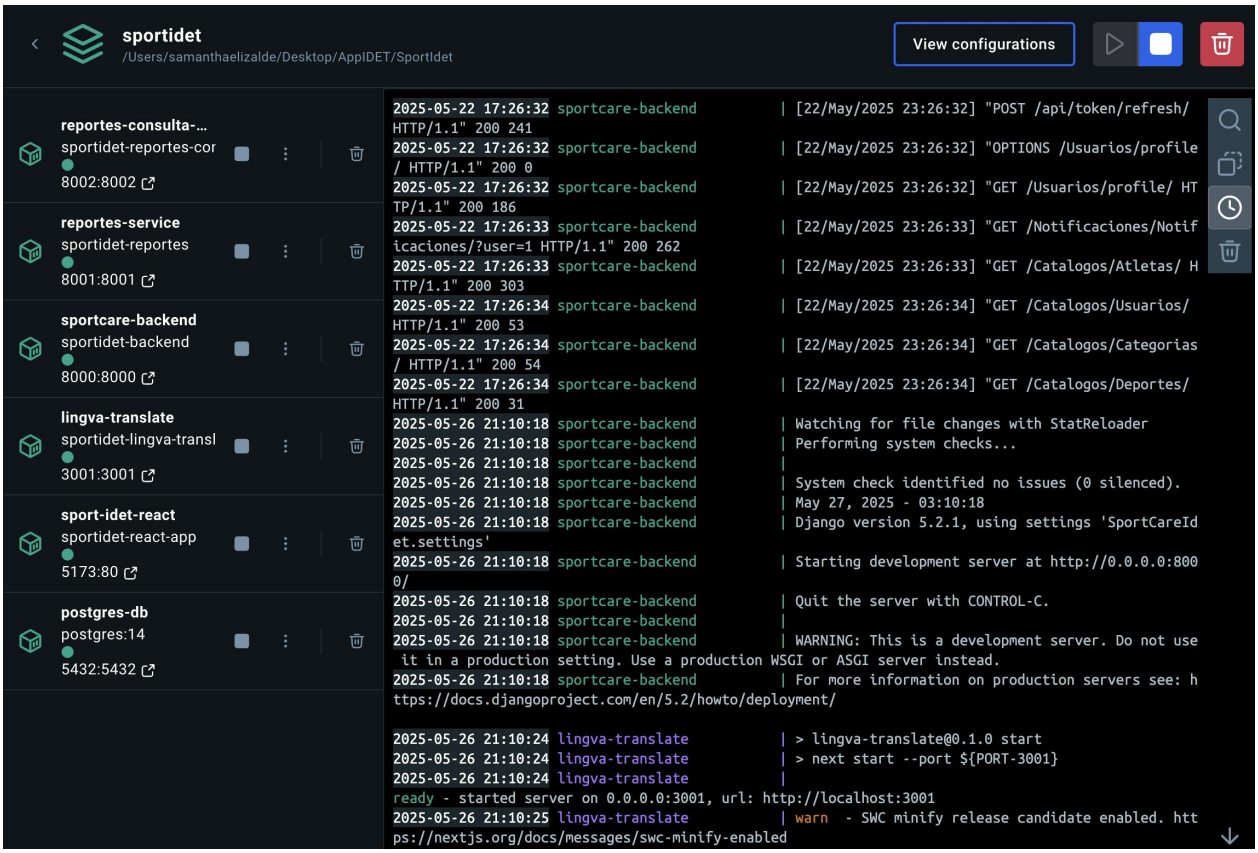

Figura 11. Microservicios en contenedores

Conclusiones

La propuesta de una arquitectura basada en microservicios en contenedores (Docker) demuestra ser un reemplazo para los sistemas legacy, ofreciendo escalabilidad, modularidad y portabilidad. Este artículo presentó la metodología que abarca desde el análisis hasta la producción, validando la arquitectura mediante un caso de estudio real en el Instituto del Deporte del Estado de Tlaxcala (IDET). Dentro de lo cual se pudo observar que, gracias a la contenerización, se simplifica el despliegue y gestión de microservicios, reduciendo conflictos de dependencias.

Mientras que la combinación de Django (backend) y React (frontend) permite una separación de responsabilidades, mientras que el usar Django REST Framework permitió facilitar la comunicación con servicios externos.

Contribución de Autoría

Samantha Yazmin Elizalde Valencia: Conceptualización, Investigación, Metodología, Software, Validación, Redacción - borrador original. José Juan Hernández Mora: Conceptualización, Investigación, Metodología, Análisis formal, Recursos, Visualización, Supervisión, Administración de proyectos, Curación de datos, Escritura, revisión y edición. María Guadalupe Medina Barrera: Conceptualización, Visualización, Análisis formal, Recursos, Supervisión, Administración de proyectos, Escritura, revisión y edición. Juan Ramos Ramos: Conceptualización, Visualización, Curación de datos, Análisis formal, Recursos, Supervisión, Administración de proyectos, Escritura, revisión y edición.

Referencias

[1] R. A., "The cost of legacy systems: How outdated it holds companies back," Mar. 21 2025. [Online]. Available: https://www.linkedin.com/pulse/cost-legacy-systems-how-outdated-holds-companies-back-andre-occec

[2] L. Sánchez, "Sistemas legacy - modernizando la infraestructura tecnológica," 2024. [Online]. Available: https://www.initiumsoft.com/blog_initium/sistemas-legacy/

[3] A. F. S. Chiza, "Análisis de rendimiento entre una arquitectura monolítica y una arquitectura de microservicios – tecnología basada en contenedores," Repositorio Core, Tech. Rep., 2018. [Online]. Available: https://core.ac.uk/download/pdf/200323828.pdf

[4] F. Bello, "Reflexión: La investigación tecnológica o cuando la solución es el problema," *Revista de la Facultad de Ingeniería, Universidad de Carabobo*. [Online]. Available: http://servicio.bc.uc.edu.ve/faces/revista/a6n13/6-13-3.pdf

[5] C. D. la Cruz Casaño, *Metodología de la investigación tecnológica en ingeniería*. Universidad Continental, 2016. [Online]. Available: https://www.academia.edu/94930372/Metodolog%C3%ADa_de_la_investigaci%C3%B3n_tecnol%C3%B3gica_en_ingenier%C3%ADa

[6] M. V. Estrada-Velasco, P. R. Saltos-Chávez, J. A. N. nez Villacis, and W. C. Cunuhay-Cuchipe, "Revisión sistemática de la metodología scrum para el desarrollo de software," *Revista Científica*, 2021. [Online]. Available: https://dialnet.unirioja.es/servlet/articulo?codigo=8384028

[7] A. Castro, "Scrum y kanban: Una combinación efectiva para la gestión ágil," Enero 13 2025. [Online]. Available: https://www.linkedin.com/pulse/scrum-y-kanban-una-combinaci%C3%B3n

[8] D. Garcia, "Scrum para equipos pequeños, una introducción," Feb. 19 2019. [Online]. Available: https://www.inteldig.com/2019/02/scrum-equipos-pequenos/

[9] G. E. O. Tibán and L. F. C. Sánchez, "Desarrollo de un prototipo de microservicios con clean architecture," 2023. [Online]. Available: https://dspace.ups.edu.ec/bitstream/123456789/28179/1/MSQ862.pdf

[10] J. D. L. Hinojosa, "Arquitectura de software basada en microservicios," 2017. [Online]. Available: https://1library.co/document/y81dmvwz-arquitectura-software-basada-microservicios-desarrollo-aplicaciones-asamblea-nacional.html

[11] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Prentice Hall, 2017.

[12] IDET, "Misión y visión," Instituto del Deporte de Tlaxcala. [Online]. Available: https://idet.tlaxcala.gob.mx/index.php/mision-vision