TRABAJO DE GRADO
Opción Seminario-Diplomado.

Plataforma de Cursos Online Saber Aprendelandia Basados en Microservicios

Corporación Universitaria Remington.
Facultad de Ingeniería
Ingeniería en Sistemas

Anyerson Arbey Tumbaco Rojas
William Mauricio Diaz Josa
Tutor: Diego Fernando Marín Lozano

Opción de Trabajo de grado Seminario-Diplomado
2025

Dedicatoria

Este trabajo lo dedicamos a nuestras personas más allegadas, como padres y hermanos, por el apoyo incondicional y motivación que nos brindan en el día a día, motivando no solo en nuestra formación profesional, si no a lo largo de nuestras vidas.

Agradecimientos

Agradecemos a Dios por permitirnos tener las condiciones de poder trabajar y estudiar dadas a las largas jornadas de trabajo. A la Corporación Universitaria Remington y sus docentes. Quienes, nos han brindado de su conocimiento y orientación, a lo largo de nuestra formación profesional, y en especial a nuestro orientador Diego Fernando Marín por el acompañamiento durante la elaboración de este seminario de grado.

Tabla de Contenido

Resumen

El siguiente trabajo analiza la problemática de las plataformas en línea, dando una solución mediante un desarrollo web para facilitar el acceso a la información de una manera sencilla, basada en la arquitectura de microservicios distribuidos, facilitando el desarrollo y su mantenimiento, ya que, al aplicar buenas prácticas, tecnologías, modernas y metodologías agiles, permite que sea adaptable, mantenible y escalable. Combinando bases de datos como PostgreSQL y MongoDB para la gestión de información y contenidos de la plataforma.

El desarrollo se ejecuta aplicando el framework FastAPI con Python minimizando la duplicidad de código ofreciendo un código limpio para un alto rendimiento, todo esto se logra mediante uso de contenedores con Docker, para incluir todo lo necesario que requiere la aplicación y permitir que esta se ejecute en un entorno diferente. Así mismo, todo esto se comprueba mediante pruebas con la herramienta pytest, garantizando el funcionamiento del código para una entrega funcional.

Finalmente, con su documentación automatizada con MkDocs, facilitando el conocimiento del desarrollo sirviendo de guía para nuevas propuestas de plataformas en línea, dando como resultado una mejor página web de cursos en línea, amigable y adaptable, en donde los usuarios puedan ver el contenido de los cursos avances y calificaciones, cumpliendo así con el objetivo.

Palabras clave

Transformación digital, microservicios, metodologías agiles, escalabilidad.

Pregunta orientadora de la búsqueda

¿Cuál es la propuesta para solucionar plataformas de aprendizaje que actualmente utilizan sistema basados en arquitecturas monolíticas, que permitan añadir nuevas características cuando se requieran sin que se van afectados sur servicios, además, que permitan el acceso fácil a la información mejorando la experiencia de aprendizaje?

Sustentación teórica de la pregunta

La arquitectura basada en microservicios da respuesta a las necesidades de los usuarios facilitando el acceso a la información y el aprendizaje, de manera ágil e intuitiva llevando un control de evaluaciones y proceso evolutivo (UNESCO, 2024), y a las limitaciones que presentan las arquitecturas monolíticas actuales, las cuales están diseñadas sobre una misma estructura o un mismo bloque, ocasionando un problema a la hora de realizar una actualización o incrementar una nueva funcionalidad afectando todo su servicio. Esta propuesta permite separar una arquitectura en componentes independientes, denominados servicios, los cuales pueden desarrollarse, desplegarse y mantenerse de forma autónoma.

Cada microservicio cumple una función dentro de la arquitectura, comunicándose con los demás a través de APIs. Este diseño facilita incorporar nuevas funcionalidades sin afectar o comprometer los demás servicios de la arquitectura. Por ejemplo, si el microservicio encargado de la autenticación de usuarios recibe un alto flujo de información, se interviene de manera individual, siendo así escalable, eficiente, estable y disponible (Ortega, 2020).

Los sistemas distribuidos, en una arquitectura de microservicios son importantes ya que cada servicio posee su propia base de datos, lo que asegura una baja dependencia entre ellos, integrando un sistema que combina bases de datos no relacionales como MongoDB, utilizado para desarrollo de páginas web, la gestión aislada y segura de información de usuarios (Ortega, 2020), junto con bases de datos relacionales como PostgreSQL, ofreciendo un sistema de código abierto que admite y ofrece muchas características modernas, agregando nuevas funcionalidades, para ser utilizado, modificado y distribuido de manera gratuita en nuestra página web de cursos (cita página oficial de PostgreSQL).

Todo el entorno se implementa mediante contenedores Docker para garantizar su portabilidad y ejecución sin complicaciones. Estos proporcionan una plataforma escalable y segura que permite crear, distribuir y ejecutar la página web con mayor rapidez, y optimizar los flujos de los servicios, garantizando implementaciones fluidas en toda la plataforma. Además, optimizan el rendimiento ejecutando los recursos de la plataforma junto con sus servicios, garantizando portabilidad y un entorno de ejecución sin importar el sistema operativo manejado. Mediante imágenes Docker, reproduciendo entornos idénticos en desarrollo, prueba y despliegue, para minimizar los conflictos por versiones o configuraciones. Además, la combinación con herramientas de orquestación como Docker Compose automatiza el despliegue y escalado, generando una infraestructura más dinámica (Ortega, 2020), (Sanchez, 2022).

Integrando lenguaje de desarrollo Python con el framework web moderno FastAPI, aportando un alto rendimiento, optimización de código, reducción de errores y facilidad para documentar, dando un aporte esencial en seguridad sin comprometer las bases de

datos, integrando tokens con JWT. Todo esto se verifica mediante pruebas, asegurando la funcionalidad del código y la correcta entrega en cada iteración . (FastAPI)

El desarrollo de sistemas basados en microservicios se alinea con los principios de las metodologías ágiles, en las cuales adoptamos la metodología incremental. Cada incremento, ya sea actualización o desarrollo, puede definirse como una iteración que incorpora mejoras o nuevas funcionalidades en los servicios. Esto permite la retroalimentación constante y reducir los riesgos de errores a implementaciones futuras. Además, facilitar el trabajo en conjunto, donde cada grupo puede especializarse en un microservicio determinado, manteniendo independencia en su ciclo de desarrollo (Leon, Acosta, & Diaz, 2021).

La arquitectura de microservicios es una transformación que impulsa la creación de sistemas modulares, escalables y sostenibles, capaces de adaptarse a las necesidades cambiantes de los usuarios. Esta plataforma, con dicha arquitectura, permite construir un sistema con módulos específicos para la gestión de aprendizaje, comunicación de estudiantes e instructores, evaluación y contenido de los cursos, garantizando que la evolución de uno no afecte la estabilidad de los demás.

La arquitectura de microservicios combinada con tecnologías como Docker, PostgreSQL, MongoDB, y Python con FastAPI proporciona una base sólida para desarrollar plataformas modernas, modulares y escalables. Esto permite optimizar recursos, reducir errores, mantener la continuidad y facilitar el crecimiento del sistema mediante una implementación ágil y controlada en cada iteración de desarrollo, dando

como resultado una plataforma de aprendizaje adaptable para el usuario con aprendizaje continuo.

Problema

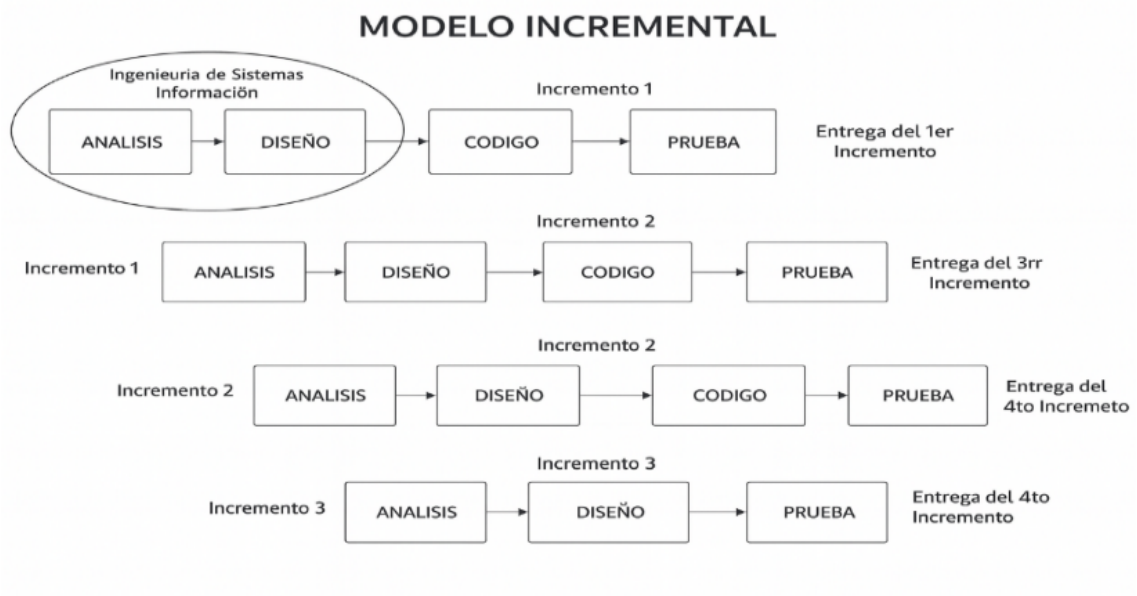
Las problemáticas de las plataformas de cursos en línea, es que muchas plataformas no cuentan con disponibilidad de información, ya que muchos contenidos están protegidos por derechos de autor, y otra parte de la información está de manera física limitando así el acceso y no se tienen un seguimiento de aprendizaje personalizado (Navarro, Berbek, & Sanchez, 2024). Además, muchos usuarios no pueden acceder por temas de distancias, discapacidades, tiempo y responsabilidades diarias (UNESCO, 2024).

Dichas plataformas de aprendizaje están construidas en arquitecturas monolíticas, las cuales ejecutan todos los procesos sobre un mismo servicio, afectando así su rendimiento, estas plataformas dificultan su escalamiento, ya que al realizar un cambio o añadir características o información se pueden ver afectados otros procesos, ocasionando la caída del servicio en su totalidad y afectando disponibilidad frente a los usuarios. Así mismo, al ser un solo servicio, tienden a demandar muchos recursos, siendo así más lentas y poco fiables para los usuarios. (Ortega, 2020).

Metodología

La metodología incremental es un ciclo de desarrollo de software propuesto por Lehman en 1984. Resulta de la combinación de elementos del modelo en cascada y en la combinación de prototipos (Aguirre & Gomez, 2002), en donde la entrega se realiza por incrementos, es decir, desarrollos por etapas, en donde cada desarrollo debe ser funcional (ver figura 1), cumpliendo con sus requisitos, el diseño, la codificación y las pruebas en cada uno. Para esta metodología cada incremento representa una versión mejorada del software, permitiendo agregar funcionalidades o perfeccionar las versiones ya desarrolladas, desacuerdo a las necesidades de los usuarios en cada ciclo. de esta manera, el sistema permite una integración progresiva de desarrollo y los resultados obtenidos en cada iteración, siendo así una evolución incremental, ya que el sistema ya no se ve como una entidad monolítica con una fecha fija de entrega ,sino como una integración de resultados sucesivos obtenidos en cada iteración, facilitando la detección de errores y corrigiendo de manera eficiente sin comprometer los demás incrementos desarrollados ((Piattini , 2018)),

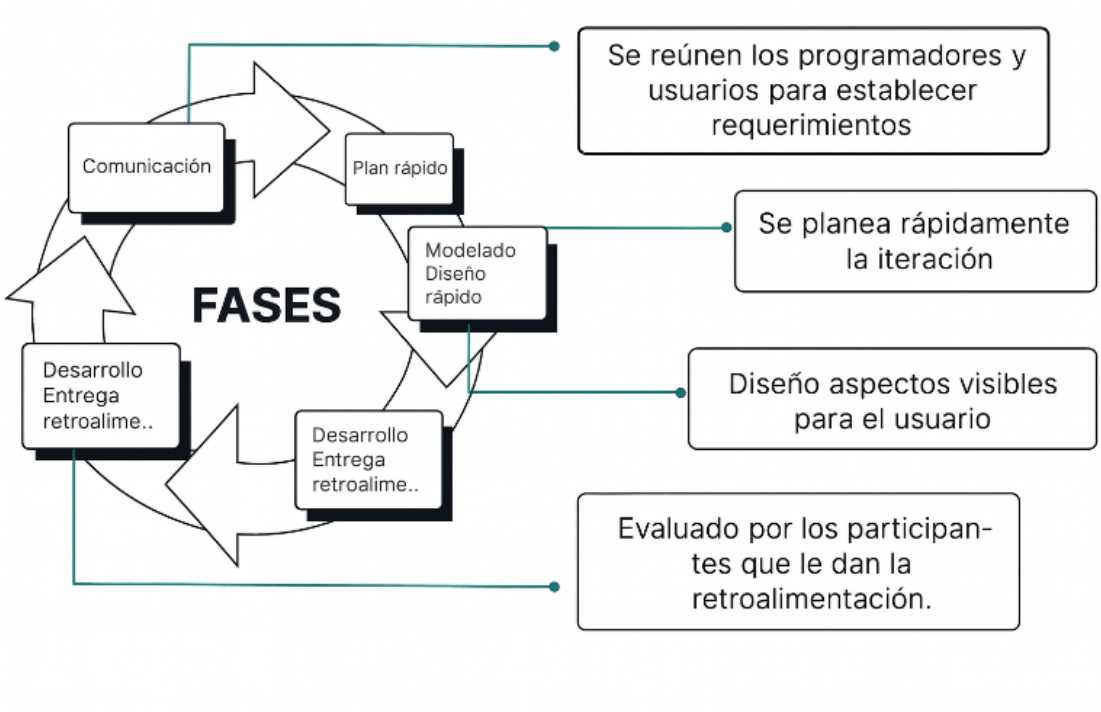
Figura 1 Modelo Incremental



fuente: Anyerson Tumbaco, William Diaz

Basada en la problemática que se describe anteriormente, se realiza un desarrollo de una plataforma de cursos en línea basada en microservicios distribuidos, para ello se optó por emplear metodología Ágil Incremental, dado que esta se adapta de manera óptima a la propuesta. Esta metodología permite desarrollar una aplicación web con capacidad de escalabilidad, modularidad, adaptabilidad, fácil mantenimiento e implementación de información necesaria. Permitiendo desarrollos por iteraciones (ver figura 2), de manera incremental esta metodología es ideal para la demanda de usuarios que requieren plataformas de aprendizaje para minimizar y optimizar sus tiempos (Leon, Acosta, & Diaz, 2021).

Figura 2 Metodología incremental y sus fases



Fuente: Anyerson Tumbaco, William Diaz.

Desarrollo

Para solventar problemática expuesta anteriormente, se propone un sistema basado en microservicios, tales como autenticación, gestión de perfiles, gestión de contenidos o información digital, evaluación y rendimiento estos se desarrollan a través de APIs e interfaz modular, cada uno actuando de manera autónoma sin afectarse entre sí.

Además, estarán aislados mediante contenedores Docker lo cual nos permite que la ampliación incluya todas las herramientas necesarias para ejecutarse de manera fiable y segura en cualquier equipo. (Sanchez,2022, Contenedores Docker), esta estructura facilita detectar fallas e intervenir en el microservicio afectado sin que se vean comprometidos los demás servicios, integrando FastAPI con Python para disminuir duplicidad de código garantizando tiempos de respuesta óptimos (Cruz, S.F).

Finalmente, los servicios se pueden incrementar según sea la necesidad sin afectar la estructura general, dando como resultado una aplicación web modular, ágil, escalable y sostenible de cursos en línea.

Comunicación y planificación.

Una vez analizada la problemática que presentan las plataformas de cursos en línea, mencionadas anteriormente, se procede, a crear diagramas y definir los requisitos para comenzar su desarrollo por iteraciones como indica la siguiente tabla.

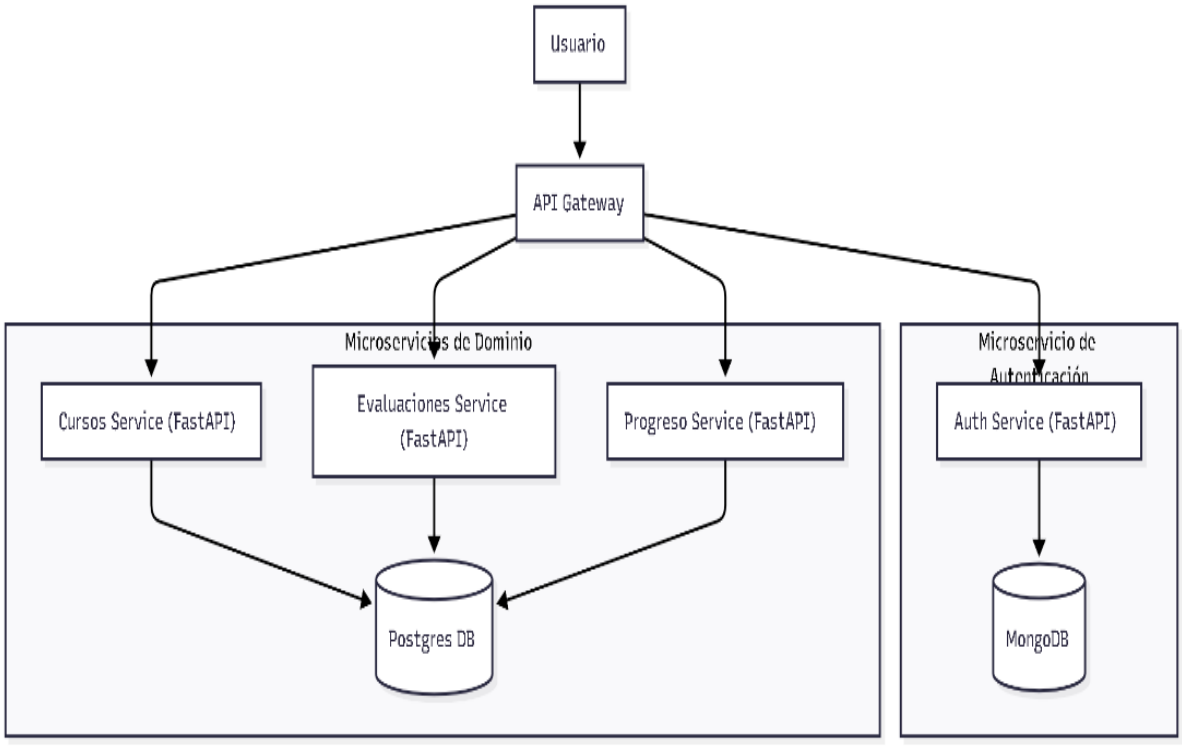
Tabla 1 Tabla de iteraciones o requisitos

Incremento	Desarrollo	Entregable principal	Responsable(s)
Incremento 1	Microservicio de autenticación	Código autenticación y pruebas	Anyerson Tumbaco
Incremento 2	Cursos, evaluaciones, progreso	Cursos, evaluaciones, progreso y pruebas	Anyerson Tumbaco, William Diaz
Incremento 3	API funcionando	Gateway funcional con rutas y pruebas	William Diaz
Incremento 4	Interfaz de usuario conectada a API Gateway	Frontend operativo integrado y pruebas	William Diaz
Incremento 5	Orquestación y documentación automática MKDocs	Url local con Documentación de página web	Anyerson Tumbaco, William Diaz

Diagrama de arquitectura de proyecto.

A continuación, se presenta diagrama de plataforma de cursos basada en microservicios como autenticación, cursos, evaluaciones y progreso, en este se muestra su funcionamiento con sus servicios implementados con FastAPI. El usuario interactúa a través de API Gateway, empleando PostgreSQL y MongoDB (ver figura 3).

Figura 3 Diagrama de arquitectura de proyecto

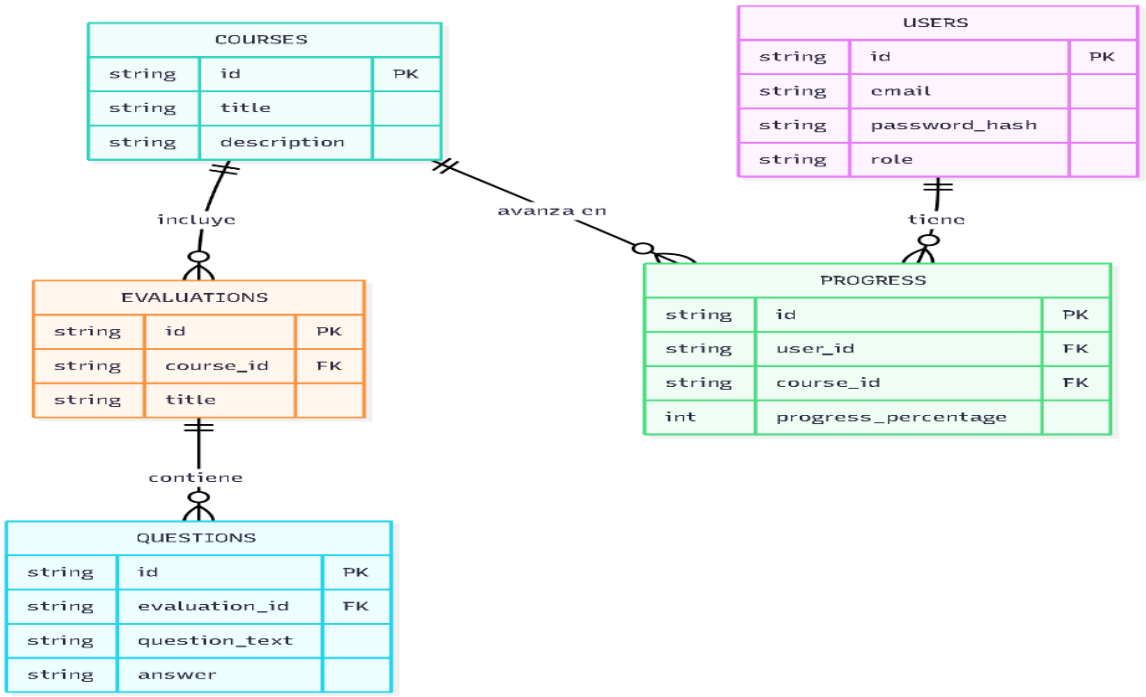


Fuente: Anyerson Tumbaco, William Diaz

Diagrama de bases de datos.

A continuación, se presenta diagrama de bases de datos MongoDB y PostgreSQL en un modelo relacional. Almacenando la información de usuarios cursos, evaluaciones y progreso, las entidades permiten la integración de la información entre los servicios, utilizando datos compartidos y evitando dependencias directas entre las mismas (ver figura 4).

Figura 4 Diagrama de bases de datos



Fuente: Anyerson Tumbaco, William Diaz

Incremento 1: Autenticación

Para la siguiente autenticación, se desarrollan las siguientes tareas utilizando tecnologías como FastAPI, MongoDB y JWT lo que permite que solo los usuarios registradores puedan ingresar de forma segura.

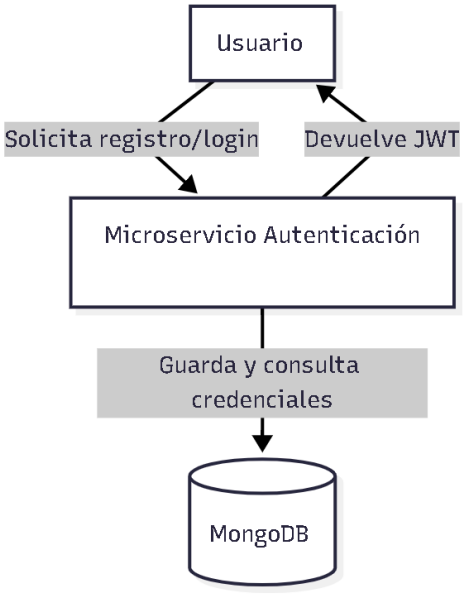
Desarrollos.

- ✓ Registro de usuarios
- ✓ Inicio de sesión
- ✓ Generador de jwt
- ✓ Validación de credenciales

Diagrama de autenticación:

A continuación, se presenta diagrama de autenticación mediante JWT, a través del inicio de sesión se obtiene y ser verifican los permisos, generando un token de confirmación para el acceso. MongoDB facilita la autenticación y garantiza el ingreso a la plataforma gestionado los datos de manera segura (ver figura 5).

Figura 5 Diagrama de autenticación.



Fuente: Anyerson Tumbaco, William Diaz

El siguiente código corresponde a un endpoint que responde a solicitudes POST asignadas a la url (“/register”), utilizando el decorador @app.post para definir el método y la ruta. Se realiza una validación de correo y contraseña verificando si ya existe el usuario para evitar datos repetidos, permitiendo registro según sea el rol.

```
@app.post("/register")
def register(user: UserCreate):
    if users.find_one({"email": user.email}):
        raise HTTPException(status_code=409, detail="Email already registered")
    hashed = get_password_hash(user.password)
    users.insert_one({"email": user.email, "password": hashed, "role": user.role,
"created_at": datetime.utcnow()})
    return {"message": "user created", "role": user.role})
```

Incremento 2: Microservicios de dominio

Se realiza el desarrollo mediante tecnología FastAPI con Python, de los microservicios encargados de gestionar todo el contenido del curso. En esta etapa interactúan el instructor y estudiante (ver figura 6).

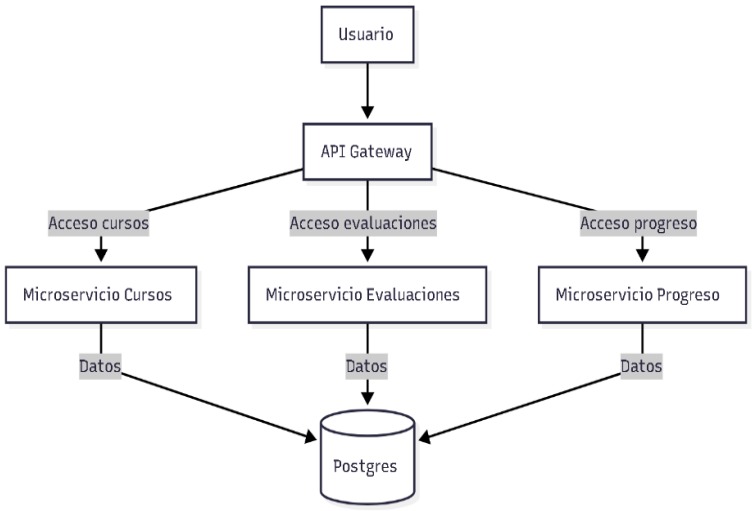
Desarrollos.

- ✓ Microservicio de cursos.
- ✓ Microservicio Evaluaciones.
- ✓ Microservicio Progreso.

Diagrama de microservicios.

El siguiente diagrama representa la arquitectura de los microservicios, donde el API Gateway es el punto de entrada de las solicitudes dirigiéndolas a los servicios de cursos, evaluaciones y progreso de manera independiente para que estos no se vean afectados.

Figura 6 Diagrama de microservicios



Fuente: Anyerson Tumbaco, William Diaz

El siguiente código de microservicio de cursos permite agregar nuevos cursos mediante el método POST, a través de la función create_curso validando el ID de curso y validando el registro. Añadiendo la información al contenedor.

```
def create_curso(curso: Curso):
    if any(c.get("id") == curso.id for c in DATA.get("cursos", [])):
        raise HTTPException(status_code=400, detail="Curso ya existe")
    DATA["cursos"].append(curso.dict())
    return {"message": "Curso creado", "curso": curso.dict()}
```

El siguiente código de microservicio de evaluaciones permite gestionar la consulta de cuestionarios, utilizando el endpoint GET para enviar respuestas y calcular la calificación método POST.

```
@app.get("/{cuestionario_id}")
def get_cuestionario(cuestionario_id: str):
    q = DATA.get("cuestionarios", {}).get(cuestionario_id)
    if not q:
        raise HTTPException(status_code=404, detail="Cuestionario no encontrado")
    safe = {"id": q["id"], "titulo": q["titulo"], "preguntas": []}
    for p in q.get("preguntas", []):
        qp = p.copy(); qp.pop("respuesta", None); safe["preguntas"].append(qp)
    return safe

@app.post("/{cuestionario_id}/responder")
def responder(cuestionario_id: str, body: Respuestas):
    return {"score": score, "correct": correct, "total": total}
```

El siguiente código indica el microservicio de progreso del estudiante, recibiendo las solicitudes mediante el decorador @app.get, el endpoint muestra el avance dentro de las dashboard del API Gateway, si el usuario no tiene ningún progreso registrado, el sistema simula avances con porcentajes bajos.

```
@app.get("/estudiantes/{estudiante_id}/cursos")
def progreso_estudiante(estudiante_id: str):
    existing = DATA.get("progreso", {}).get(estudiante_id)
    if not existing:
        ...
        DATA["progreso"][estudiante_id] = {"cursos": cursos_asignados}
        return DATA["progreso"][estudiante_id]
    return existing
```

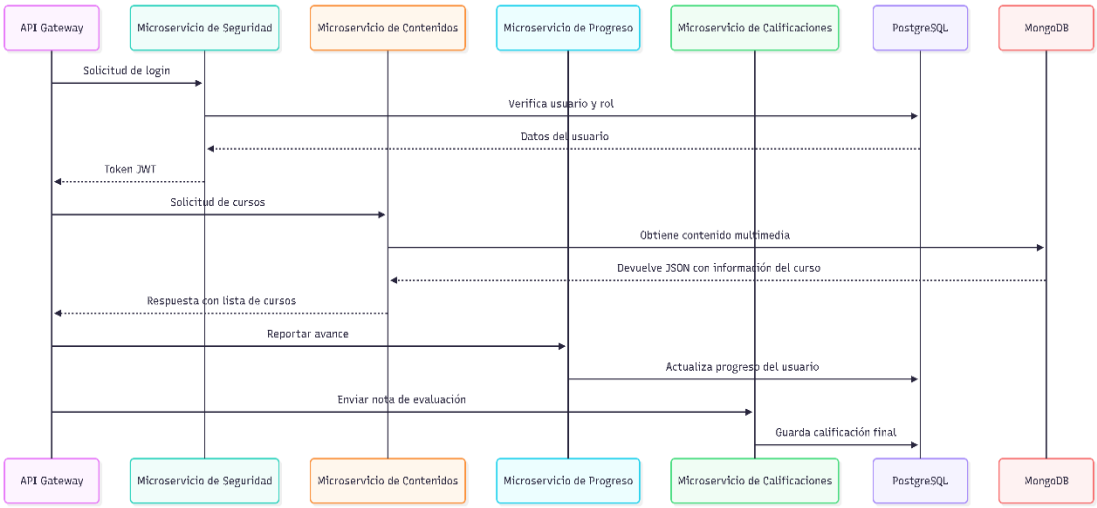
Incremento 3: API Gateway

Se aplican rutas genéricas protegidas con jwt a través de API Gateway para controlar el acceso seguro a los servicios.

Diagrama de API Gateway.

A continuación, se presenta diagrama de API Gateway donde se representa la iteración entre los servicios y la API Gateway durante su ejecución, siendo el punto de entrada para ñas solicitudes desde la autenticación, permitiendo el desplazamiento por los dashboard según el rol del usuario, gestionando las solicitudes que se realicen en la plataforma (ver figura 7).

Figura 7 Diagrama de API Gateway



Fuente: Anyerson Tumbaco, William Diaz

El siguiente código pertenece al enrutamiento del API Gateway, donde el endpoint GET, permite las solicitudes a diferentes microservicios utilizando los parámetros service_name y path, las cuales representan la ruta de envío. La API Gateway construye la url correspondiente a la solicitud y ofrece una respuesta.

```
@router.get("/{service_name}/{path:path}")
async def forward_get(service_name: str, path: str, request: Request):
    headers = {}
    auth = request.headers.get("Authorization")
    if auth: headers["Authorization"] = auth
    response = requests.get(f"{SERVICES[service_name]}/{path}",
params=request.query_params, headers=headers)
    return response.json(
```

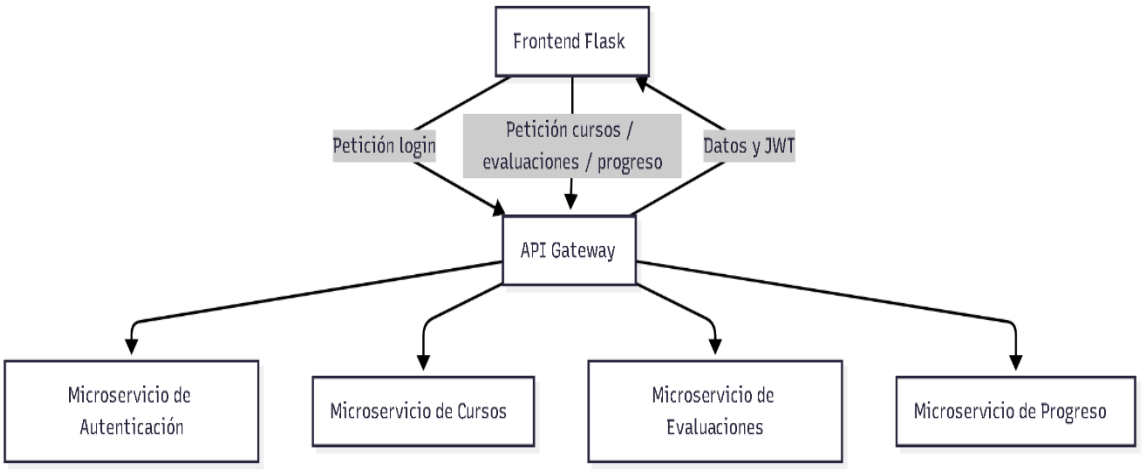
Incremento 4: Frontend

Para el desarrollo de Frontend se utiliza framework Flask, siendo ligero y flexible, en donde los usuarios podrán iniciar sesión, y navegar por los diferentes microservicios que ofrece la plataforma de manera ágil y eficiente.

Diagrama de Frontend.

A continuación, se presenta diagrama de Frontend que muestra la comunicación desde el frontend Flask, a través de la interacción del usuario, hacia el proceso de autenticación, coordinando las solicitudes y gestionando los tokens automáticos, siendo el API Gateway el orquestador de los procesos correspondientes de cada servicio (ver figura 8), (ver figura 9).

Figura 8 Diagrama de Frontend.



Fuente: Anyerson Tumbaco, William Diaz

El siguiente código corresponde a la vista de inicio de sesión dentro del servicio Fronted, respondiendo a las solicitudes GET Y POST mediante el decorador @app. route, permitiendo el acceso al usuario según el rol , permitiendo el ingreso al dashboard.

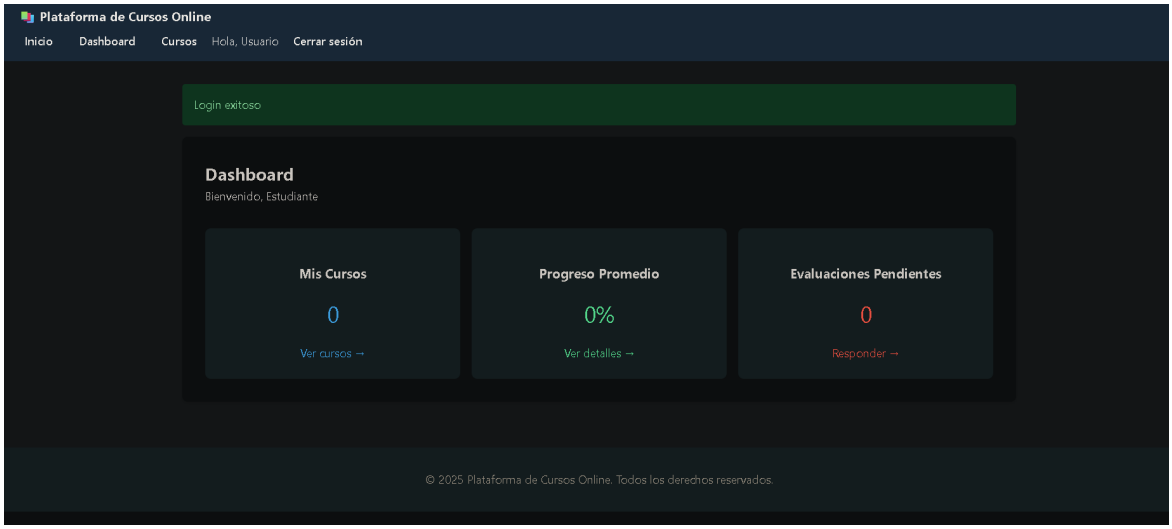
```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('email'); password = request.form.get('password')
        resp = _call_service('POST','auth','login', json={'email': email, 'password':
password})
        if resp and 'access_token' in resp:
            session['access_token'] = resp['access_token']
            user_info = _call_service('GET','auth','me')
            if user_info and user_info.get('user'):
                u = user_info['user']; session['user'] = {'id': u.get('id'), 'email': u.get('email'),
'role': u.get('role','estudiante')}
            return redirect(url_for('dashboard'))
```

```
        flash('Email o contraseña incorrectos', 'error')
    return render_template('login.html')

<nav>
  <a href="{{ url_for('index') }}">Inicio</a>
  {% if session.get('access_token') %}
    <a href="{{ url_for('dashboard') }}">Dashboard</a>
    <a href="{{ url_for('cursos_list') }}">Cursos</a>
    <span>Hola, {{ session.get('user', {}).get('role','Usuario').capitalize() }} ({{
session.get('user', {}).get('email','') }})</span>
    <a href="{{ url_for('logout') }}">Cerrar sesión</a>
  {% else %}
    <a href="{{ url_for('login') }}">Iniciar sesión</a>
    <a href="{{ url_for('register') }}">Registrarse</a>
  {% endif %}
</nav>
```

La siguiente captura de pantalla muestra la vista de inicio de sesión en la dashboard del usuario estudiante, quien obtienen los permisos de ingreso mediante la autenticación.

Figura 9 Dashboard.



Fuente: Anyerson Tumbaco, William Diaz

Incremento 5: Despliegue y orquestación

En este incremento se realiza despliegue mediante contenedores con Docker, con los requisitos que requiere el desarrollo de la plataforma y posteriormente se realiza su entrega la documentación, ver diagrama de orquestación y creación de contenedores e imágenes.

Desarrollos

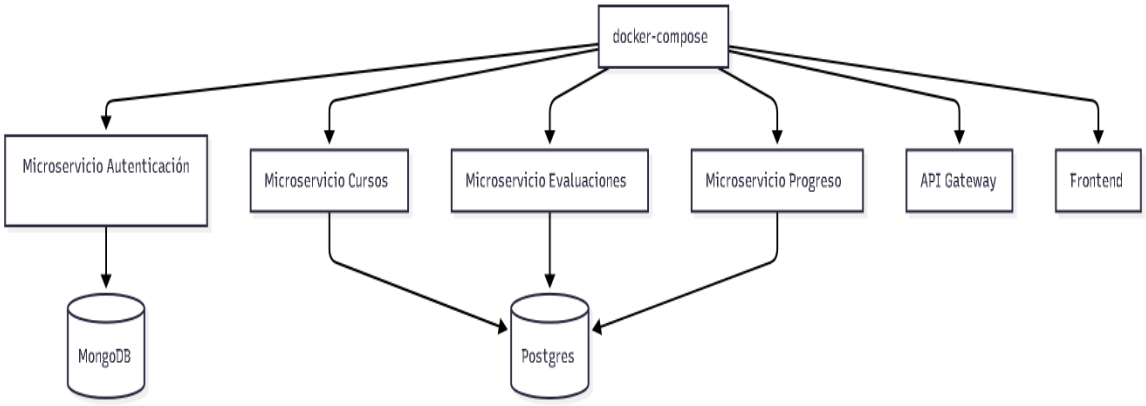
- ✓ Contenedorizacion con Docker.
- ✓ Documentación.

Orquestación con Contendores usando Docker.

A continuación, se muestra diagrama de orquestación de contenedores (ver figura 10, 11 y 12)

El siguiente diagrama indica la orquestación con Docker compose, como las arquitecturas del Frontend Flask, con su contenedor independiente, el API Gateway como punto único de entrada según las rutas, y los microservicios ejecutados en contenedores separados incluyendo las bases de datos MongoDB y PostgreSQL.

Figura 10 Diagrama de despliegue y orquestación



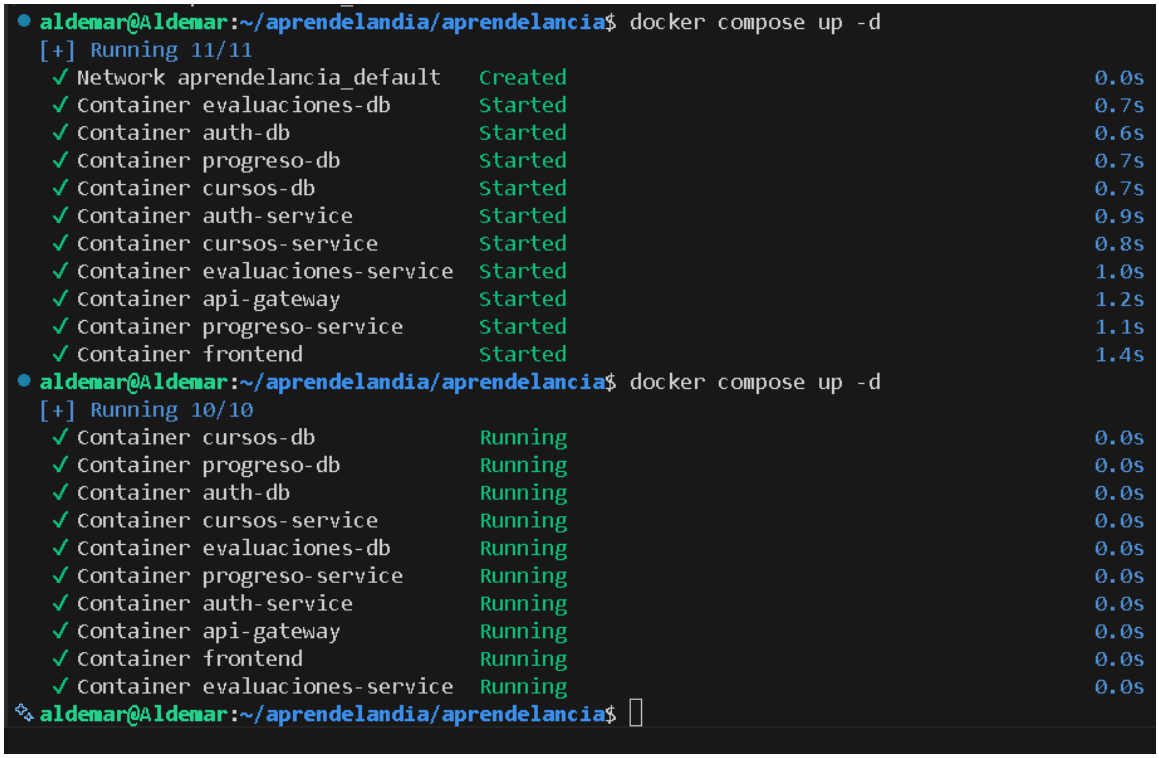
Fuente: Anyerson Tumbaco

El siguiente código contiene la configuración para el despliegue del micro servicio de autenticación mediante docker compose, ejecutando el servicio en su base de datos, orquestando los demás contenedores de forma automatizada, aislada y reproducible. reflejando el despliegue de los microservicios.

```
auth-service:
  build: ./services/authentication
  ports:
    - "8001:8001"
  environment:
    - DATABASE_URL=mongodb://auth-db:27017/auth_db
  depends_on:
    - auth-db
auth-db:
  image: mongo:latest
  ports:
    - "27017:27017"
```

La siguiente captura de pantalla muestra la salida del comando Docker compose up –d, donde se observa la arquitectura del proyecto y la activación de todos los procesos aislados, imágenes y contenedores que componen la plataforma, cada servicio inicia en su puerto asignado, incluyendo el API Gateway, la creación de las imágenes y el inicio de cada contenedor conforme a lo definido en el archivo docker-compose.yml.

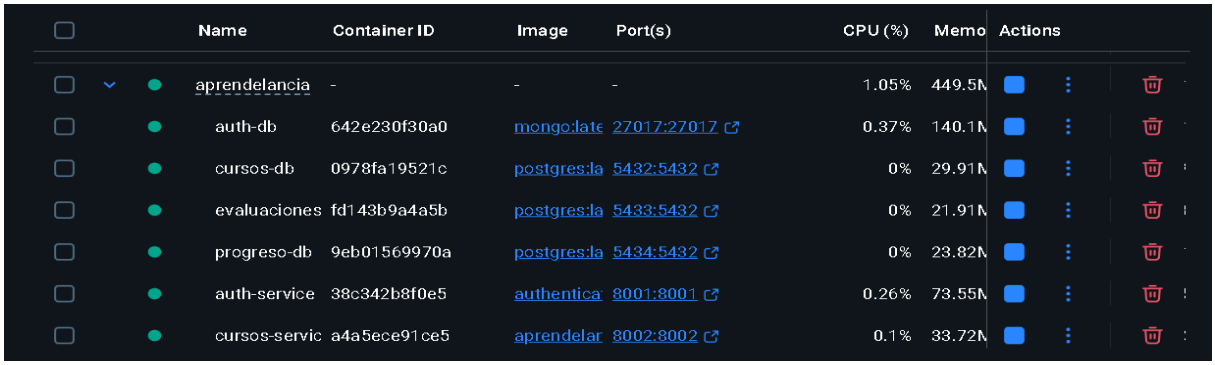
Figura 11 Creación de contenedores



Fuente: Anyerson Tumbaco, William Diaz

En la siguiente captura se observa la interfaz de Docker con sus contenedores y servicios ejecutándose con sus APIs y puertos asignados, al igual que las bases de datos independientes para cada servicio.

Figura 12 Contenedores en Docker



Fuente: Anyerson Tumbaco, William Diaz

Documentación.

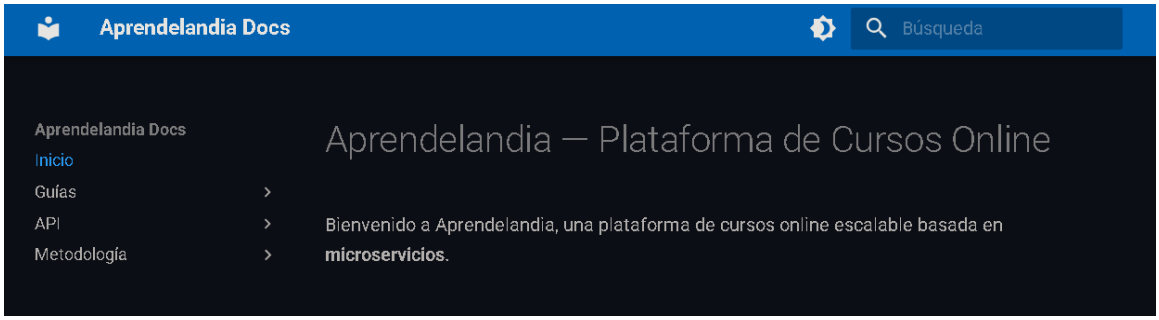
A continuación, se muestra la documentación de código del proyecto mediante MKDocs, llamado a los sitios donde este se encuentra para que sean visualizados mediante puerto local (ver figura 13).

```
site_name: Aprendelancia Docs
theme:
  Inicio: material
  palette:
    primary: blue
    accent: orange
nav:
  - Home: index.md
  - Arquitectura: architecture.md
  - Setup & Configuración: setup.md
```

```
  - API Reference: api.md
  - Desarrollo: development.md
  - Entrega Incremental: entrega.md
docs_dir: docs
markdown_extensions:
  - codehilite
  - extra
```

La siguiente captura muestra la pantalla de la documentación por medio de puerto local, mostrando la guía de cómo funciona la plataforma de cursos.

Figure 13 Documentación de código



Fuente: Anyerson Tumbaco, William Diaz.

Conclusiones

La combinación de Docker, FastAPI, y Python en la propuesta permite crear una plataforma segura, ágil y sostenible, Docker asegurando un despliegue aislado en contenedores, FastAPI ofreciendo un framework rápido y moderno, garantizando la calidad.

Aplicado la metodología incremental la propuesta basada en la arquitectura de microservicios, nos permite construir plataformas de cursos en línea flexibles por iteraciones con entregas en cada iteración, ofreciendo una mejora en cada entrega.

Esta propuesta aporta un material importante al campo de la educación digital, al establecer una metodología que sirve como referencia para futuras investigaciones y desarrollos de transformación digital.

Las plataformas basadas en microservicios son importantes, ya que nos permiten intervenir un servicio, ya sea para validar una falla o realizar un cambio requerido, sin afectar todos los servicios que este contiene.

Referencias

Cruz, A. (S.F). *Primeros pasos con FastApi Aquí continúa tu camino en el desarrollo de aplicaciones web en Python con FastApi.* (A. Cruz, Ed.) Chicago.

Galan, J., Martin, A., Bernal, C., & Lopez, E. (2017). *Los MOOC y la Educación Superior.* (E. Octaedro, Ed.) Obtenido de https://www.google.com.co/books/edition/Los_MOOC_y_la_Educaci%C3%B3n_Superior/tgiIDwAAQBAJ?hl=es&gbpv=0.

Leon, A. R., Acosta, J. L., & Diaz, R. A. (2021). Apliacacion de la metodologia incremental en el desarrollo de sistema de informacion. Obtenido de http://scielo.sld.cu/scielo.php?pid=S2218-36202021000500175&script=sci_arttext

Muñoz, P. C. (2009). *Plataformas de teleformación y herramientas telemáticas.* España.

Navarro, M., Berbek, P., & Sanchez, J. (2024). *Investigacion y conocimientos en la eduacacion actual.* (S. Dykinson, Ed.) Chicago.

Newman, S. (2019). *Monolith to Microservices:Evolutionary Patternsto Transform Your Monolith.* Chicago: O'Reilly Media, Incorporated.

Ortega, J. M. (2020). *Tecnologías para arquitecturas basadas en microservicios.* (J. M. Ortega, Ed.)

Rocio , E. (2005). *Trabajo colaborativo en educación universitaria:.* (N. E. Educativa, Ed.) Mexico.

Sanchez, J. J. (2022). *Aprender Docker, un enfoque práctico.* España : Marcombo.

UNESCO. (2024). (UNESCO, Ed.) Obtenido de https://www.google.com.co/books/edition/Informe_de_seguimiento_de_la_educaci%C3%B3n/gNINEQAAQBAJ?hl=es&gbpv=0

Usaola, M. (2015). (M. P. Usaola, Ed.) España. Obtenido de https://www.google.com.co/books/edition/_/UiAvCwAAQBAJ?hl=es-419&gbpv=0