TRABAJO DE GRADO
Seminario desarrollo moderno con Python.

MesaNova: Desarrollo de plataforma modular para la gestión automatizada de reservas en restaurantes mediante microservicios.

Corporación Universitaria Remington
Facultad de Ingenierías
Ingeniería en Sistemas.

Harold Piedrahita Valencia
Sebastián Ramírez Parra
Tutor: Diego Fernando Marín Lozano

Opción de Trabajo de grado Seminario-Diplomado
2025.

Dedicatoria

Dedicamos este proyecto como primera medida a Dios, a la familia por su apoyo incondicional, nuestros padres son parte fundamental del desarrollo de esta carrera y motivación constante para continuar día a día conquistando metas, además a nuestra Corporación Universitaria Uniremington por la oportunidad y el aprendizaje impartido con todo el profesionalismo.

Agradecimientos

Agradecemos a Dios por concedernos la sabiduría, la fuerza y la claridad necesaria para desarrollar este proyecto. También extendemos nuestro agradecimiento a nuestros profesores, quienes con su compromiso, guía y apoyo constante han influido significativamente en nuestro desarrollo profesional y personal. De la misma forma, queremos expresar nuestra gratitud a la Corporación Universitaria Uniremington por proporcionarnos una formación de excelencia y un equipo docente comprometido, que ha mejorado cada uno de nuestros aprendizajes y nos ha acompañado en este recorrido académico.

Contenido

Resumen

La gestión de reservas de restaurantes tiene una continua dependencia, en muchos casos, de procedimientos que se hacen de forma manual o de sistemas con baja integración, esto genera ineficiencias operativas, errores en la asignación de mesas y experiencias limitadas para el cliente. En respuesta a esta problemática, este proyecto busca como objetivo diseñar e implementar una plataforma de reservas para restaurantes empleando una arquitectura basada en microservicios, la finalidad de dicho proyecto es mejorar la flexibilidad, escalabilidad y mantenibilidad del sistema que permita mejorar la experiencia del cliente.

Los resultados preliminares sugieren que la adopción de microservicios puede mejorar la escalabilidad, reducir el acoplamiento entre componentes y permite una administración más eficiente de los recursos del sistema. Se puede concluir que este enfoque es una alternativa viable para la digitalización del proceso de reservas en restaurantes y crea bases sólidas para futuras extensiones hacia módulos de analítica y personalización en la experiencia del cliente.

Palabras Clave

Microservicios, Reservas, Transformación digital, tecnologías y Desarrollo de software, Flask, FastAPI, Docker, Contenedores.

Pregunta Orientadora

¿Cómo puede una plataforma basada en microservicios optimizar procesos de reservas en restaurantes y contribuir a la gastronomía en el sector digital?

Sustentación teórica de la pregunta

Un sistema de reserva de restaurante es una plataforma diseñada para automatizar y gestionar reservas, pedidos y disponibilidad en mesas, la idea principal es facilitar la interacción entre el cliente y el restaurante. (Hernandez Acosta, 2024). Estos sistemas surgen como respuesta a la necesidad de mejorar la eficiencia operativa y disminuir los errores de métodos manuales, especialmente las reservas telefónicas las cuales tienen más probabilidad de inconsistencias (Guanilo Pareja, Barragán Codina, & Guerra Rodríguez, 2024).

En el desarrollo de soluciones tecnológicas se emplean arquitecturas de software modernas, como lo son los microservicios, los cuales dividen el sistema en módulos independientes (Usuario, reservas, menú y autentificación) lo cual mejora la escalabilidad, el mantenimiento y su naturaleza modular permite futuras integraciones de funciones (Ortega Candel J. M., 2020). Además, reduce riesgos permitiendo que el sistema evolucione con las necesidades del negocio.

Por su parte, el uso de las de datos relacionales, como PostgreSQL permite una estructura eficiente y organizada que garantiza la integridad, consistencia en el manejo de la información (Zea Ordóñez, Molina Ríos, & Redrován Castillo, 2017). Esto es relevante

en el sistema, donde se gestionan datos sensibles como usuarios, horarios, mesas y disponibilidad.

En este contexto, la digitalización de los procesos operativos mejora la experiencia del cliente reduce errores humanos y ayuda a la transformación digital en el sector gastronómico. (Ortega Candel J. M., 2020). En la actualidad, los sectores gastronómicos se encuentran en procesos de transformación digital por la alta demanda en servicios más eficientes, personalizados y accesibles. Las reservas telefónicas incrementan la probabilidad de errores operativos. Esta situación ha motivado a establecimientos a adoptar soluciones tecnológicas que mejoren su eficiencia y experiencia del cliente (Guanilo Pareja, Barragán Codina, & Guerra Rodríguez, 2024).

Ante esta problemática y tendencias actuales, el presente proyecto busca diseñar e implementar una plataforma digital para la reserva de restaurantes, basada en una arquitectura de microservicios, la cual divide el sistema en componentes independientes. Esta arquitectura permitirá un desarrollo escalable, mantenible y modular, lo que facilitará futuras integraciones con otros servicios (Jiménez, R., Mazo;, 2022). De esta manera,

la plataforma digital de reservas no solo busca optimizar los procesos internos del restaurante, sino también aportar a la digitalización del sector gastronómico en la región.

Definición del problema

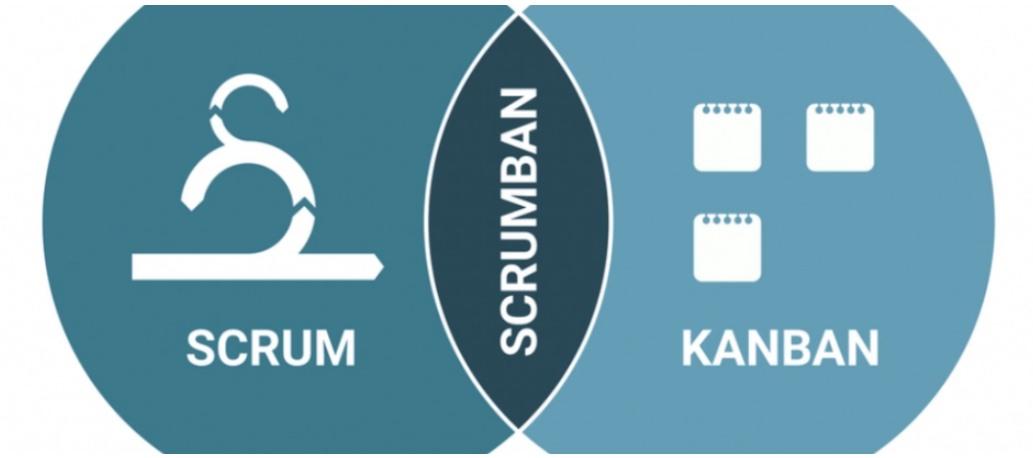
Actualmente, numerosos restaurantes gestionan las reservas y pedidos de manera manual o con sistemas poco integrados (Hernández Trujillo & Gil Gómez, 2022). Esta operación genera ineficiencias y errores operativos y en la asignación de la reserva, lo que deteriora la experiencia del cliente, la digitalización podrá otorgar a los usuarios la opción de visualizar disponibilidad de mesas y menú en tiempo real antes de su visita. La ausencia de una infraestructura tecnológica que integre los procesos de reservas y pedidos impiden optimizar los recursos disponibles (Hernandez Acosta, 2024).

Desarrollo.

Metodología:

El desarrollo se llevó a cabo con la metodología Scrumban, una adaptación de Scrum y Kanban que permitió avanzar de manera organizada, visual, interactiva y controlada en el proyecto. El proceso se gestionó mediante tablero Kanban digital, con columnas: por hacer, en progreso, en revisión y hecho, lo que permitió ver el progreso en tiempo real (Reddy, 2015).

Figura 1. Metodología incremental y sus fases.



1SCRUMBAN EN RELACIO A SCRUM Y KANBAN

Fuente: https://dev.to/oroscoloyamena/que-es-la-metodologia-scrumban-y-como-utilizarla-3933

Para evitar la sobrecarga de tareas y mantener un flujo de trabajo constante al trabajo se usó WIP (Work in Progress), las tareas relacionadas con los endpoints iniciaban en la columna por hacer, donde se definen los criterios de aceptación (Cabezas Benitez, 2025); luego pasa a proceso en progreso para la codificación; posteriormente pasa a revisión donde se hace uso de Pytest; cuando pasa por cada uno de estos procesos se marca como hecho.

Tabla 1. Tabla de Sprints y requisitos.

Resumen	Clave de incidencia	Nombre del proyecto	Tipo de proyecto	Persona asignada	Sprint
despliegue en Git Pages	MPDRO-21	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 6 Documentación
documentación automática	MPDRO-20	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 6 Documentación
Creación de documentos con modos y diagramas en mermad	MPDRO-19	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 6 Documentación
Actualización de README	MPDRO-18	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 6 Documentación
mejoras de las vistas aumento de botones y funcionalidades	MPDRO-17	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 5 Desarrollo Frontend
pruebas de funcionalidad manuales con endpoints	MPDRO-16	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 5 Desarrollo Frontend
Pruebas automáticas	MPDRO-15	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 5 Desarrollo Frontend

Desarrollo de las vistas para cada servicio	MPDRO-14	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 5 Desarrollo Frontend
pruebas automáticas+	MPDRO-13	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 4 APIGateway
Configuración de la misma con sus endpoints	MPDRO-12	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 4 APIGateway
Creacion del API	MPDRO-11	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 4 APIGateway
pruebas automáticas	MPDRO-10	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 2 Contenedores orquesta
pruebas automáticas	MPDRO-9	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 3 Desarrollo de microservicios
Creación del servicio de autenticación	MPDRO-8	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 3 Desarrollo de microservicios
Creación de los servicios reservas restaurantes menú	MPDRO-7	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	
configuración para la orquestación y levantamiento automático	MPDRO-6	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 2 Contenedores orquesta
Creación de contenedores para las BD	MPDRO-5	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 2 Contenedores orquesta
Creación de contenedores por cada servicio	MPDRO-4	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 2 Contenedores orquesta
Toma de decisiones	MPDRO-3	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 1 Análisis y Diseño
Diseño del software	MPDRO-2	MesaNova Plataforma de reservas online	software	Sebastian Ramirez Parra	Sprint 1 Análisis y Diseño

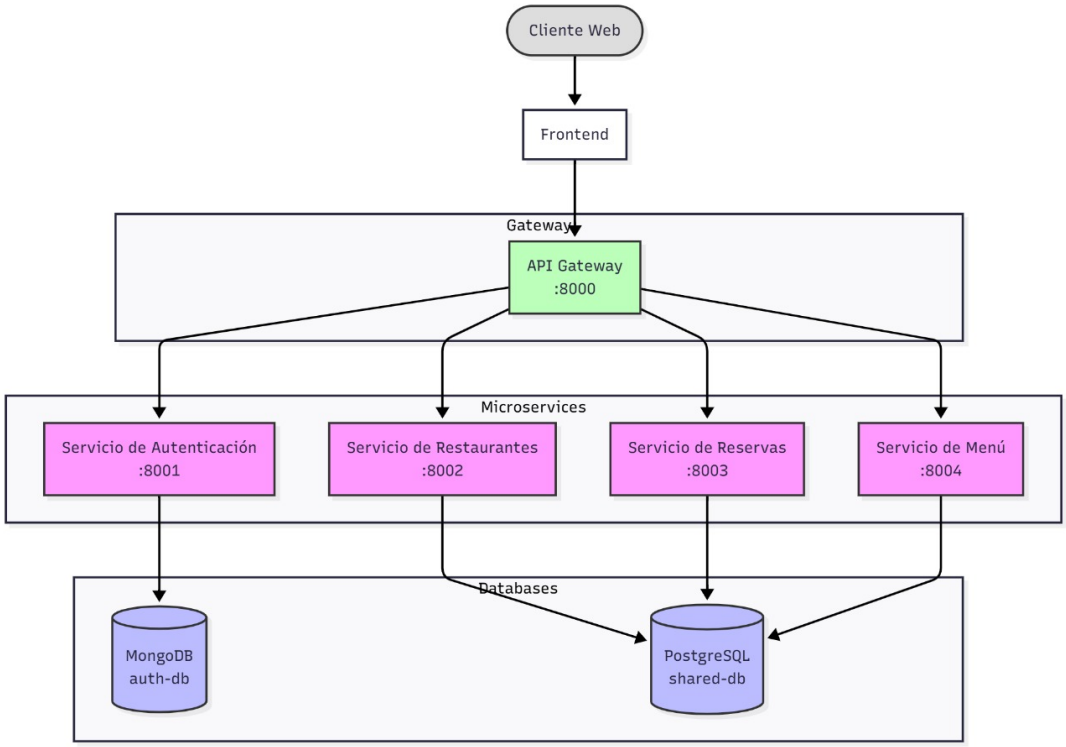
Análisis de la problemática	MPDRO-1	MesaNova Plataforma de reservas online	software	Harold Eduardo Piedrahita Valencia	Sprint 1 Análisis y Diseño

Fuente: https://miremington-team-z1hscc9a.atlassian.net/jira/software/projects/MPDRO/list?groupBy=customfield_10020&jql=project%20%3D%20%22MPDRO%22%20ORDER%20BY%20created%20DESC

Arquitectura:

El sistema se desarrolló bajo una arquitectura de microservicios garantizando modularidad, escalabilidad y mantenimiento. Cada componente del sistema fue definido como microservicio independiente, encargado de una función específicas: reserva, menú, autorización y restaurante (Ortega Candel J. M., 2023).

Figura 2. Arquitectura del proyecto.



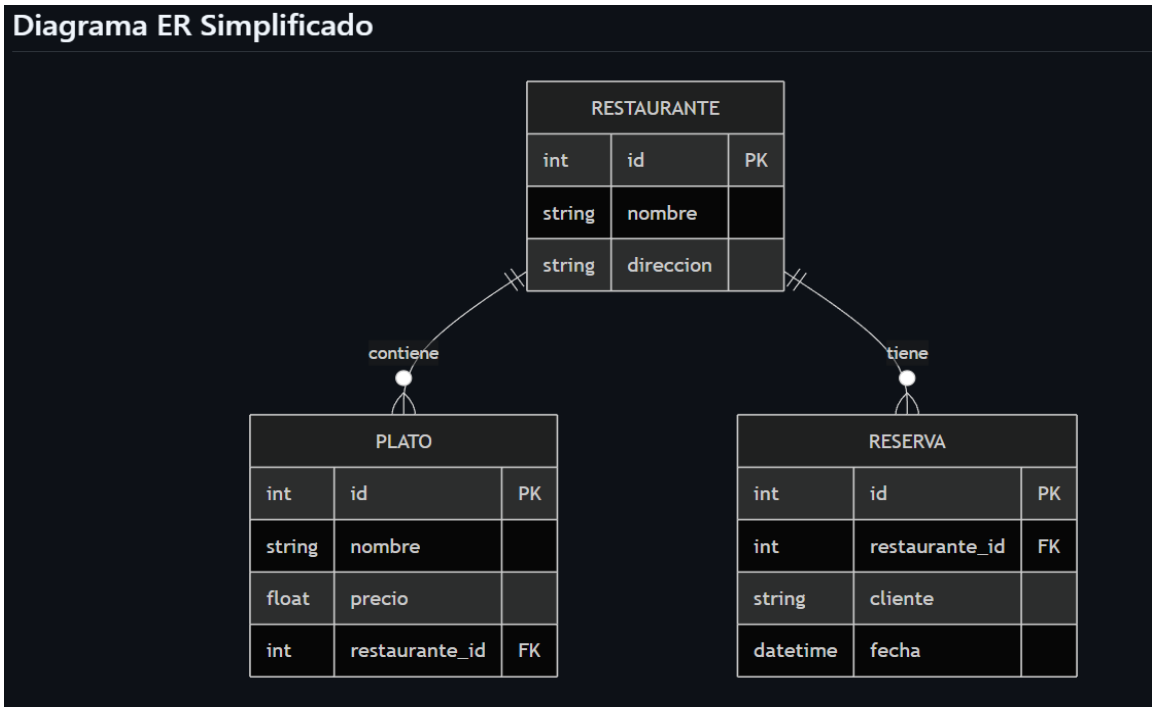
Fuente: Sebastian Ramirez, Harold Piedrahita

Implementación:

La implementación se realizó utilizando Python como lenguaje principal, Flask y Docker para la creación de contenedores, Esto Permitiendo ejecutar los servicios de manera independiente. Además, se utilizó GitHub como repositorio, gestionando el control de versiones y mantener la trazabilidad del progreso del equipo (Ortega Candel J. M., 2023).

La base de datos PostgreSQL se integró para manejar información estructurada (menú, usuarios, reservas) también se aplicaron principios de desarrollo para la comunicación entre los microservicios mediante solicitudes HTTP.

Figura 3. Diagrama entidad relación de la BD.



Fuente: Sebastian Ramirez, Harold Piedrahita

Sprint 1 – Análisis y Diseño

Ítems incluidos:

- MPDRO-1 – Análisis de la problemática
- MPDRO-2 – Diseño del software+
- MPDRO-3 – Toma de decisiones

Descripción del sprint:

Se llevó a cabo el análisis detallado del problema, la definición del diseño general de la plataforma y la toma de decisiones técnicas clave para establecer la arquitectura y los lineamientos iniciales del proyecto.

Sprint 2 – Contenedores y Orquestación

Ítems incluidos:

- MPDRO-4 – Creación de contenedores por cada servicio
- MPDRO-5 – Creación de contenedores para las BD
- MPDRO-6 – Configuración para la orquestación y levantamiento automático

Descripción del sprint:

Se construyeron los contenedores de los servicios y sus bases de datos, y se configuró la orquestación necesaria para levantar automáticamente toda la infraestructura del sistema.

Código fuente utilizado para creación de Docker File por medio del cual se inician los contenedores.

Figura 4. Inicialización de contenedores.

```
harold@Ingeniero:~/proyectos/reservas-restaurantes$ # MPDRO-4: Contenedor por servicio
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8001
CMD ["python", "main.py"]
```

Fuente: Sebastian Ramirez, Harold Piedrahita

Figura 5. inicialización de contenedores.

```
harold@Ingeniero:~/proyectos/reservas-restaurantes$ docker-compose up -d
Creating network "reservas-restaurantes_default" with the default driver
Pulling db-init (python:3.9-slim)...
3.9-slim: Pulling from library/python
38513bd72563: Already exists
b3ec39b36ae8: Pull complete
fc7443084902: Pull complete
ea56f685404a: Pull complete
Digest: sha256:2d97f6910b16bd338d3060f261f53f144965f755599aab1acda1e13cf1731b1b
Status: Downloaded newer image for python:3.9-slim
Creating postgres ... done
Creating mongodb  ... done
Creating auth-service ... done
Creating reservas-service     ... done
Creating menu-service         ... done
Creating restaurantes-service ... done
Creating db-init              ... done
Creating api-gateway          ... done
Creating frontend             ... done
```

Fuente: Sebastian Ramirez, Harold Piedrahita

Sprint 3 – Desarrollo de Microservicios

Ítems incluidos:

- MPDRO-7 – Creación de los servicios (reservas, restaurante, menú)
- MPDRO-8 – Creación del servicio de autenticación
- MPDRO-9 – Pruebas automáticas

Descripción del sprint:

Se desarrollaron los microservicios principales (reservas, autenticación y servicios del restaurante), junto con pruebas automáticas para validar su funcionamiento e integridad.

Código fuente utilizado para creación de reservas.

```
@app.post("/reservas/", response_model=ReservaRead, tags=["Reservas"])
def crear_reserva(reserva: ReservaCreate, db: Session = Depends(get_db)):

   if reserva.fecha_reserva < datetime.now():
        raise HTTPException(status_code=400, detail="La fecha de reserva no
puede ser en el pasado")

   reservas_existentes = db.query(Reserva).filter(
        and_(
            Reserva.restaurante_id == reserva.restaurante_id,
            Reserva.fecha_reserva.between(
                reserva.fecha_reserva - timedelta(hours=2),
                reserva.fecha_reserva + timedelta(hours=2)
            ),
            Reserva.estado != "cancelada"
        )
    ).count()

    if reservas_existentes >= 3:
         raise HTTPException(status_code=400, detail="No hay disponibilidad
en ese horario")

    nueva_reserva = Reserva(**reserva.dict())
    db.add(nueva_reserva)
    try:
        db.commit()
        db.refresh(nueva_reserva)
        return nueva_reserva
    except Exception as e:
        db.rollback()
         raise HTTPException(status_code=400, detail=str(e))
```

Código fuente utilizado para creación de menú:

```python
@app.post("/platos/", response_model=PlatoRead, tags=["Platos"])
def crear_plato(plato: PlatoCreate, db: Session = Depends(get_db)):
    """
    Crear un nuevo plato en el menu.

    - **nombre**: Nombre del plato (requerido)
    - **descripcion**: Descripción detallada del plato
    - **precio**: Precio del plato (requerido, > 0)
    - **categoria**: Categoría del plato (ej. Entrada, Plato Principal)
    - **disponible**: Si el plato está disponible en el menú
    - **restaurante_id**: ID del restaurante al que pertenece el plato
    """
    nuevo_plato = Plato(**plato.dict())
    db.add(nuevo_plato)
    try:
        db.commit()
        db.refresh(nuevo_plato)
        return nuevo_plato
    except Exception as e:
        db.rollback()
        raise HTTPException(status_code=400, detail=str(e))
```

Código fuente utilizado para creación de restaurantes:

```python
@app.get("/", tags=["root"])
def read_root():
    return {"message": "Servicio de restaurantes en funcionamiento."}


@app.get("/health", tags=["root"])
def health_check():
    return {"status": "ok"}

@app.post("/restaurantes/", response_model=dict, tags=["restaurantes"])
def create_restaurante(rest: models.RestauranteCreate, db: Session =
Depends(get_db)):

    payload = rest.dict()
    # Extraer campos relacionados con la creacion del usuario
    owner_password = payload.pop("owner_password", None)
    create_owner = payload.pop("create_owner", False)

    # Asegurarse de pasar solo campos que existen en el modelo DB
    db_fields = {k: v for k, v in payload.items() if k in {
        "nombre", "direccion", "telefono", "capacidad", "tipo_cocina",
"horario", "activo", "owner_email"
    }}

    new = models.Restaurante(**db_fields)
    db.add(new)
    db.commit()
```

```
    db.refresh(new)
```

Sprint 4 – API Gateway

ítems incluidos:

- MPDRO-10 – Pruebas automáticas (segunda iteración)
- MPDRO-11 – Configuración del API
- MPDRO-12 – Configuración de esta con sus endpoints

Descripción del sprint:

Se configuró el API Gateway para centralizar las rutas del sistema, integrando los microservicios existentes y estableciendo los endpoints necesarios para el consumo desde el frontend.

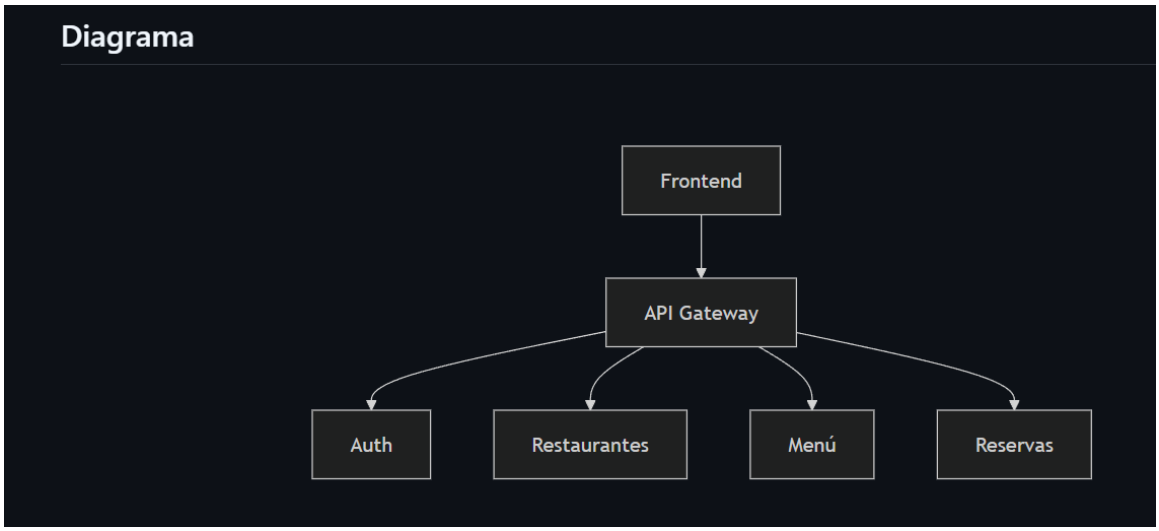
Configuración del APIGateway:

```
SERVICES = {
    "auth": os.getenv("AUTH_SERVICE_URL", "http://auth-service:8004"),
    "restaurantes": os.getenv("RESTAURANTES_SERVICE_URL",
"http://restaurantes-service:8001"),
    "reservas": os.getenv("RESERVAS_SERVICE_URL", "http://reservas-
service:8003"),
    "menu": os.getenv("MENU_SERVICE_URL", "http://menu-service:8002"),
}

app = FastAPI(title="API Gateway")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Figura 6. Diagrama del ApiGateway.



Fuente: Sebastian Ramirez, Harold Piedrahita

Sprint 5 – API Gateway

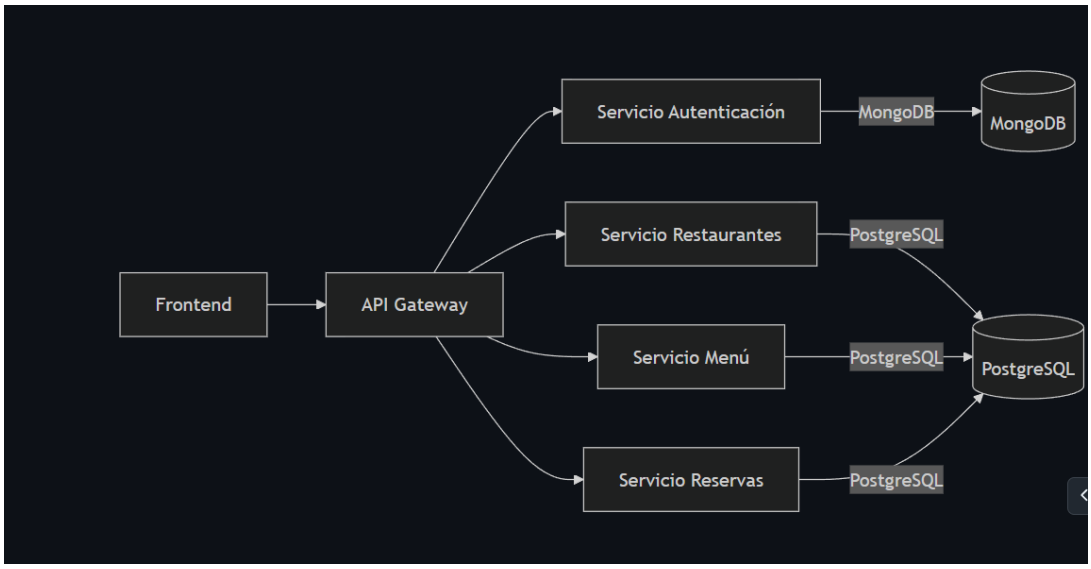
ítems incluidos:

- MPDRO-13 – Pruebas automáticas
- MPDRO-14 – Desarrollo de vistas para cada servicio
- MPDRO-15 – Pruebas de funcionalidad manuales con endpoints
- MPDRO-16 – Mejoras de las vistas: aumento de botones y funcionalidades

Descripción del sprint:

Se desarrollaron las interfaces de usuario, se realizaron pruebas manuales y automáticas, y se aplicaron mejoras visuales y funcionales a las distintas vistas para optimizar la experiencia del usuario.

Figura 7. Diagrama del Frontend.



Fuente: Sebastian Ramirez, Harold Piedrahita

Figura 8. Vista principal.

Fuente: Sebastian Ramirez, Harold Piedrahita

Sprint 6 – Documentación y Despliegue

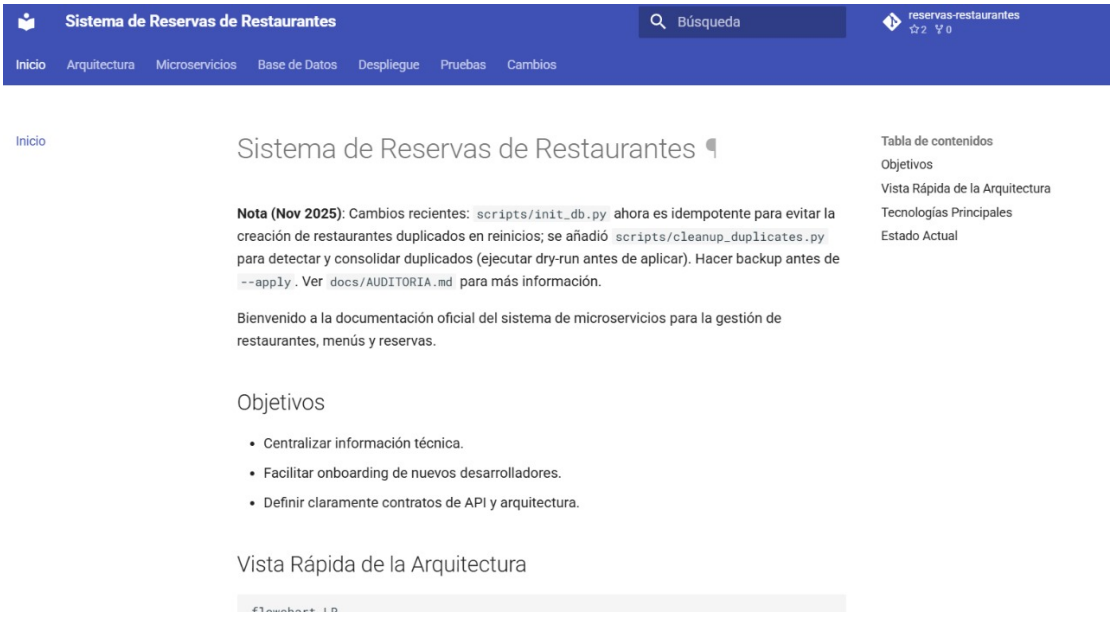
ítems incluidos:

- MPDRO-17 – Actualización de README
- MPDRO-18 – Documentación
- MPDRO-19 – Creación de documentos con MkDocs y diagramas en Mermaid
- MPDRO-20 – Documentación automática
- MPDRO-21 – Despliegue en GitHub Pages

Descripción del sprint:

Se completó toda la documentación del proyecto, se generaron manuales y diagramas con herramientas profesionales, y finalmente se realizó el despliegue público del sitio mediante GitHub Pages.

Figura 8. Vista principal.



Fuente: Sebastian Ramirez, Harold Piedrahita

Pruebas:

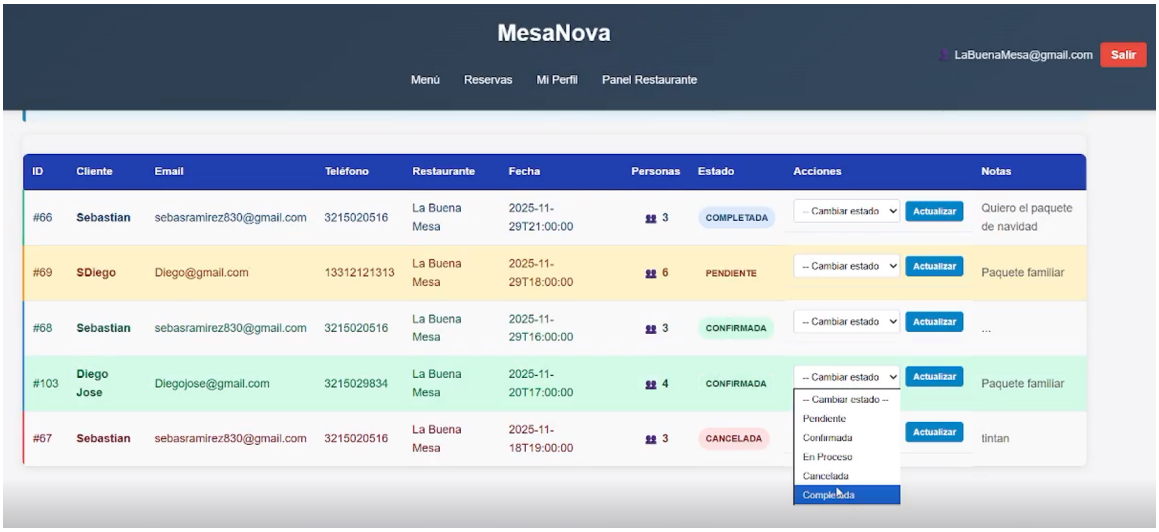
Las pruebas de desarrollo hacen parte del proceso sistemático para evaluar la calidad de un software, el propósito es encontrar errores y defectos en el software (R. S, 2010). Ya que se identificó la necesidad de encontrar fallas antes de la entrega de proyectos. (Mubarak Albarka , 2019). Es así como esta actividad reduce los riegos, gracias a la detección de defectos, lo cual mejora la calidad del software (Morgan, 2010).

Durante el proceso de desarrollo nos permitió pruebas con Pytest, la finalidad es validar la disponibilidad, estabilidad y correcto funcionamiento de los microservicios que componen la plataforma, las cuales nos permiten prevenir futuros fallos y mejorar la calidad del software (Martinez, 8 nov 2024).

Pruebas Manuales:

Durante el desarrollo de la plataforma se ejecutaron de forma continua pruebas manuales para verificar su funcionalidad incorporada.

Figura 10. Pruebas manuales.



Fuente: Sebastian Ramirez, Harold Piedrahita

Pruebas Automatizadas con Pytest:

Además de las pruebas manuales se implementaron pruebas automatizadas con Pytest, las cuales ayudaron a validar el funcionamiento interno de la plataforma.

Ejecución de las pruebas con Pytest desde el host:

Activar contenedores:

```
docker compose up -d
```

Ejecutar las pruebas con Pytest:

```
pytest tests/test_auth.py -v
```

Ejecución de Pytest dentro del contenedor:

Ejecución de Pytest dentro del contenedor Docker de los microservicios:

```
docker run --rm --network reservas-restaurantes_default \
  -v "$(pwd)":/app -w /app \
  --entrypoint bash \
  reservas-restaurantes-auth-service:latest \
  -c "pip install -q pytest requests && PYTHONPATH=/app
AUTH_URL=http://auth-service:8004 pytest -v
tests/test_auth.py"
```

Para esta función Pytest utiliza la variable de entorno:

```
AUTH_URL=http://auth-service:8004
```

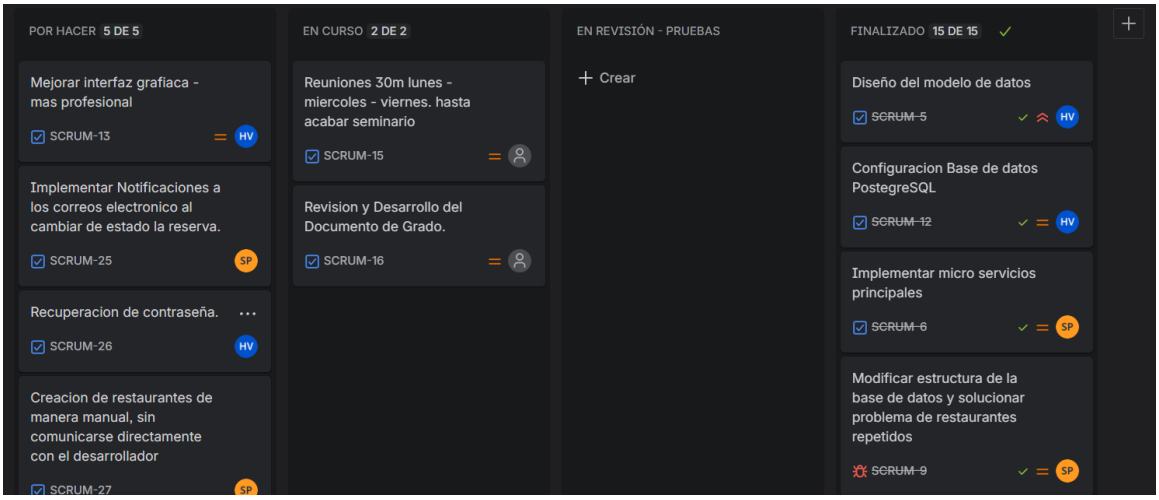
El uso de Pytest permitió automatizar y estandarizar la validación de los microservicios de autenticación, aceleración de ciclos de desarrollo y asegurando estabilidad funcional, combinando cada una de las pruebas garantizo una validación completa del sistema (Olivares, 2018).

Resultados Obtenidos:

El proyecto consistió en el desarrollo de una plataforma para reserva de restaurantes basada en una arquitectura de microservicios, se emplearon tecnologías como Python, Flask, MongoDB y Docker, y se integró una base de datos PostgreSQL para el manejo de información.

La aplicación de la metodología Scrumban facilito la gestión del flujo de trabajo mediante tableros interactivos Kanban, Priorizando el avance continuo de los microservicios (autenticación, restaurantes, reservas y menú) logrando un sistema modular, escalable y fácil de mantener (Reddy, 2015).

Figura 11. Manejo de avances con Jira.



Fuente: Sebastian Ramirez, Harold Piedrahita

Así mismo se implementaron pruebas manuales que permitieron ver en tiempo real el flujo y funcionalidad de la plataforma, y posteriormente se ejecutaron pruebas automatizadas con Pytest validaron el comportamiento interno de los endpoints y comunicaciones HTTP. Las pruebas se ejecutaron desde el host y dentro de los contenedores Docker (Olivares, 2018).

Como resultado, se obtuvo una plataforma estable y lista para el despliegue, con un diseño modular lo que facilitara futuras mejoras.

Figura 8. Vista principal.

Fuente: Sebastian Ramirez, Harold Piedrahita

Conclusiones.

El desarrollo de la plataforma ***MesaNova*** nos ha permitido entender y aprender sobre la adopción de las arquitecturas basadas en microservicios las cuales son estrategias eficaces para optimizar los procesos en este caso de reservas en restaurantes, de acuerdo con lo evidenciado en el desarrollo de este proyecto, esta arquitectura favorece la modularidad, escalabilidad y mantenimiento del sistema, aspectos claves que coinciden con lo propuesto por Ortega Candel (2020. 2023) en relación con las ventajas de dividir una solución en componente independientes.

La implementación de tecnologías como Python, Flask, Docker y PostgreSQL nos han permitido construir servicios aislados por contenedores orquestados, capaces de gestionar toda la información crítica de los usuarios en sus reservas por cada restaurante con su respectivo menú y también autenticar el ingreso de manera segura y fiable. De igual forma, la utilización de GitHub garantizo el adecuado control de versiones y una trazabilidad clara en el avance del proyecto y el trabajo en equipo, desde un punto de vista metodológico, esta aplicación facilita el flujo de trabajo continuo y organizado.

El proceso de pruebas tanto manuales como automatizadas con Pytest contribuyeron a garantizar que cada uno de los microservicios mantuviera de manera estable su funcionalidad, gracias a dichas pruebas se lograron identificar errores oportunamente, realizar validaciones de respuesta en los diferentes endpoints y asegurar que la comunicación entre cada uno de los microservicios funcionara de manera coherente, también estos resultados nos permitieron mostrar que la digitalización de la gestión de reservas tiene un impacto directo en la reducción de errores operativos, nos permite

optimizar tiempos y agilizar procesos pero sobre todo mejora la experiencia del cliente, además el enfoque modular adoptado deja una sólida base para futuras extensiones que se puedan implementar en analítica de datos, personalización de servicios o integración de otros aplicativos externos.

Respecto a la pregunta orientadora, podemos decir que esta plataforma construida bajo microservicios orquestados **si optimiza de manera comprobable** los procesos de reservas en restaurantes, contribuye a la transformación digital de el sector gastronómico y porque no incluso el sector hotelero y expande las posibilidades con soluciones mucho más flexibles, seguras, agiles y escalables. Este proyecto evidencia que el enfoque no solo es pertinente, sino que aporta lo necesario desde un contexto donde los establecimientos buscan una mayor eficiencia en sus procesos y mejores experiencias para sus clientes.

Bibliografía

Zea Ordóñez, M. P., Molina Ríos, J. R., & Redrován Castillo, F. F. (2017). ADMINISTRACIÓN DE BASES DE DATOS CON POSTGRESQL. En M. P. Zea Ordóñez, J. R. Molina Ríos, & F. F. Redrován Castillo, *ADMINISTRACIÓN DE BASES DE DATOS CON POSTGRESQL* (pág. 82). 3Ciencias.

Cabezas Benitez, E. M. (14 de Octubre de 2025). *Repositorio Digital EPN*. Obtenido de Repositorio Digital EPN: https://bibdigital.epn.edu.ec/handle/15000/26873

Guanilo Pareja, C. G., Barragán Codina, J. N., & Guerra Rodríguez, P. (2024). *Incidencia de las plataformas digitales en la fidelización de cliente en restaurantes de Lima.* Lima: Vinculatégica EFAN.

Hernandez Acosta, L. M. (2024). *ULPGC.* Obtenido de ULPGC: https://accedacris.ulpgc.es/jspui/handle/10553/130056

Hernández Trujillo , O., & Gil Gómez, I. (2022). *Arte, cultura y nuevas tecnologías en América Latina y el Caribe: Economía Creativa. Gastronomía. Tendencias y estrategias digitales.* Ciudad de México (la coordinación general es con base en CDMX): Banco Interamericano de Desarrollo (BID).

Jiménez, R., Mazo;. (2022). *Universidad Politécnica de Madrid (UPM)*. Obtenido de Universidad Politécnica de Madrid (UPM): https://oa.upm.es/71582/

Martinez, J. (8 nov 2024). Curso de Programación Básica Aprende Python Paso a Paso. En J. Martinez, *Curso de Programación Básica Aprende Python Paso a Paso* (pág. 388). Independently published.

Morgan, P. (2010). *Software Testing.* Swindon, Reino Unido: British Informatics Society Limited (BISL) / BCS, The Chartered Institute for IT.

Mubarak Albarka , U. (2019). *Comprehensive Study of Software Testing: Categories, Levels, Techniques, and Types.* IJARIIT.

Olivares, B. (2018). *Pytest quick start guide : write better Python code with simple and maintainable tests.* Birmingham, Reino Unido: Packt Publishing, Birmingham, UK, .

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

Ortega Candel, J. M. (2023). *Desarrollo de microservicios con Python.* Marcombo.

R. S, P. (2010). *Software Engineering: A Practitioner's Approach (7th).* McGraw-Hill Education.

Reddy, A. (2015). *The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban.*