TRABAJO DE GRADO
Opción Seminario-Diplomado

Eventia: Plataforma Integral de Gestión de Eventos Corporativos

Corporación Universitaria Remington
Facultad de Ingenierías
Ingeniería de Sistemas

Angie Gisell Vargas Solorzano
Tutor: Diego Fernando Marín Lozano

Opción de Trabajo de grado Seminario-Diplomado
2025

Dedicatoria

Dedico este trabajo a mi familia, por su apoyo incondicional, sus palabras de ánimo y la confianza que siempre han depositado en mí.

También a quienes con paciencia, amor y motivación me impulsaron a continuar, incluso en los momentos más exigentes del proceso académico.

De igual manera, extiendo esta dedicatoria a todas las personas que creen en la educación como un camino de transformación personal y profesional, y que inspiran a nunca detenerse en la búsqueda del conocimiento y la mejora continua.

Agradecimientos

En primer lugar, agradezco a papito Dios por la fortaleza y la sabiduría otorgada durante este proceso, a mi familia por su apoyo constante, y a la Corporación Universitaria Remington por los espacios de formación que han contribuido a mi crecimiento académico y profesional.

También extiendo mi reconocimiento a los docentes que, con su guía y conocimientos, hicieron posible el desarrollo de este trabajo.

Tabla de Contenido

Resumen

Este trabajo de grado se basa en el desarrollo de una plataforma web que permite a los usuarios efectuar gestiones completas de eventos corporativos de manera organizada y optimizada. El objetivo de esta es lograr reducir tiempos, permitir una interacción entre usuarios, proveedores y asistentes, así como también tener un control en cuanto a asistencia, costo, disponibilidad y estadísticas de eventos.

Estamos en la era de la transformación digital donde los cambios en cada uno de los espacios de la vida cotidiana, en los que, desenvolvemos crecen a pasos agigantados, lo que nos obliga también a estar actualizados en materia digital y practica para la organización de eventos.

Mediante los requerimientos planteados para el desarrollo de esta herramienta, los problemas más frecuentes que se lograron identificar se centran en la baja eficiencia al llevar a cabo los diferentes eventos corporativos, así como también la duplicidad de información en cuanto a los registros de ingreso y disponibilidad de espacios, por otra parte también nos encontramos con la limitada trazabilidad y la falta de estadística que generan los procesos manuales o herramientas aisladas.

La metodología utilizada tiene un enfoque cualitativo, basado en la revisión documental de literatura académica y científica, apoyada en bases de datos y efectuando el análisis reflexivo de las buenas prácticas de transformación digital y gestión de software. Este enfoque permite sustentar teóricamente la pertinencia del desarrollo incremental y su aplicación en plataformas corporativas.

Por otra parte en los resultados esperados se destacan, la identificación de estrategias metodológicas que, permiten fortalecer la trazabilidad, la centralización de datos y la eficiencia operativa, incorporando progresivamente los módulos de autenticación, gestión de eventos, control de proveedores e inscripciones, lo cual proyecta un impacto académico práctico, teniendo en cuenta el aporte al conocimiento sobre metodologías de desarrollo y al mismo tiempo ofreciendo una ruta viable para la digitalización del sector empresarial enfocado en eventos corporativos.

Para concluir, se le atribuye a la reflexión que, el desarrollo incremental favorece aspectos como la adaptabilidad, control del riesgo y la mejora constante, convirtiéndose en una estrategia eficaz y eficiente para el diseño e implementación de plataformas tecnológicas que responden a las necesidades actuales de la era de transformación digital.

Palabras clave

Gestión de Eventos, Transformación Digital, Microservicios, Python, FastAPI, API Gateway, Contenedores, MongoDB.

Pregunta orientadora de la búsqueda

¿Cómo implementar una plataforma web para la gestión integral de eventos corporativos utilizando una metodología de desarrollo incremental que permita digitalizar, optimizar y automatizar los procesos logísticos, operativos y administrativos, permitiendo el control de cada una de los actores en la misma?

Sustentación teórica de la pregunta

Ingeniería de Software.

La ingeniería de software puede entenderse como un proceso compuesto por varias capas que trabajan juntas y siempre van orientadas a garantizar la calidad. En la base está la capa de procesos, que permite organizar y controlar la gestión de proyecto; luego está la capa de métodos, que, aportan la experiencia técnica necesaria para construir el software de manera adecuada; por último, la capa herramientas brinda apoyo automático o semiautomático para facilitar los procesos y los métodos utilizados. (Roger S, 2005).

Ciclo de Vida del Software

El ciclo de vida del software es todo el proceso que estructura y sigue el proceso desde que inicia su desarrollo hasta que se pone en funcionamiento y recibe el mantenimiento necesario para seguir operando correctamente.(Sommerville & Ian, 2011).

Fases del Ciclo de Vida del Software

- **Fases:** Una fase es un conjunto de actividades las cuales buscan lograr un objetivo dentro del desarrollo del proyecto, estas se forman agrupando tareas más pequeñas que comparten un mismo periodo dentro del ciclo de vida del proyecto. Al organizar estas tareas en una misma etapa, también se establecen requisitos de tiempo, así como asignar recursos, donde encontramos personal, presupuesto o materiales.

- **Análisis:** En el análisis el cliente manifiesta sus necesidades y que problemas debe resolver el sistema, una vez recopilada esta información se estudia para entender y conforme a este análisis, se elabora la especificación del sistema que se va a desarrollar. (Lis, 2021)

- **Diseño:** La fase del diseño crea la estructura del software tomando como base las especificaciones definidas en el análisis previo. El resultado es un documento principalmente gráfico donde se muestran los componentes del sistema y la manera en que estarán organizados. (Lis, 2021)

- **Codificación:** Esta etapa utiliza las herramientas necesarias para construir los componentes definidos en el diseño. Además se desarrollan los elementos que

permitirán realizar pruebas, con el fin de verificar que cada parte funcione correctamente. (Lis, 2021)

- **Integración:** Cuando todos los componentes del sistema están listos, se unen para formar el sistema completo, en esta fase también se realizan pruebas para asegurarse del funcionamiento integrado de manera adecuada. (Lis, 2021)

- **Explotación:** No es directamente del ciclo de la vida del software, sin embargo, está relacionada porque corresponde al momento en el que el sistema ya está funcionando y siendo utilizado, El desempeño durante esta fase influye en lo que ocurrirá. (Lis, 2021)

- **Mantenimiento:** Durante la explotación, pueden surgir necesidades de cambios, ya sea para corregir errores que no se detectaron antes o para añadir mejoras y nuevas funcionalidades. En esta fase revisa la evolución del sistema para adaptarlo a las necesidades cambiantes de los usuarios. (Lis, 2021)

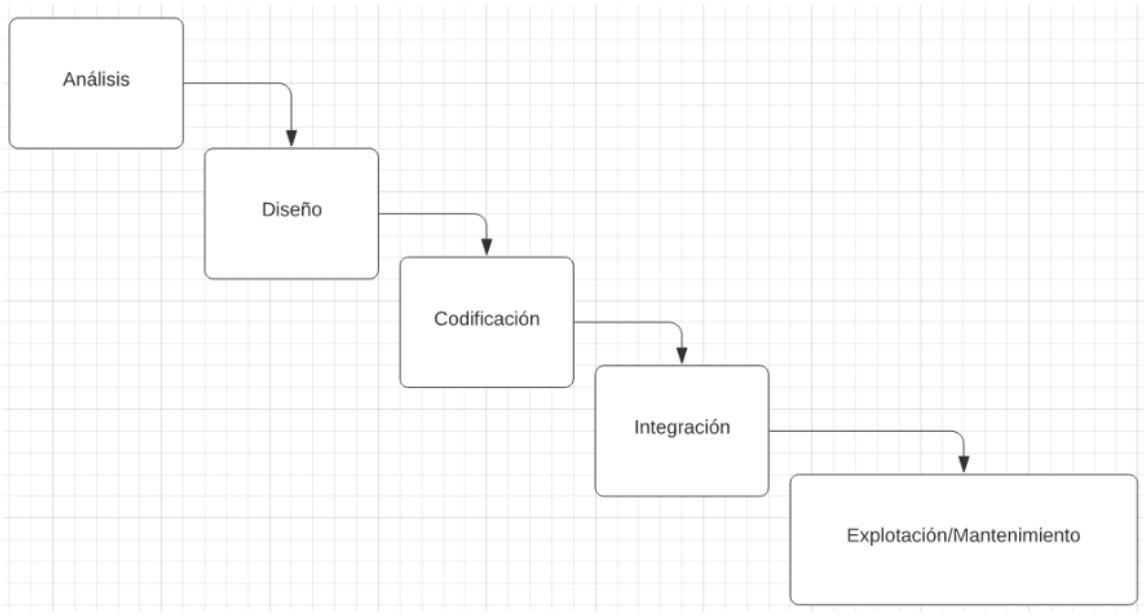


Figure 1 Ciclo de vida del Software (Lis, 2021)

Aplicación Web

Las aplicaciones web son un tipo de software basado en redes que abarca una gran variedad de soluciones. Estas No solo ofrecen funciones específicas, herramientas y contenido para el usuario final, sino que también pueden integrarse con bases de datos empresariales y sistemas de negocio más amplios. (Pressman & Maxim, 2020).

Desarrollo Ágil

El desarrollo ágil se basa en priorizar a las personas y su comunicación por encima de los procesos rígidos y las herramientas. También valora más un software que funcione que una documentación extensa, así como la colaboración constante con el cliente en lugar de centrarse únicamente en contratos. Tiene como principio fundamental adaptarse al cambio en vez de seguir un plan estrictamente. Aunque los aspectos mencionados en segundo lugar también son importantes, los métodos ágiles dan mayor peso a los primeros. Este enfoque nace para enfrentar las limitaciones del desarrollo tradicional de software. El ágil ofrece beneficios significativos, pero no es adecuado para cualquier tipo de proyecto, producto, equipo o contexto. (Pressman & Maxim, 2020).

HTTP

HTTP, de sus siglas en inglés: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. Así, una página web completa resulta de la unión de distintos sub-documentos recibidos, como, por ejemplo: un documento que especifique el estilo de maquetación de la página web (CSS), el texto, las imágenes, vídeos, scripts, etc... (Lis, 2021)

Métodos de Petición HTTP

Como se puede leer en el artículo de Mozilla, HTTP, define una serie de métodos para indicar la acción que se desea realizar para un recurso en concreto, los principales métodos de petición HTTP, son:

- **GET:** Solicita una representación del recurso, las peticiones get sólo recuperan datos.
- **POST:** Se envía una entidad a un recurso, por lo general causando un cambio de estado o efectos secundarios en el servidor.
- **PUT:** Reemplaza las representaciones actuales del recurso con el contenido de la petición.
- **DELETE:** Se usa para borrar un recurso especificado. (Lis, 2021)

API Gateway

Es un servicio de fuerte gestión y control, centrado en la API y centralizado en serie, que aparece en el límite del sistema. Antes de la popularidad del concepto de microservicios, nació la entidad de Pasarela de API. El principal escenario de aplicación en este momento era Open API, que es una plataforma abierta para socios externos. Uno de los beneficios de API Gateway es encapsular la estructura interna de la aplicación. En comparación con llamar al servicio especificado, el cliente interactúa con la pasarela de forma más sencilla.

API Gateway proporciona a cada cliente una API específica, lo que reduce el número de comunicaciones cliente-servidor y simplifica el código del cliente (Zhao, Jing, & Jiang, 2018).

Docker

Docker es la tecnología de empaquetamiento y ejecución que resuelve el problema de la "deuda de dependencia" y garantiza la coherencia del entorno en un sistema de microservicios (PHILIPPE, 2018).

- **Definición Operacional:** Docker es la herramienta que te permite crear imágenes (paquetes inmutables que contienen un microservicio y todo lo que necesitas para ejecutarse) y correrlas como contenedores aislados (PHILIPPE, 2018).

- **Rol en Microservicios:** Es la base para asegurar que cada microservicio (el de usuarios, el de pedidos, el de pagos) pueda ser desarrollado, probado y desplegado de forma independiente sin afectar a los demás (PHILIPPE, 2018).

- **Dockerización (El Proceso Práctico)**
- Dentro de la arquitectura de microservicios, la Dockerización es el paso fundamental para transformar la lógica de la aplicación en unidades desplegables (PHILIPPE, 2018).

MongoDB

Es un sistema de gestión de bases de datos (SGBD) NoSQL de código abierto que utiliza un modelo orientado a documentos.

- **Modelo de Datos:** Almacena los datos en estructuras flexibles tipo BSON (Binary JSON), conocidas documentos como.

- **Estructura:** Los documentos están agrupados en colecciones (análogo a las tablas en SQL).

- **Clave de característica:** Ofrece un esquema flexible (sin esquema), lo que permite que los documentos dentro de una misma colección tengan estructuras de campos diferentes. Esto facilita el desarrollo rápido y el manejo de datos semiestructurados.

- **Propósito:** Está diseñado para ser altamente escalable horizontalmente (mediante sharding), priorizando la disponibilidad y la flexibilidad sobre la rigidez de la consistencia relacional estricta. (Boicea, Radulescu, & Agapin, 2012).

Frontend: Es la parte visual de la plataforma con la que el usuario interactúa directamente, para este proyecto se utilizo la herramienta V0 de Vercel, la cual a través de IA, proporciono el frontend para Eventia.

GitHub: GitHub es una plataforma basada en la nube donde puedes almacenar, compartir y trabajar junto con otros usuarios para escribir código (GitHub, s.f.).

CRUD: (Create, Read, Update, Delete) es un acrónimo para las maneras en las que se puede operar sobre información almacenada. Es un nemónico para las cuatro funciones del almacenamiento persistente. CRUD usualmente se refiere a operaciones llevadas a cabo en una base de datos o un almacén de datos, pero también pude aplicar a funciones de un nivel superior de una aplicación como soft deletes donde la información no es realmente eliminada, sino es marcada como eliminada a través de un estatus (Lis, 2021).

Problema

En los últimos años, los eventos empresariales dentro del marketing relacional han cambiado de forma notable, tanto en su esencia como en la manera en que se gestionan los procesos de comunicación antes, durante y después del evento. Hoy en día, además de la estrategia tradicional de marketing y el trabajo con los stake-holders mediante acciones relacionales, la era digital en su continuo avance, ha hecho que los social media se conviertan en herramientas fundamentales. Por lo tanto estas plataformas permiten conectar e interactuar con los grupos de interés de las organizaciones por medio de la comunicación 2.0 y las redes sociales, fortaleciendo así las relaciones y la visibilidad empresarial. (Campillo, Irene, & Castelló-Martínez, 2014)

Este cambio amplía notablemente el alcance de las estrategias de comunicación y de relacionamiento que las empresas pueden aplicar en sus eventos corporativos. Además, nos lleva a reconsiderar la manera en que se debe gestionar la “marca-acontecimiento” y el tipo de profesional encargado de estos eventos, quien ahora debe asumir funciones propias de los nuevos perfiles digitales dentro de la comunicación online. Esto implica que la gestión del evento cambia, basándose en una estrategia de comunicación 2.0 continua, o que, por el contrario, el uso de estas herramientas se limite únicamente al momento del evento. Analizar este panorama nos permitirá establecer algunas pautas esenciales para manejar de forma profesional la estrategia de marca en los eventos empresariales. (Campillo, Irene, & Castelló-Martínez, 2014)

Las tecnologías disruptivas generan economías disruptivas, lo cual obliga al mundo a replantearse sus actividades en general, el área de Marketing no es ajeno a esta realidad dado que tienen que analizar las dificultades y desafíos del Marketing cambiante e identificar soluciones tecnológicas que les permitan mantener la visión sobre los nuevos mercados, agregando valor e innovando a los productos o servicios, desarrollados a medida y en función de las expectativas de clientes, cada vez más exigentes y cambiantes. Las empresas en general requieren almacenar todos los datos, lo cual representa grandes volúmenes de información, los mismos que se irán incorporando poco a poco en sus gestiones operativas, por lo cual debe estar disponibles en cualquier momento. (Vega & y Gonzalez, 2018)

Metodología

La metodología de este proyecto se fundamenta en la necesidad de las organizaciones de evolucionar sus procesos de gestión de eventos, pasando de esquemas tradicionales a soluciones digitales e integradas. Tradicionalmente, la organización de eventos se ha enfrentado a desafíos logísticos significativos relacionados con la coordinación de actividades y la disponibilidad de información. La fragmentación de tareas y datos entre diferentes áreas o herramientas que puede generar ineficiencia y altos costos operativos (Mendoza, Camacho, Mendoza, & Mendoza, 2024)

La implementación de plataformas web modulares es esencial para optimizar estos procesos, ya que permiten la centralización de la información y la automatización de tareas. Esto garantiza la eficiencia, una adecuada trazabilidad de las acciones y la escalabilidad del sistema ante el crecimiento de la demanda (Ramos, 2025)

Metodología de Desarrollo Incremental

Sommerville lo posiciona como uno de los modelos de proceso de software genéricos y lo define por las siguientes características:

Combinación de Modelos: El modelo incremental integra elementos del desarrollo por secuencia lineal (análisis, diseño, codificación, prueba) con la filosofía iterativa o evolutiva de la construcción de prototipos.

Entrega Progresiva: La funcionalidad del sistema se entrega al cliente en incrementos (o versiones). Cada incremento se construye sobre el anterior, añadiendo nueva funcionalidad o mejorando la existente (Sommerville & Ian, 2011).

Funcionamiento del Proceso

El desarrollo incremental se basa en un ciclo de proceso que se repite para cada porción del sistema:

- **Definición del Esbozo de Requerimientos:** Se definen los requerimientos generales del sistema.

- **Asignación a Incrementos:** Los requerimientos se asignan a incrementos específicos. Se priorizan y se deciden qué funcionalidad se incluye en cada entrega.

- **Desarrollo del Incremento:** Cada incremento pasa por las etapas de **Especificación, Desarrollo y Validación** (que incluyen pruebas).

- **Entrega Operacional:** El rasgo distintivo, según Sommerville, es que al final de cada incremento se entrega un **producto completamente operativo** (aunque incompleto en funcionalidad total) que puede ser utilizado por el cliente (Sommerville & Ian, 2011).

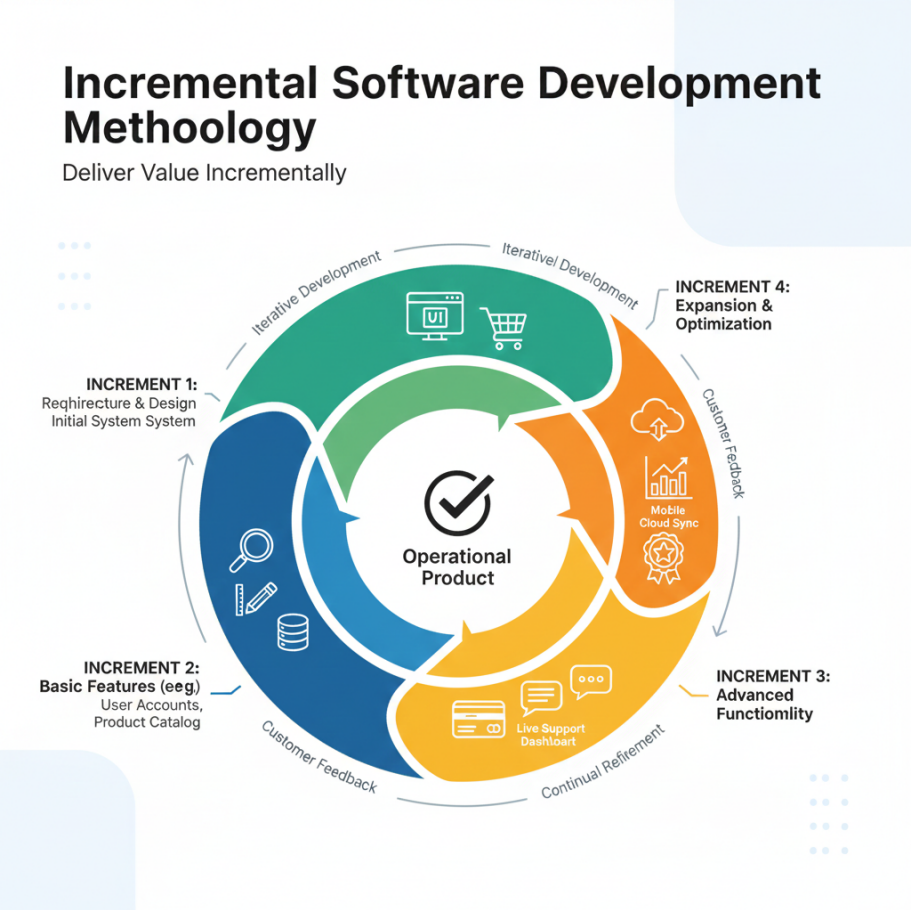


Figure 2 Incremental Methodology (Autora)

La metodología de desarrollo incremental es fundamental en proyectos que se desarrollan en entornos de requisitos cambiantes o con un alto grado de incertidumbre. Este enfoque permite segmentar el proyecto en partes más pequeñas y manejables, entregando versiones funcionales del software de manera periódica. (León, Acosta, & Díaz, 2021) El desarrollo incremental se caracteriza por construir el sistema en pequeñas porciones, donde cada incremento es una versión funcional que se construye sobre la anterior. Este modelo reduce la complejidad, permite la retroalimentación temprana del usuario y facilita la integración continua de los módulos funcionales. (Pressman & Maxim, 2020)

Cada incremento en el desarrollo de Eventia sigue un ciclo iterativo que incluye las fases de requisitos, diseño, implementación y pruebas. Esta aproximación asegura que los módulos se integran progresivamente al ecosistema de microservicios, minimizando la aparición de errores críticos en etapas tardías. (Tserkovny, 2025)

Desarrollo

El desarrollo del proyecto Eventia se organizó en tres incrementos, siguiendo un enfoque de microservicios y utilizando contenedores Docker para garantizar la portabilidad y escalabilidad.

Fases

La metodología se desarrolla mediante la repetición de las siguientes fases en cada incremento:

- **Requerimientos:** Recolección y definición de los requisitos para el incremento actual.
- **Diseño:** Creación de la arquitectura y el diseño técnico para implementar esos requisitos.
- **Implementación (Codificación):** Desarrollo del código para las funcionalidades diseñadas.
- **Pruebas:** Verificación del incremento para asegurar que cumple con los requisitos y que no introduce errores en funcionalidades anteriores.
- **Despliegue/Entrega:** Puesta en funcionamiento del incremento, agregando las nuevas funciones al sistema existente.

Descripción de Requerimientos

Los requisitos de esta Plataforma se dividen en funcionales y no funcionales

IDENTIFICACIÓN	REQUISITO FUNCIONALES	ACTOR PRINCIPAL
RF-001	El sistema debe permitir a los Administradores crear, editar y eliminar eventos.	Administrador
RF-002	El sistema debe permitir a los Usuarios registrarse en el evento y reservar su cupo.	Sistema
RF-003	El sistema debe generar y enviar un comprobante o entrada digital (ej. código QR) al usuario tras la reserva.	Sistema
RF-004	El sistema debe mostrar el aforo disponible y la cantidad de cupos reservados para cada evento.	Administrador/Sistema
RF-005	El sistema debe permitir a los Administradores ver una lista de todos los asistentes registrados para un evento.	Administrador
RF-006	El sistema debe permitir a los usuarios buscar eventos por fecha, nombre o categoría.	Sistema

IDENTIFICACIÓN	REQUISITO NO FUNCIONALES	CATEGORIA
RNF-001	La plataforma debe ser accesible y funcionar correctamente en navegadores web modernos (Chrome, Firefox, Safari, Edge).	Usabilidad
RNF-002	El sistema debe ser capaz de soportar hasta 100 reservas simultáneamente sin fallar ni ralentizarse.	rendimiento
RNF-003	Toda la información personal de los usuarios (nombres, correos) debe ser almacenada de forma segura y encriptada .	Seguridad
RNF-004	El tiempo de carga de cualquier página principal no debe exceder los 3 segundos .	rendimiento
RNF-005	La interfaz de usuario debe ser intuitiva y fácil de usar para registrarse en un evento.	Usabilidad

Incrementos para la construcción Eventia progresivamente

Incremento 1: Este incremento se enfoca en establecer los componentes lógicos principales de la aplicación, encapsulados en sus respectivos contenedores, y asegurando su comunicación inicial.

Fase	**Tareas Clave (Desarrollo)**	**Producto entregable**
Requerimientos	Definición precisa de las operaciones (CRUD) de los servicios de Eventos y Usuarios.	Contratos de los servicios (lo que hacen).
Diseño	Diseño de la arquitectura de contenedores (Dockerfiles, Docker Compose inicial). Definición de los Microservicios .	Estructura de código base para Servicio de Eventos y Servicio de Usuarios .
Implementación	Creación y codificación de la lógica interna de los servicios. Exposición de endpoints de prueba (simulando datos).	Contenedores de Microservicios funcionales , pero aún sin persistencia de datos.
Pruebas	Pruebas unitarias de la lógica de cada servicio. Verificación de la orquestación básica de contenedores (que levanten correctamente).	Servicios listos para estar conectados a un punto de entrada.
Entrega	Microservicios Core (Eventos y Usuarios) disponibles y corriendo en contenedores.	

Incremento 2: Este incremento añade la base de datos para la persistencia y el API Gateway como punto de entrada único para todos los servicios.

Fase	Tareas Clave (Desarrollo)	Producto Entregable
Requerimientos	Definición de los requerimientos de acceso y seguridad a través del Gateway. Definición del Esquema de la Base de Datos.	Requerimientos de Seguridad (RNF-003) y Requerimientos de Acceso (RF-001/RF-005).
Diseño	Diseño del Esquema de la DB. Diseño y configuración de las rutas y políticas del API Gateway.	Arquitectura completa de Back-end definida.

Incremento 3: Interfaz de Usuario (Front-end) y Funcionalidad Completa, el incremento final consume los servicios ya funcionales a través del API Gateway y se centra en la experiencia del usuario (RF-002, RF-006).

Fase	Tareas Clave (Desarrollo)	Producto Entregable
Requerimientos	Definición de la interfaz de usuario y el flujo de navegación (RNF-001, RNF-005). Definición de los flujos de reserva (RF-002).	Diseño UX/UI detallado para las vistas de usuario y administrador.
Diseño	Diseño del Front-end (estructura de componentes, diseño visual). Mapeo de las llamadas del Front-end al API Gateway.	Maquetas y prototipos de la interfaz.
Implementación	Desarrollo del Front-end (aplicación web) en su propio contenedor. Implementación de los flujos de interacción con el usuario (búsqueda, registro, etc.).	Front-end enlazado al API Gateway.
Pruebas	Pruebas de usabilidad y experiencia de usuario. Pruebas de extremo a extremo (desde la interfaz hasta la DB y viceversa).	Sistema Eventia funcional y utilizable.
Entrega	Sistema Eventia Completo, desplegado y listo para su uso, cumpliendo con los requerimientos funcionales y no funcionales (RF-001 a RF-006, RNF-001 a RNF-005).	

Incremento 1: Creación de los Primeros Microservicios y Contenedores

Este primer incremento estableció los primeros microservicios de Eventia Platform y su ejecución en un entorno contenedorizado mediante Docker. Se configuraron los servicios de Autenticación, Organizador, Proveedor y Asistente, además del API Gateway, asegurando que todos los componentes pudieran levantarse correctamente usando docker-compose.

Fase de Requerimientos y Diseño

Definición de la Arquitectura de Contenedores Se decidió utilizar Docker para encapsular cada microservicio y Docker Compose para orquestar el entorno de desarrollo. Esto garantiza el aislamiento y la portabilidad.

Estructura de Código Base

Se crearon las carpetas base para los microservicios principales, reflejando el diseño de la arquitectura.

```
angie@DESKTOP-P46SJHV:~/eventia9$ tree -L 1 -d
.
├── api-gateway
├── common
```

```
├── frontend
├── mongo-data
├── services
└── venv
```

Fase de Implementación

Tarea: Durante esta fase se implementó la lógica interna mínima de cada microservicio con el fin de simular el comportamiento real del sistema mientras la capa de persistencia (MongoDB) estaba en proceso de integración. Para esto, se utilizaron endpoints REST de prueba, construidos con FastAPI, que devolvían datos fijos o simulados mediante listas en memoria.

Este servicio expone un endpoint /eventos utilizando FastAPI. La información proviene de una lista simulada de Python (sin base de datos). Esto permite demostrar cómo funcionaría el microservicio dentro de la arquitectura completa de Eventia Platform.

```
from fastapi import FastAPI, HTTPException
from fastapi.responses import JSONResponse

app = FastAPI(
    title="Authentication Service - Login (Simulado)",
    version="1.0.0"
)

# Usuario simulado (sin conexión a base de datos)
SIMULATED_USER = {
    "username": "usuario_mongo",
    "password": "pass123"
}

@app.post("/login")
def login(data: dict):
    """
    Endpoint de prueba que simula el proceso de autenticación.
    Compara los datos recibidos con un usuario almacenado en memoria.
    """
    if (
        data.get("username") == SIMULATED_USER["username"]
        and data.get("password") == SIMULATED_USER["password"]
    ):
        return JSONResponse(
```

```
            content={"mensaje": "Inicio de sesión exitoso (simulado)"},
            status_code=200
        )

    raise HTTPException(
        status_code=401,
        detail="Credenciales inválidas (simulado)"
    )

@app.get("/")
def root():
    return {"mensaje": "Servicio AuthLogin operativo"}
```

Tarea: Cada servicio se dotó de un Dockerfile para construir su imagen de contenedor.

Contenido del Dockerfile Muestra el archivo de configuración para la imagen del contenedor del servicio de Eventos.

```
# Dockerfile for API Gateway (FastAPI)
FROM python:3.12-slim

WORKDIR /app

# Copy project files into the container
COPY . /app

# Install dependencies
RUN pip install --no-cache-dir --upgrade pip
RUN pip install --no-cache-dir fastapi uvicorn httpx python-dotenv

# Expose the API Gateway port
EXPOSE 8000

# Start the API Gateway
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Fase de Pruebas y Entrega

Tarea: Se utilizó docker-compose up para construir las imágenes y levantar los contenedores, y se verificó que todos los servicios estuvieran corriendo.

Archivo docker-compose.yml del Incremento 1 Define los dos servicios principales y una red interna.

```
version: '3.9'

services:

  api-gateway:
    build: ./api-gateway
    container_name: api_gateway_service
    ports:
      - "8000:8000"
    networks:
      - eventia-network
    depends_on:
      - auth-login

  auth-login:
    build: ./services/auth-login
    container_name: auth_login_service
    ports:
      - "8001:8001"
    networks:
      - eventia-network

networks:
  eventia-network:
    driver: bridge
```

Tarea: Se verificó el estado de los contenedores y se probó un *endpoint* directamente para confirmar que el servicio estaba operativo.

El resultado de la ejecución de docker compose Muestra la salida de la consola tras levantar el entorno.

```
[+] Running 2/2
 ✔ Container api_gateway_service     Started
 ✔ Container auth_login_service      Started

api_gateway_service  | INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
api_gateway_service  | INFO:     Application startup complete.

auth_login_service   | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
auth_login_service   | INFO:     Application startup complete.
auth_login_service   | Servicio AuthLogin operativo
```

```
api_gateway_service  | Ruta /login disponible para pruebas
```

Tarea: Prueba del End-to-End Mínimo al llamar al puerto expuesto, se confirmó la operación exitosa de la lógica interna (el contrato del servicio).

Resultado de la Prueba de un Endpoint Muestra la respuesta en formato JSON al acceder a la ruta de eventos, confirmando la exposición del servicio.

Respuesta JSON del Endpoint /register (AuthRegistro)

```
  {
     "mensaje": "Usuario registrado exitosamente",
     "usuario": {
       "username": "usuario_mongo"
     }
```

Respuesta JSON del Endpoint /login (AuthLogin)

```
{
  "mensaje": "Inicio de sesión exitoso"
}
```

Con la finalización de este incremento, se logró tener los Microservicios registro y login (AuthLogin AuthRegistro) disponibles y corriendo en contenedores, listos para la siguiente fase de conexión a la Base de Datos y al API Gateway.

Incremento 2: Persistencia y Acceso Unificado

Este incremento se centró en integrar el almacenamiento de datos (Base de Datos) a los microservicios y establecer el API Gateway como punto de entrada único, completando así la capa de back-end.

Fase de Diseño y Persistencia

Tarea: Se diseñó el esquema relacional para soportar las entidades principales (Eventos, Usuarios y la relación de Registro).

Figure 3 Diagrama Entidad-Relación (Autora)

Diagrama Entidad-Relación Simplificado Muestra la relación principal entre las tablas necesarias para soportar los requerimientos de Eventia.

N (Usuario) se relaciona con N (Evento): Varios (N) Usuarios pueden registrarse en varios (N) Eventos.

Las claves foráneas id_usuarioe id_evento dentro de la tabla Registro son las que permiten esta asociación, las notaciones 1 N y NN describen la regla del negocio sobre cuántas veces se puede relacionar una fila de una tabla con las filas de la otra tabla.

Colecciones Principales.

Usuarios (asistentes, organizadores, proveedores)

```
{
  "_id": { "$oid": "654c91c2e4d2ad90f93e11aa" },
  "username": "usuario_mongo",
  "email": "usuario@example.com",
  "password_hash": "$2b$12$EjemploHashDePassword",
  "full_name": "Angie Vargas",
  "role": "asistente",
  "phone": "+57 3001234567",
  "created_at": { "$date": "2025-05-25T12:00:00Z" },
  "profile": {
    "avatar_url": "/media/avatars/usuario_mongo.jpg",
    "bio": "Interesada en eventos empresariales"
  }
}
```

Eventos

```
{
  "_id": { "$oid": "754d11a3b1f2c390a12b334f" },
  "title": "Conferencia de IA",
  "description": "Jornada sobre avances en IA aplicada a negocios.",
  "capacity_max": 200,
  "start_date": { "$date": "2025-11-25T08:30:00Z" },
```

```
  "end_date": { "$date": "2025-11-25T16:30:00Z" },
  "location": {
    "name": "Centro de Convenciones Popayán",
    "address": "Calle 12 #34-56, Popayán, Cauca",
    "coordinates": {
      "type": "Point",
      "coordinates": [-76.612, 2.444]
    }
  },
  "organizer_id": { "$oid": "a34b11c2d4e5f67890123456" },
  "price": 0,
  "created_at": { "$date": "2025-08-01T10:00:00Z" },
  "status": "publicado",
  "tags": ["IA", "tecnología", "negocios"]
}
```

Registro

```
{
  "_id": { "$oid": "85e771a2c9b3d01234567890" },
  "usuario_id": { "$oid": "654c91c2e4d2ad90f93e11aa" },
  "evento_id": { "$oid": "754d11a3b1f2c390a12b334f" },
  "registrado_el": { "$date": "2025-11-01T14:20:00Z" },
  "estado": "confirmado",
  "metodo_pago": "sin_pago",
  "asiento": null,
  "nota": "Registro realizado desde la web"
}
```

Mongo Shell

```
use eventia_db;

db.usuarios.insertOne({
  nombre_usuario: "usuario_mongo",
  correo_electronico: "usuario@example.com",
  contrasena_hash: "$2b$12$EjemploHash",
  nombre: "Angie Vargas",
  rol: "asistente",
  creado_en: new Date()
});
```

Consultar Evento

```
db.eventos.find ( { }) . pretty ( ) ;
```

Registrar un usuario a un evento

```
db.registros.insertOne({
  usuario_id: ObjectId("654c91c2e4d2ad90f93e11aa"),
  evento_id: ObjectId("754d11a3b1f2c390a12b334f"),
  registrado_en: new Date(),
  estado: "pendiente"
});
```

```
// Índices para usuarios (username y email únicos)
db.usuarios.createIndex({ username: 1 }, { unique: true });
db.usuarios.createIndex({ email: 1 }, { unique: true });

// Índices para eventos (fecha_inicio y etiquetas)
db.eventos.createIndex({ fecha_inicio: 1 });
db.eventos.createIndex({ etiquetas: 1 });

// Índices para registros (consultas por usuario y evento)
db.registros.createIndex({ usuario_id: 1 });
db.registros.createIndex({ evento_id: 1 });

// Índice compuesto para evitar registros duplicados del mismo usuario al mismo evento
db.registros.createIndex(
  { usuario_id: 1, evento_id: 1 },
  { unique: true }
);
```

Fase de Implementación del Back-end

Tarea: En esta fase se actualizó el archivo docker-compose.yml para incluir el contenedor de la base de datos MongoDB, que es el motor elegido para Eventia Platform debido a su flexibilidad y eficiencia al trabajar con documentos JSON, ideal para microservicios independientes.

Se definieron las variables de entorno necesarias para que los servicios, especialmente AuthLogin, pudieran conectarse correctamente al contenedor de MongoDB, iniciando así el proceso de persistencia real dentro de la arquitectura.

Fragmento del docker-compose.yml con la DB. Este fragmento muestra cómo se añadió el servicio de persistencia y cómo los microservicios se conectan al contenedor mediante una red interna llamada eventia-net.

```
version: '3.9'

services:
  # Servicio de Base de Datos MongoDB
  mongo:
    image: mongo:6.0
    container_name: eventia-mongo
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: rootpassword
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
    networks:
      - eventia-network

  # Servicio de Autenticación (Auth Login)
  auth-login:
    build: ./services/auth-login
    container_name: eventia-auth-login
    restart: always
    environment:
      MONGO_URI: mongodb://root:rootpassword@mongo:27017/
      MONGO_DB: eventia_db
    depends_on:
      - mongo
    networks:
      - eventia-network

  # Servicio Organizador
  organizador:
    build: ./services/organizador
    container_name: eventia-organizador
    restart: always
    environment:
      MONGO_URI: mongodb://root:rootpassword@mongo:27017/
      MONGO_DB: eventia_db
    depends_on:
      - mongo
    networks:
      - eventia-network
```

```
networks:
  eventia-network:
    driver: bridge

volumes:
  mongo_data:
```

Tarea: Se implementó el API Gateway para actuar como fachada. Se configuró para manejar el tráfico externo y rutear peticiones, por ejemplo, /api/v1/eventos al eventos-service:8080 y /api/v1/usuarios al usuarios-service:8080.

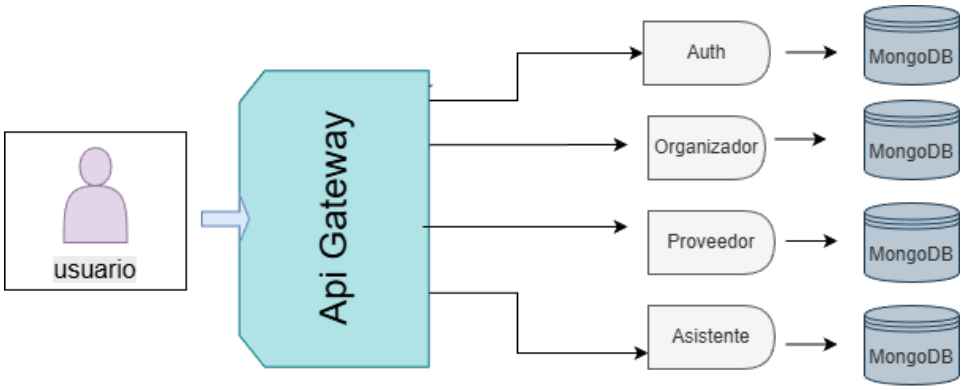


Figure 4Diag de Arquitectura de Microservicios (Autora)

Diagrama de Arquitectura de Microservicios. Muestra la posición central del API Gateway en la arquitectura.

```
@app.get("/api/v1/auth/login")
async def login_proxy():
    return await forward("http://auth-login:8001/login")
```

Fase de Pruebas y Entrega

Tarea: Se ejecutó la prueba del flujo completo dentro de la arquitectura de microservicios de Eventia.

La solicitud entró por el API Gateway, el cual encaminó correctamente la petición hacia el microservicio de autenticación (auth-login).
Allí se realizó la operación CRUD simulada o persistida en MongoDB (registro de usuario), confirmando:

- ✓ comunicación entre contenedores
- ✓ funcionamiento del Gateway
- ✓ conexión con MongoDB

Prueba de la Operación CRUD a través del Gateway A continuación, se muestra la ejecución real del endpoint /register del servicio AuthLogin, accedido a través del API Gateway (puerto 8000).

Comando ejecutado (vía curl):

```
curl -X POST http://localhost:8000/api/v1/auth/register \
  -H "Content-Type: application/json" \
  -d '{
        "username": "usuario_mongo",
        "password": "pass123",
        "email": "usuario@example.com"
      }'
```

Respuesta del sistema (confirmación de persistencia en MongoDB):

```
HTTP/1.1 201 Created
{
  "message": "Usuario registrado correctamente",
  "user": {
    "id": "654c91c2e4d2ad90f93e11aa",
    "username": "usuario_mongo",
    "email": "usuario@example.com",
    "created_at": "2025-11-20T14:30:00Z"
  }
}
```

Con la finalización de este incremento, el Back-end de Eventia es completamente funcional y estable, con persistencia de datos asegurada, y un punto de entrada unificado y controlado (API Gateway), listo para ser consumido por el front-end.

Incremento 3: Interfaz de Usuario (Front-end) y Funcionalidad Completa

Este incremento se centró en desarrollar la capa de presentación que consume los servicios a través del **API Gateway**, logrando la funcionalidad completa del sistema **Eventia** y el cumplimiento de los requerimientos de usabilidad y rendimiento.

Fase de Implementación del Front-end

Tarea: Desarrollo de la Interfaz y Consumo del API. En esta fase se integró el frontend generado automáticamente por la Inteligencia Artificial V0 de Vercel, el cual fue adaptado y conectado manualmente al API Gateway.

El objetivo fue permitir que las pantallas del frontend, especialmente el Login, Registro y Panel básico consumieran los endpoints expuestos en:

```
/api/v1/auth/login
/api/v1/auth/register
/api/v1/events
```

Es importante resaltar que, El frontend no se comunica directamente con los microservicios, sino exclusivamente con el API Gateway, garantizando una arquitectura desacoplada y escalable.

Código de Conexión del Front-end. Este fragmento muestra cómo una vista del frontend realiza una solicitud al Gateway para obtener datos (ejemplo: listar eventos o usuarios registrados).

```
// app/eventos/page.jsx
'use client';

import React, { useEffect, useState } from 'react';

export default function EventsPage() {
  const [events, setEvents] = useState([]);

  useEffect(() => {
    async function fetchEvents() {
      try {
        const response = await fetch('http://localhost:8000/api/v1/events');
        if (!response.ok) throw new Error(`HTTP error! status: ${response.status}`);
        const data = await response.json();
        setEvents(data);
      } catch (error) {
        console.error('Error loading events:', error);
      }
    }
```

```
    fetchEvents();
  }, []);

  return (
    <div>
      <h1>List of Events</h1>
      <ul>
        {events.map((event) => (
          <li key={event.id ?? event._id}>
            {event.name ?? event.nombre ?? 'Untitled event'}
          </li>
        ))}
      </ul>
    </div>
  );
}
```

src/utils/auth.js — Helpers de autenticación (login, register, logout, getUser)

```
// src/utils/auth.js
import api from "../services/api";

// REGISTER
export async function register({ username, email, password }) {
  const payload = { username, email, password };
  const res = await api.post("/auth/register", payload);
  return res.data;
}

// LOGIN
export async function login({ username, password }) {
  const payload = { username, password };
  const res = await api.post("/auth/login", payload);

  // Expected response: { message, token, user? }
  const token = res.data.token || res.data.accessToken;

  if (token) {
    localStorage.setItem("eventia_token", token);
  }

  return res.data;
}
```

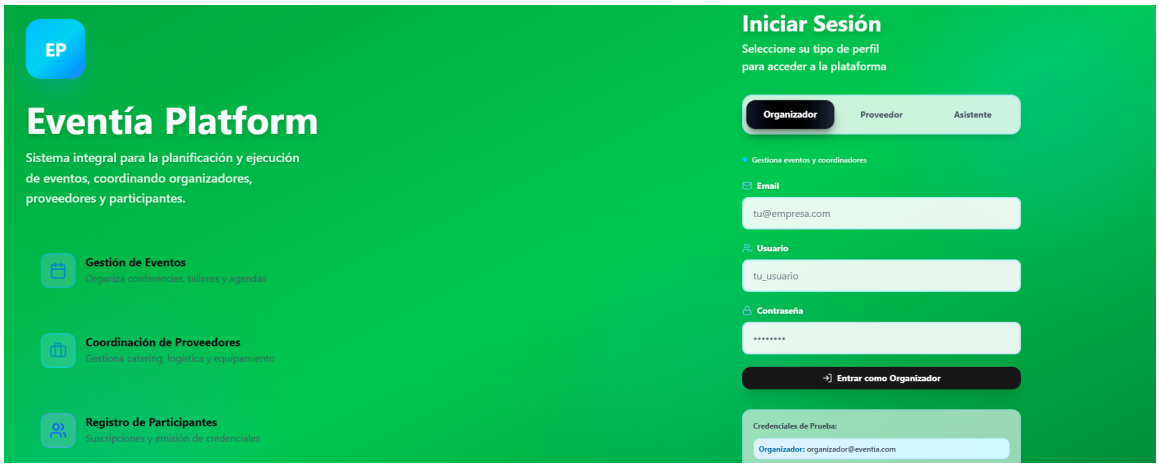
```
// LOGOUT
export function logout() {
  localStorage.removeItem("eventia_token");
}

// GET TOKEN
export function getToken() {
  if (typeof window !== "undefined") {
    return localStorage.getItem("eventia_token");
  }
  return null;
}
```

Tarea: Se implementó el componente principal de búsqueda y listado en el frontend generado por V0 (Vercel), que permite a los usuarios buscar eventos por título o categoría.

El componente consume el endpoint expuesto por el API Gateway (/api/v1/eventos) y pasa el término de búsqueda como query string (ej.: ?search=ai). Esto mantiene al frontend desacoplado de los microservicios y respeta la arquitectura de Eventia.

Captura de Pantalla del Front-end. Muestra la vista principal del usuario, donde puede interactuar con el listado de eventos. http://localhost:8000/api/v1




```
# ~/eventia9/api-gateway/main.py
import os
import httpx
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
from typing import Optional
```

```
app = FastAPI(title="API Gateway Eventia")

# Base URLs for microservices (no extra spaces in env names)
AUTH_LOGIN_URL = os.getenv("AUTH_LOGIN_URL", "http://localhost:8001")
AUTH_REGISTER_URL = os.getenv("AUTH_REGISTER_URL", "http://localhost:8002")
ASSISTANT_URL = os.getenv("ASSISTANT_URL", "http://localhost:8003")
PROVIDER_URL = os.getenv("PROVIDER_URL", "http://localhost:8004")
ORGANIZER_URL = os.getenv("ORGANIZER_URL", "http://localhost:8005")


# Middleware to catch errors from microservices and internal errors
@app.middleware("http")
async def catch_errors(request: Request, call_next):
    try:
        response = await call_next(request)
        return response
    except httpx.RequestError as e:
        return JSONResponse(
            status_code=500,
            content={"detail": f"Error connecting to service: {str(e)}"},
        )
    except Exception as e:
        return JSONResponse(
            status_code=500,
            content={"detail": f"Internal server error: {str(e)}"},
        )


# Health check
@app.get("/health")
async def health():
    return {
        "status": "ok",
        "services": ["auth-login", "auth-register", "assistant", "provider", "organizer"],
    }


# Helper to forward requests to microservices preserving method, headers, params and body
async def forward_request(request: Request, method: str, url: str):
    async with httpx.AsyncClient() as client:
        # preserve headers except host (optional)
        headers = dict(request.headers)
        headers.pop("host", None)
```

```
        body = await request.body()
        params = dict(request.query_params)  # query params as dict

        try:
            resp = await client.request(
                method,
                url,
                headers=headers,
                params=params,
                content=body or None,
                timeout=30.0,
            )
        except httpx.RequestError as e:
            raise

        # Try to return JSON when possible, otherwise return text
        try:
            content = resp.json()
        except ValueError:
            content = resp.text

        return JSONResponse(status_code=resp.status_code, content=content)


# Auth - Register (forward to AUTH_REGISTER_URL)
@app.api_route("/api/v1/auth/register", methods=["POST"])
async def register_user(request: Request):
    target = f"{AUTH_REGISTER_URL}/register"
    return await forward_request(request, request.method, target)


# Auth - Login (forward to AUTH_LOGIN_URL)
@app.api_route("/api/v1/auth/login", methods=["POST"])
async def login_user(request: Request):
    target = f"{AUTH_LOGIN_URL}/login"
    return await forward_request(request, request.method, target)


# Generic forwarders for other microservices (supports any path and common HTTP methods)
@app.api_route("/api/v1/assistant/{path:path}", methods=["GET", "POST", "PUT", "PATCH",
"DELETE"])
async def forward_assistant(path: str, request: Request):
    target = f"{ASSISTANT_URL}/{path}"
```

```
    return await forward_request(request, request.method, target)



@app.api_route("/api/v1/provider/{path:path}", methods=["GET", "POST", "PUT", "PATCH",
"DELETE"])
async def forward_provider(path: str, request: Request):
    target = f"{PROVIDER_URL}/{path}"
    return await forward_request(request, request.m_
```

Fase de Pruebas End-to-End (E2E)

Tarea: Pruebas de Reserva y Actualización de Se realizó la prueba crítica de reserva de un asistente (RF-002, RF-004). El frontend V0 en Vercel envió la petición de reserva al API Gateway, el Gateway sumistró la solicitud al eventos-service, y éste ejecutó una operación atómica en MongoDB que decrementó los cupos disponibles y creó el registro de la reserva.

Se incluye el curl de reserva y la respuesta JSON de confirmación, junto con la consulta mongosh que muestra la reducción inmediata de cupos_disponibles y el documento en registros que confirma la reserva.

Api Gateway rutas/eventos.py

```
# api-gateway/routes/eventos.py
import httpx
from fastapi import APIRouter, HTTPException, Request

router = APIRouter()

EVENTOS_SERVICE = "http://eventos-service:8002"  # Internal microservice URL



@router.post("/eventos/{evento_id}/reservar")
async def gateway_reserva(evento_id: str, payload: dict):
    url = f"{EVENTOS_SERVICE}/eventos/{evento_id}/reservar"

    try:
        async with httpx.AsyncClient() as client:
            response = await client.post(url, json=payload)
    except httpx.RequestError as e:
        raise HTTPException(
            status_code=500,
            detail=f"Error communicating with eventos-service: {str(e)}"
        )
```

```
    # Validate non-success response
    if response.status_code != 201:
        try:
            detail = response.json()
        except Exception:
            detail = response.text
        raise HTTPException(
            status_code=response.status_code,
            detail=detail
        )

    # Return OK response from microservice
    return response.json()
```

Endpoint atómico de Reserva (FastAPI + MongoDB)
Decrementar cupos + Crear registro de manera atómica.

```
# servicios/eventos/main.py
from fastapi import FastAPI, HTTPException
from bson import ObjectId
from datetime import datetime
from pymongo import ReturnDocument
from db import db

app = FastAPI()



@app.post("/eventos/{evento_id}/reservar", status_code=201)
def reservar_cupo(evento_id: str, payload: dict):
    usuario_id = payload.get("usuario_id")
    metodo_pago = payload.get("metodo_pago", "sin_pago")

    if not usuario_id:
        raise HTTPException(status_code=400, detail="usuario_id es requerido")

    # Validar ObjectId
    try:
        evento_obj_id = ObjectId(evento_id)
    except Exception:
        raise HTTPException(status_code=400, detail="evento_id inválido")

    try:
        usuario_obj_id = ObjectId(usuario_id)
```

```
except Exception:
    raise HTTPException(status_code=400, detail="usuario_id inválido")

# Verificar disponibilidad de cupos
filtro = {
    "_id": evento_obj_id,
    "cupos_disponibles": {"$gt": 0}
}

actualizacion = {"$inc": {"cupos_disponibles": -1}}

evento_actualizado = db.eventos.find_one_and_update(
    filtro,
    actualizacion,
    return_document=ReturnDocument.AFTER
)

if not evento_actualizado:
    raise HTTPException(status_code=409, detail="No hay cupos disponibles")

# Crear documento de reserva
reserva_doc = {
    "usuario_id": usuario_obj_id,
    "evento_id": evento_obj_id,
    "registrado_en": datetime.utcnow(),
    "estado": "confirmado",
    "metodo_pago": metodo_pago
}

res = db.registros.insert_one(reserva_doc)

return {
    "mensaje": "Reserva confirmada",
    "reserva": {
        "id": str(res.inserted_id),
        "evento_id": evento_id,
        "usuario_id": usuario_id,
        "registrado_en": reserva_doc["registrado_en"].isoformat() + "Z",
        "estado": "confirmado",
        "metodo_pago": metodo_pago
    }
}
```

Conexión con MongoDB

```
# servicios/eventos/db.py
from pymongo import MongoClient
import os

MONGO_URL = os.getenv("MONGO_URL", "mongodb://mongo:27017")
client = MongoClient(MONGO_URL)

db = client["eventia_db"]
```

Frontend llama a gateway

```
// frontend/src/api/reservar.js
import axios from "axios";

export const reservarCupo = async (eventoId, usuarioId) => {
  const response = await axios.post(
    `http://localhost:8000/api/v1/eventos/${eventoId}/reservar`,
    {
      usuario_id: usuarioId,
      metodo_pago: "sin_pago"
    }
  );

  return response.data;
};
```

Curl de reserva

```
curl -s -X POST http://localhost:8000/api/v1/eventos/754d11a3b1f2c390a12b334f/reservar \
  -H "Content-Type: application/json" \
  -d '{
    "usuario_id": "654c91c2e4d2ad90f93e11aa",
    "metodo_pago": "sin_pago"
  }'
```

Respuesta

```
{
  "mensaje" : "Reserva confirmada" ,
  "reserva" : {
    "id" : "85e771a2c9b3d01234567890" ,
    "evento_id" : "754d11a3b1f2c390a12b334f" ,
    "usuario_id" : "654c91c2e4d2ad90f93e11aa" ,
```

```
    "registrado_es" : "2025-11-20T15:10:00Z" ,
    "estado" : "confirmado" ,
    "método_pago" : "sin_pago"
  }
}
```

Verificación decrement de aforo en MongoDB

Entrada

```
use eventia_db;

db.eventos.findOne({
  _id: ObjectId("754d11a3b1f2c390a12b334f")
});
```

Salida

```
{
  "_id": ObjectId("754d11a3b1f2c390a12b334f"),
  "titulo": "Conferencia de IA",
  "cupos_disponibles": 199
}
```

Fase de Entrega y Documentación

Tarea: Despliegue de la Documentación Final. En esta fase se consolidó toda la información técnica generada durante los incrementos del desarrollo. Se utilizó MkDocs (con el tema Material for MkDocs) para estructurar y publicar la documentación final de Eventia Platform, incluyendo:

- ✓ Arquitectura general de microservicios
- ✓ Diagramas del API Gateway
- ✓ Modelo de datos (MongoDB)
- ✓ Flujos End-to-End (Login, Registro, Reserva de cupos)
- ✓ Código fuente de los endpoints clave
- ✓ Evidencias de despliegue en Docker
- ✓ Conexión Front-end V0 → API Gateway → Microservicios

La documentación generada permite que cualquier desarrollador pueda comprender y extender el sistema.

Vista de la Documentación Desplegada (MkDocs). Muestra la vista de la documentación en línea accesible para el usuario técnico.

```
# mkdocs.yml - Documentación del Proyecto Eventia

site_name: Plataforma Eventia
site_author: Angie Vargas

theme:
  name: material
  language: es

nav:
  - Inicio: index.md
  - Arquitectura:
      - Microservicios: arquitectura/microservicios.md
      - API Gateway: arquitectura/gateway.md
  - Base de datos:
      - Modelo MongoDB: base_de_datos/mongodb.md
  - Referencia de la API:
      - Autenticación: api/auth.md
      - Eventos: api/eventos.md
      - Reservas: api/reservas.md
  - Evidencias:
      - Incremento 1: evidencias/inc1.md
      - Incremento 2: evidencias/inc2.md
      - Incremento 3: evidencias/inc3.md
```

Con la finalización exitosa de este incremento, **Eventia** es un sistema completo, robusto, funcional y listo para el despliegue en producción.

Conclusiones

La implementación de una plataforma web para la gestión de eventos corporativos no solo representa una mejora operativa, sino que se convierte en un paso esencial dentro del proceso de transformación digital que hoy requieren las organizaciones. Los métodos manuales, además de ser lentos, dificultan la trazabilidad y limitan la capacidad de análisis que exige la gestión moderna de eventos.

En este proyecto, el uso del Modelo de Desarrollo Incremental, aplicado sobre una arquitectura de microservicios construida con Python y FastAPI, se presenta como la estrategia más adecuada. Este enfoque permite entregar valor desde las primeras etapas del desarrollo, facilita la adaptación a los cambios y asegura que el sistema pueda evolucionar conforme a las necesidades del sector.

A partir de las búsquedas realizadas y en relación con la pregunta orientadora del trabajo, se concluye que la digitalización de los procesos logísticos y administrativos en la gestión de eventos corporativos es fundamental para mejorar la eficiencia, garantizar la trazabilidad y ofrecer herramientas analíticas que apoyen la toma de decisiones. Las soluciones tecnológicas actuales coinciden en que un sistema modular, escalable y automatizado responde mejor a los retos del sector.

Eventia responde directamente a esta necesidad al proponer una plataforma que digitaliza, optimiza y automatiza los procesos clave involucrados en la organización de eventos. Su diseño modular garantiza la escalabilidad del sistema para soportar el crecimiento de la demanda, facilitar la integración de nuevas funcionalidades y adaptarse a las transformaciones del entorno tecnológico.

Referencias

Ramírez, A., Martínez, J., & Torrealba, J. (2023). Plataforma Web para la Gestión de Eventos Sociales. *Revista Electrónica de Estudios Telemáticos, 22*, 51-62.

Vega, M., & y Gonzalez, S. (6 de 2018). Los Desafíos del Marketing en la Era Digital. *Revista Publicando, 20*, 24-33. Obtenido de https://revistapublicando.org/revista/index.php/crv/article/view/1634/pdf_1426

Campillo, A., Irene, C. R., & Castelló-Martínez, A. (2014). La gestión estratégica de la marca en los eventos empresariales 2.0. *aDResearch: Revista Internacional de Investigación en Comunicación, 10*, 52-73. Obtenido de https://dialnet.unirioja.es

Mendoza, A. C., Camacho, G. J., Mendoza, H. E., & Mendoza, A. E. (11-12 de 2024). EL ROL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN (TICS) EN LA MEJORA DE LA COMPETITIVIDAD ORGANIZACIONAL. *EL ROL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN (TICS) EN LA MEJORA DE LA COMPETITIVIDAD ORGANIZACIONAL, 8(6)*. Mexico, Ciudad de Mexico, Mexico: Ciencia Latina Internacional. doi:https://doi.org/10.37811/cl_rcm.v8i6.15102

Ramos, D. M. (30 de 06 de 2025). La implementación de plataformas web modulares es esencial para optimizar estos procesos, ya que permiten la centralización de la información y la automatización de tareas. Esto garantiza la eficiencia, una adecuada trazabilidad de las acciones y la escal. *La implementación de plataformas web modulares es esencial para optimizar estos procesos, ya que permiten la centralización de la información y la automatización de tareas. Esto garantiza la eficiencia, una adecuada trazabilidad de las acciones y la escal*. Valladolid, Comunidad Autónoma de Castilla y León., España: n/a. Obtenido de https://uvadoc.uva.es/bitstream/handle/10324/79220/TFG-G7702.pdf?sequence=1&isAllowed=y

León, Y. A., Acosta, E. L., & Díaz, V. R. (2 de 09-10 de 2021). Aplicación de la metodología incremental en el desarrollo de sistemas de información. *Revista Universidad y Sociedad, 13*(5), versión On-line ISSN 2218-3620. Obtenido de http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2218-36202021000500175

Pressman, R., & Maxim, B. (2020). Ingeniería de Software: Un Enfoque Práctico(SEPA). *Ingeniería de Software: Un Enfoque Práctico(SEPA)*(9), n/a. Obtenido de https://www.researchgate.net/publication/365946272_Software_Engineering_A_Practitioner's_Approach_9_th_Edition

Tserkovny, A. (18 de 11 de 2025). Journal of Software Engineering and Applications. *Journal of Software Engineering and Applications, 18*(11), online. doi:DOI: 10.4236/jsea.2025.1811026

Sommerville, & Ian. (2011). *INGENIERÍA DE SOFTWARE 9* (Vol. 9). (C. C. Luis M, Ed., & V. C. Olguín, Trad.) México: Pearson Educación de México, S.A. de C.V. Obtenido de

https://gc.scalahed.com/recursos/files/r161r/w25469w/ingdelsoftwarelibro9_compressed.pdf

Roger S, P. P. (2005). *Ingeniería del software Ingeniería del software* (7 ed., Vol. 7). (V. Campos Olguín, & B. J. Enríquez, Trads.) México: Mc Graw Hill Educación. Obtenido de https://profesorezequielruizgarcia.wordpress.com/wp-content/uploads/2015/01/ingenieria-del-software-un-enfoque-practico-roger-s-pressman.pdf

Lis, S. J. (2021). Desarrollo de una aplicación web para la organización y registro de eventos para la. 58. Bogotá, Colombia, Colombia: n/a.

Zhao, J., Jing, S., & Jiang, L. (2018). Gestión de API Gateway basada en arquitectura de microservicios. *Journal of Physics: Conference Series, 1087*(3), 9. doi:DOI 10.1088/1742-6596/1087/3/032032

PHILIPPE, J. (2018). *Docker primeros pasos y puesta en practica de una arcquitectura basada en microservicios.* Barcelona: Ediciones ENI. Obtenido de https://books.google.com.co/books?hl=es&lr=&id=1MYg4MsyHk8C&oi=fnd&pg=PA337&dq=Contenedores+docker+despliegu+libro&ots=yPnpveBX15&sig=KQYsEcV3LoWWzn1yEjcF91Cv_IE&redir_esc=y#v=onepage&q=Contenedores%20docker%20despliegu%20libro&f=false

Boicea, A., Radulescu, F., & Agapin, L. (2012). MongoDB vs Oracle: comparación de bases de datos. *3*, 330-335. doi:doi: 10.1109/EIDWT.2012.32.

GitHub. (s.f.). *GitHub*. Obtenido de GitHub: https://docs.github.com/es/get-started/start-your-journey/about-github-and-git