DOCENTE:

RUBÉN DARÍO GONZÁLEZ BARRERA ASIGNATURA:

ARQUITECTURA DE COMPUTADORES

ESTUDIANTES:

DEIBID BENAVIDES YAYA

MATEO RAMIREZ ORTIZ

MATEO ALEJANDRO VARGAS VALERO

NRC: 80983

INSTITUCIÓN: UNIVERSIDAD MINUTO DE DIOS

FECHA DE ENTREGA: /17/11/2025

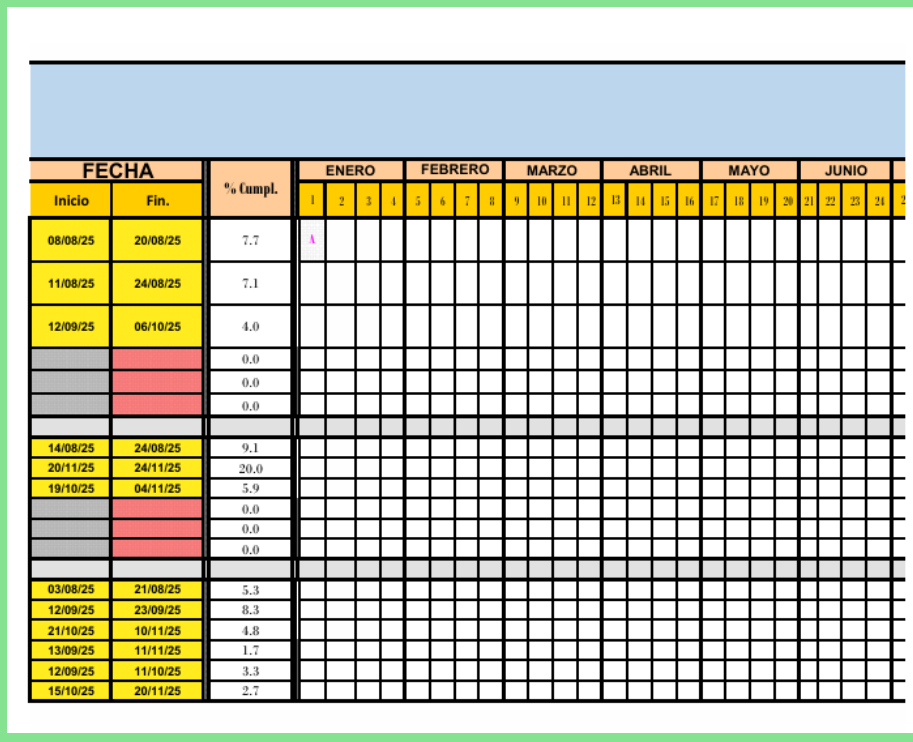
1.Crear el repositorio del proyecto (GitHub o GitLab).

https://github.com/varagasmateo0-lgtm/ARQUITECTURA-mmv.git

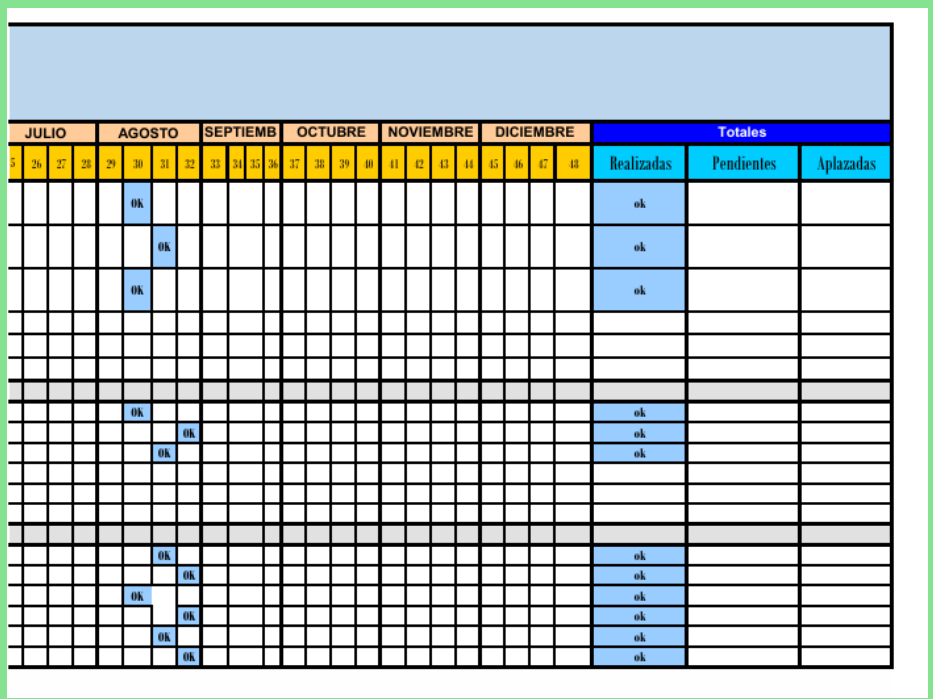
2. Dividir tareas entre los integrantes del grupo.

cronograma

NOMBRE DE LA ENTIDAD DONDE RELIZARA EL PROYECTO			
FASE	**ACTIVIDADES**	**RESPONSABLE**	**PRODUCTO**
ANALISIS	Definición de Requerimientos	Analista Líder	Documento de Requisitos
	Selección de Tecnología del Bus	Arquitecto/Dev Líder	Decisión tecnológica
	Diseño de Tópicos Iniciales	Analista/Arquitecto	Listado y especificación
PLANEACION	Diseño de la Arquitectura Lógica	Arquitecto	Diagrama
	Configuración del Entorno (Dev/Test)	DevOps	Servidores del Bus
	Planificación de Integración (MVP)	Dev Líder	Documento
EJECUCION	Desarrollo del Servicio Publicador (Contactos)	Equipo de	Microservicio
	.Desarrollo del Servicio Suscriptor	Equipo de	consume
	Desarrollo del Servicio Suscriptor	Equipo de	Reporte de Pruebas
	Despliegue en Producción (Go-Live)	QA/DevOps	Sistema de Bus
	Monitorización y Estabilización	Equipo de Soporte	Monitorización
	Pruebas de Integración y Carga	QA/DevOps	Lecciones

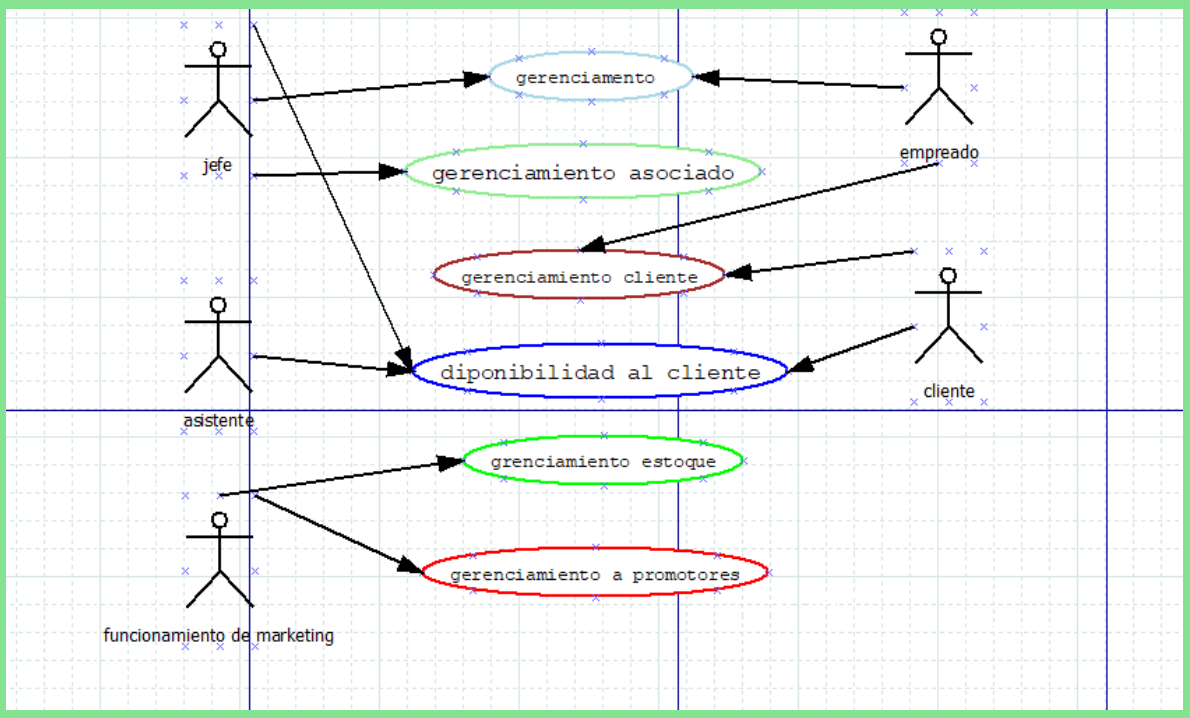


FECHA		% Cumpl.	ENERO				FEBRERO				MARZO				ABRIL				MAYO				JUNIO			
Inicio	Fin.		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
08/08/25	20/08/25	7.7	A																							
11/08/25	24/08/25	7.1																								
12/09/25	06/10/25	4.0																								
		0.0																								
		0.0																								
		0.0																								
14/08/25	24/08/25	9.1																								
20/11/25	24/11/25	20.0																								
19/10/25	04/11/25	5.9																								
		0.0																								
		0.0																								
		0.0																								
03/08/25	21/08/25	5.3																								
12/09/25	23/09/25	8.3																								
21/10/25	10/11/25	4.8																								
13/09/25	11/11/25	1.7																								
12/09/25	11/10/25	3.3																								
15/10/25	20/11/25	2.7																								



JULIO				AGOSTO				SEPTIEMB				OCTUBRE				NOVIEMBRE				DICIEMBRE				Totales		
25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	Realizadas	Pendientes	Aplazadas
					0k																			ok		
						0k																		ok		
					0k																			ok		
					0k																			ok		
							0k																	ok		
						0k																		ok		
						0k																		ok		
							0k																	ok		
					0k																			ok		
						0k																		ok		
							0k																	ok		
							0k																	ok		

3. Programar los módulos principales paso a paso.



4. Aplicar pruebas unitarias y depurar errores.

Depuración de Errores (Debugging)

La depuración en una arquitectura de eventos es más compleja porque el flujo es asíncrono. No se puede seguir una línea de ejecución directa de un servicio a otro.

Estrategias de Depuración:

- **Inspección del Bus (Broker): Utiliza herramientas de monitorización del Bus de Eventos (como *Kafka UI* o herramientas de gestión de *RabbitMQ*) para verificar lo siguiente:**

  Si el evento fue realmente publicado por el servicio de origen.

  Si el evento está en el Tópico correcto.

  Si el evento tiene el formato (schema) esperado.

- **Logging Detallado: Implementar *logging* en puntos críticos de cada servicio:**

Publicador: Registrar cuando se recibe la solicitud y cuando se publica el evento.

Suscriptor: Registrar cuando se recibe el evento del Bus y cuando se inicia la operación de negocio (ej. guardar en DB).

- **ID de Correlación: Este es fundamental. Al recibir una solicitud inicial (ej. POST /contactos), se debe generar un ID de correlación único y propagarlo a través de todos los eventos que se generen a partir de esa solicitud.**

Uso del ID de Correlación: Si un error ocurre en el Servicio Persistencia, puedes buscar en los logs de *todos* los servicios usando ese ID para rastrear el flujo completo desde la solicitud inicial.

- **Entorno de Prueba Controlado: Nunca depurar directamente en producción. Usa entornos de *staging* o *testing* con datos representativos para replicar el error de forma segura.**

Componente	Objetivo de la Prueba Unitaria	Técnicas Clave
Servicio Contactos (Publicador)	Verificar la validación de datos de entrada y que el objeto del evento se construya correctamente antes de su publicación.	*Mockear* (simular) la llamada al Bus para asegurar que el método de publicación se llama una vez con el *payload* correcto.
Servicio Persistencia (Suscriptor)	Verificar que la función de almacenamiento en la base de datos se ejecute correctamente al recibir un evento válido.	*Mockear* el evento entrante (simulando que el Bus lo entrega) y *Mockear* la conexión a la base de datos.

Componente	Objetivo de la Prueba Unitaria	Técnicas Clave
Modelos/Entidades	**Verificar la correcta serialización/deserialización de los datos del evento y la manipulación de las entidades.**	**Crear y modificar instancias de la clase del evento y asegurar que los *getters* y *setters* funcionan según lo esperado.**

5. Documentar el código y avances.

```java
public interface Subscriber {
    /**
     * Método llamado por el Bus de Eventos para manejar un evento recibido.
     * @param topic El canal o tema del evento.
     * @param eventData Los datos (payload) del evento.
     */
    void handleEvent(String topic, String eventData);
}
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class EventBus {

    // Mapa que relaciona el nombre del Tópico con una lista de Suscriptores
    private final Map<String, List<Subscriber>> subscribers;

    public EventBus() {
        this.subscribers = new HashMap<>();
    }

    /**
     * Permite que un servicio se suscriba a un tópico específico.
     */
    public void subscribe(String topic, Subscriber subscriber) {
        // Usa computeIfAbsent para simplificar la creación de la lista si no existe
        this.subscribers.computeIfAbsent(topic, k -> new ArrayList<>()).add(subscriber);
        System.out.println(" " + subscriber.getClass().getSimpleName() + " suscrito a '" + topic + "'");
    }

    /**
     * Publica un evento en un tópico, notificando a todos los suscriptores.
     */
    public void publish(String topic, String eventData) {
        System.out.println("\nPUBLICADO en '" + topic + "': " + eventData);

        // Obtiene y notifica a todos los suscriptores de ese tópico
        if (this.subscribers.containsKey(topic)) {
            for (Subscriber subscriber : this.subscribers.get(topic)) {
                subscriber.handleEvent(topic, eventData);
            }
        } else {
            System.out.println(" No hay suscriptores para el tópico '" + topic + "'.");
        }
    }
}
```

```java
}
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class EventBus {

    // Mapa que relaciona el nombre del Tópico con una lista de Suscriptores
    private final Map<String, List<Subscriber>> subscribers;

    public EventBus() {
        this.subscribers = new HashMap<>();
    }

    /**
     * Permite que un servicio se suscriba a un tópico específico.
     */
    public void subscribe(String topic, Subscriber subscriber) {
        // Usa computeIfAbsent para simplificar la creación de la lista si no existe
        this.subscribers.computeIfAbsent(topic, k -> new ArrayList<>()).add(subscriber);
        System.out.println(" " + subscriber.getClass().getSimpleName() + " suscrito a '" +
    }

    /**
     * Publica un evento en un tópico, notificando a todos los suscriptores.
     */
    public void publish(String topic, String eventData) {
        System.out.println("\n PUBLICADO en '" + topic + "': " + eventData);

        // Obtiene y notifica a todos los suscriptores de ese tópico
        if (this.subscribers.containsKey(topic)) {
            for (Subscriber subscriber : this.subscribers.get(topic)) {
                subscriber.handleEvent(topic, eventData);
            }
        } else {
            System.out.println("No hay suscriptores para el tópico '" + topic + "'.");
        }
    }
}
public class PersistenceService implements Subscriber {
    @Override
    public void handleEvent(String topic, String eventData) {
        System.out.println("[PersistenceService] Recibido: " + topic);
        // Aquí se simularía la lógica de guardar en la DB.
        System.out.println("      GUARDANDO en DB: " + eventData);
    }
}
```

Java: Ready

```java
}
public class NotificationService implements Subscriber {
    @Override
    public void handleEvent(String topic, String eventData) {
        System.out.println("[NotificationService] Recibido: " + topic);
        // Aquí se simularía la lógica de enviar un email.
        System.out.println("  ENVIANDO email: " + eventData + " (Asunto: ¡Bienvenido!)");
    }
}
public class ContactService {
    private final EventBus bus;

    public ContactService(EventBus bus) {
        this.bus = bus;
    }

    public void createContact(String nombre, String email) {
        System.out.println("\n[ContactService] Solicitud para crear contacto: " + nombre);

        // Simulación de la construcción del evento (el payload sería JSON en un sistema real)
        String eventPayload = "{\"nombre\":\"" + nombre + "\", \"email\":\"" + email + "\"}";

        // Publicación del evento
        bus.publish("contacto.creado", eventPayload);
    }
}
public class Main {
    public static void main(String[] args) {
        // 1. Inicializar el Bus de Eventos
        EventBus bus = new EventBus();

        // 2. Inicializar los servicios
        ContactService publicador = new ContactService(bus);
        PersistenceService persistencia = new PersistenceService();
        NotificationService notificaciones = new NotificationService();

        // 3. Suscribir los servicios a los tópicos relevantes
        bus.subscribe("contacto.creado", persistencia);
        bus.subscribe("contacto.creado", notificaciones);

        // 4. INICIAR EL FLUJO DE DATOS
        // El ContactService publica un evento que activará a los suscriptores
        publicador.createContact("Maria Lopez", "maria@ejemplo.com");

        // 5. Otro flujo para demostrar el desacoplamiento
        publicador.createContact("Juan Perez", "juan@ejemplo.com");
    }
}
```

Fase de Costos:

F	G	H	I
Recurso	Cantidad	Costo Unitario (COP)	Costo Total (COP)
Mano de Obra (Desarrollo)	160 h	12.000	1.920.000
Energía Eléctrica	1 mes	30.000	30.000
Internet / Conectividad	1 mes	60.000	60.000
Licencia de IDE / Software	1 mes	45.000	45.000
Infraestructura de Prueba (Bus)	2 mes	150.000	150.000
Herramientas de Monitoreo	1 mes	75.000	75.000
Equipos / Mantenimiento	1 mes	100.000	100.000
TOTAL ESTIMADO:			*2.380.000*