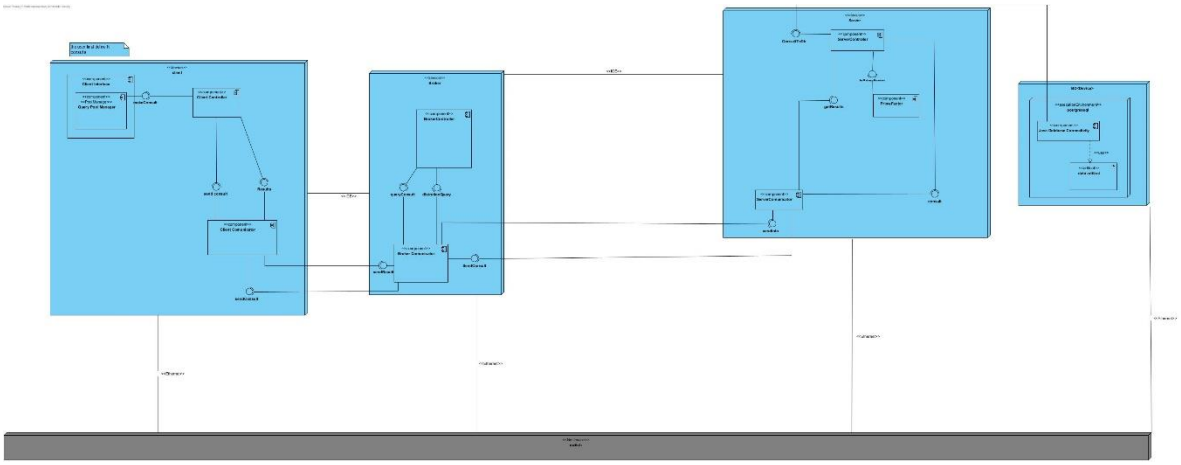
DIAGRAMA DEPLOYMENT SIN PATRONES



El diseño del sistema incluye tres capas principales (base de datos, servidor y cliente) conectadas mediante un **broker** que gestiona la comunicación. Este enfoque mejora la escalabilidad y fiabilidad del sistema al centralizar el intercambio de mensajes entre las diferentes partes.

**1. Capa de Base de Datos (BD):**

- La base de datos **PostgreSQL** aloja toda la información de los votantes, agrupaciones, mesas, y ciudades.
- Es accesible únicamente desde el servidor central para garantizar la seguridad de los datos.

**2. Capa de Consulta (Servidor):**

- Esta capa intermedia se implementa en servidores que procesan las solicitudes de los clientes.
- Realiza las siguientes tareas:
    - Recibe mensajes del broker (consultas de los clientes).
    - Procesa las consultas, incluyendo cálculos como el número de factores primos.
    - Accede a la base de datos para obtener la información solicitada.
    - Responde al cliente a través del broker.

**3. Capa de Cliente:**

- Los clientes son dispositivos ligeros utilizados para realizar consultas ingresando el número de cédula.
- Cada cliente se comunica exclusivamente con el broker, enviando solicitudes y recibiendo respuestas sin necesidad de conocer los detalles del servidor o la base de datos.

**4. Broker de Mensajes:**

- **broker** para centralizar y gestionar la comunicación entre los clientes y el servidor.
    - **Desacoplamiento:** Los clientes y el servidor no necesitan comunicarse directamente; el broker actúa como intermediario, simplificando la arquitectura.

**Infraestructura de Comunicación:**

- El sistema utiliza una red segura y de baja latencia.
- La inclusión del broker permite una mayor flexibilidad en la incorporación de nuevos clientes o servidores sin interrumpir el sistema.

---

**Estrategia para Resolver el Problema**

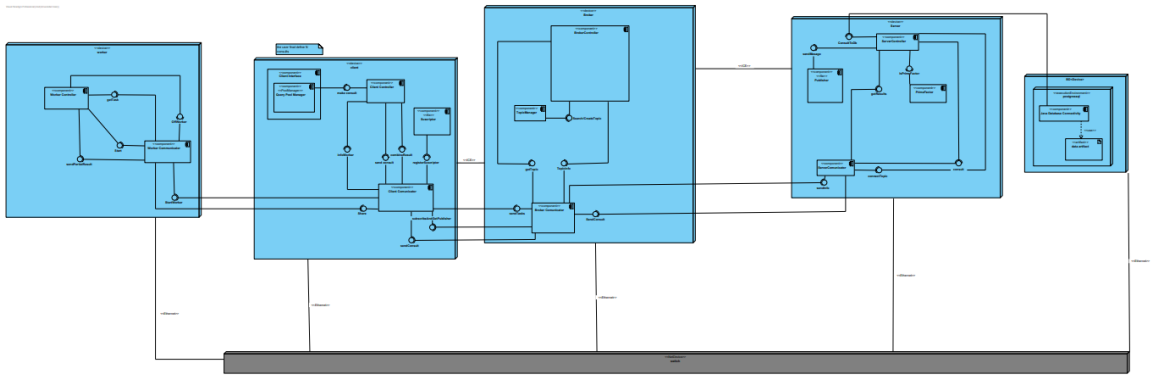
1. **Desacoplamiento de Componentes:**
    - El uso del broker permite que clientes, servidores y la base de datos funcionen de manera independiente.
    - Esto reduce la complejidad del sistema y facilita su mantenimiento.
2. **Escalabilidad:**
    - Se pueden agregar más instancias del servidor o nuevos clientes en tiempo real, ya que el broker balancea automáticamente la carga.
    - El diseño admite un número creciente de consultas sin comprometer el rendimiento.
3. **Alta Disponibilidad:**
    - El broker garantiza que las solicitudes se almacenen temporalmente incluso si un componente está inactivo momentáneamente, reduciendo la probabilidad de pérdida de datos.

4. **Optimización del Rendimiento:**
    - El servidor realiza tareas intensivas como cálculos matemáticos y consultas a la base de datos de manera eficiente.
    - El broker minimiza los tiempos de espera al gestionar de manera óptima las solicitudes y respuestas.
    -

Este diseño, con la incorporación del broker, mejora significativamente la fiabilidad, escalabilidad y capacidad de auditoría del sistema, cumpliendo con los requerimientos funcionales y de calidad especificados.

DIAGRAMA DEPLOYMENT REFINADO CON PATRONES

pdf adjunto (refinamiento diagrama deploy)



MAPEO DE ESTRUCTURA DE FLYNN

Almacenamiento:

Tipo: SIMD (Single Instruction, Multiple Data)

Se accede a la base de datos mediante una única operación de consulta desde el servidor (una instrucción), pero se aplican sobre múltiples datos simultáneamente (registro de votantes, mesas de votación, etc.).

El diseño soporta múltiples clientes consultando información en paralelo mientras la base de datos maneja las operaciones como conjuntos de datos separados.

Implementación:

Base de datos PostgreSQL con replicación para alta disponibilidad.

Indización para consultas rápidas de cédulas y su mesa de votación correspondiente.

Procesamiento:

Servidor:

Tipo: MIMD (Multiple Instruction, Multiple Data)

El servidor maneja múltiples flujos de instrucciones (consulta de factores primos, consulta a la base de datos, comunicación con el cliente) y datos diferentes simultáneamente.

Soporta concurrencia mediante hilos o procesos, procesando múltiples solicitudes independientes.

Clientes:

Tipo: MISD (Multiple Instruction, Single Data)

- Los clientes pueden realizar múltiples consultas paralelas (a través de un pool de consultas definido por el usuario), pero todas estas consultas apuntan a un único dato en cada operación: la cédula del ciudadano.
- El uso de threads o conexiones paralelas para interactuar con el servidor permite este comportamiento.

ESTILOS DE ARQUITECTURA:

• Arquitectura Cliente-Servidor:

El sistema se organiza en nodos cliente y servidor. Los clientes solicitan servicios al servidor, que procesa y devuelve respuestas.

Componentes involucrados: Client, Master, Worker.

• Arquitectura de Microservicios:

El sistema se divide en servicios pequeños e independientes que se comunican entre sí.

Componentes involucrados:

Cada nodo puede considerarse como un microservicio que realiza una función específica.

• Arquitectura Basada en Eventos:

Mediador de Eventos (Event Broker):

- componente central (broker) para gestionar la distribución de eventos. Este componente:
    - Recibe eventos del publicador.
    - Notifica a los suscriptores interesados en esos eventos.

Publicador (Servidor):

Publica eventos relacionados con el estado de las consultas, como el resultado de las búsquedas o la asignación de cargas de trabajo.

Suscriptores (Clientes):

Reciben los eventos publicados y actúan en consecuencia, como ejecutar las consultas asignadas o mostrar resultados al usuario.

**PATRONES DE DISEÑO**

• Patrón Master-Worker:

Divide una tarea en subtareas y distribuye estas a los trabajadores para su procesamiento. El maestro recopila los resultados y los combina.

Componentes involucrados: Client (maestro), Worker (trabajadores).

• Patrón Observer (Variación Sub-Pub):

Publicador (Publisher):

Es el sujeto que genera eventos o cambios de estado.

En el caso del sistema electoral, el servidor actúa como publicador, emitiendo notificaciones sobre las consultas realizadas, el reparto de trabajo, o los resultados.

Suscriptor (Subscriber):

Son los observadores interesados en ciertos eventos o cambios de estado.

En el sistema, los clientes son los suscriptores, registrados para recibir notificaciones sobre las consultas que deben ejecutar o los resultados.

Broker de Mensajes (Event Broker):

Es un intermediario que permite la comunicación entre publicadores y suscriptores.

Desacopla completamente a los publicadores de los suscriptores, de forma que:

El publicador solo envía eventos al broker.

El broker distribuye estos eventos a los suscriptores interesados.

Patrón Thread Pool

Gestiona un conjunto de hilos reutilizables para ejecutar tareas concurrentemente, mejorando la eficiencia y el control sobre la ejecución de hilos.

Patrón Proxy: Proporciona un sustituto o marcador de posición para controlar el acceso a otro objeto.

Mapeo de Componentes a Patrones/Estilos

1. Cliente-Servidor:

Client (Cliente)

Server(Server)

Worker (Servidor de tareas)

2. Microservicios:

Client (Microservicio de interfaz de usuario)

Master (Microservicio de coordinación de tareas)

Worker (Microservicio de procesamiento de tareas)

(ThreadPool para manejo de tareas)

4. Observer:

Relación dinámicamente establecida: Los observadores se pueden registrar y eliminar en tiempo de ejecución.

Desacoplamiento: El sujeto no necesita conocer la lógica de los observadores, solo notificar cambios.

Comunicaciones asincrónicas: la notificación ocurre de manera asíncrona.

5. Proxy:

Cliente (Frontend):

No accede directamente al servidor o la base de datos, sino a través de un proxy.

Por ejemplo, un cliente envía una consulta a través del proxy, que valida la solicitud y la redirige al servidor.(Broker)

Servidor (Backend):

El servidor real permanece oculto detrás del proxy, lo que refuerza la seguridad y permite gestionar el acceso de los clientes de manera centralizada.

**Estrategia para Resolver el Problema en Forma Distribuida**

División del Trabajo:

El Master recibe la solicitud de trabajo del Client y la divide en tareas más pequeñas.

Estas tareas se distribuyen entre varios Worker para su procesamiento.

Asignación de Tareas:

El Master utiliza una lista de Worker disponibles y les asigna tareas a medida que estén disponibles.

Utiliza un ThreadPool para gestionar la asignación y ejecución concurrente de las tareas.

Cada Worker recibe una tarea, la procesa y devuelve una solución parcial al Master.

**Recolección de Resultados:**

El Master recolecta todas las soluciones parciales de los Worker.

Una vez que todas las tareas están completas, el Master combina las soluciones parciales en una solución final.

**Notificación de Resultados:**

El Master envía la solución final de vuelta al Client

El Client muestra el resultado al usuario final