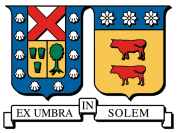

Universidad Técnica Federico Santa María
Sistemas Distribuidos
Primer Semestre 2025

Laboratorio 2: Sistemas Distribuidos

Torneo Pokémon

Profesor: Jorge Díaz
Ayudantes de Lab: Felipe Marchant V. & Maximiliano Tapia C.

Abril 2025

1 Objetivos del Laboratorio

- Comprender y aplicar la comunicación en sistemas distribuidos mediante el uso de **gRPC** en Golang.
- Familiarizarse y hacer uso de la comunicación asíncrona por medio de **RabbitMQ**
- Profundizar el uso de **Golang**
- Profundizar el uso de **Docker**

2 Introducción

El laboratorio introduce los sistemas distribuidos, formados por múltiples computadoras que se comunican para alcanzar un objetivo común. Se plantea un problema que requiere implementar comunicación síncrona (con gRPC) y asíncrona (con RabbitMQ) para resolverlo. También se proporciona documentación sobre estas tecnologías.

3 Tecnologías

- El lenguaje de programación a utilizar es **Go**.
- Para la comunicación síncrona se utilizará **gRPC**.
- Para la comunicación asíncrona se usará **RabbitMQ**
- Para la definición de mensajes síncronos se usará **ProtocolBuffers**
- Para distribuir el programa se utilizará **Docker**

4 Laboratorio

4.1 Contexto

El universo Pokémon ha evolucionado hacia una estructura de competencia interregional, donde cientos de entrenadores participan en torneos organizados por gimnasios distribuidos a lo largo de diversas regiones como Kanto, Johto, Sinnoh, entre otras. Para coordinar estas batallas de manera eficiente, la Liga Central Pokémon (LCP) ha establecido un sistema distribuido que permite el registro, ejecución y validación de combates Pokémon en tiempo real.
Cada gimnasio regional opera de forma autónoma, ejecutando combates y reportando resultados. Para mantener la integridad de la competencia y evitar fraudes, estos resultados deben ser cifrados antes de ser enviados al Centro de Datos Pokémon (CDP), el cual valida y reporta oficialmente los resultados a la LCP. Este flujo requiere tanto comunicación síncrona (gRPC) para operaciones críticas como la inscripción y actualización de rankings, como comunicación asíncrona (RabbitMQ) para el envío de resultados y notificaciones.
Debido al aumento del número de torneos y reportes fraudulentos en versiones anteriores, se ha decidido incorporar **cifrado AES-256** y un sistema de verificación en múltiples pasos, que asegure la confiabilidad de los datos. Además, se ha implementado un **Sistema de Notificaciones Pokémon (SNP)** para que los entrenadores reciban actualizaciones en tiempo real sobre cambios en su ranking, penalizaciones y nuevos torneos disponibles.

4.2 Explicación

En el mundo competitivo Pokémon, los combates entre entrenadores no solo representan una prueba de habilidades estratégicas, sino también una compleja red de validación, reporte y sincronización de resultados. Con el auge de los torneos interregionales, se vuelve esencial garantizar que los resultados de cada combate sean **auténticos, verificables y seguros**, para lo cual se ha implementado un sistema distribuido con múltiples entidades cooperando en tiempo real.
Cuando un entrenador se inscribe en un torneo a través de la **Liga Central Pokémon (LCP)**, es asignado a un combate organizado por un gimnasio certificado de su región. El combate se ejecuta localmente, y el **gimnasio genera un resultado cifrado** usando AES-256 con una clave compartida con el **Centro de Datos Pokémon (CDP)**. Este resultado se envía de forma asíncrona a través de **RabbitMQ**, garantizando una separación clara entre la ejecución del combate y la validación posterior del resultado.
Una vez recibido el mensaje, el **CDP desencripta, valida y verifica** que el resultado no sea fraudulento ni duplicado. Si el resultado es válido, este es enviado mediante comunicación asíncrona vía **RabbitMQ** a la LCP, que lo utiliza para actualizar los rankings oficiales de los entrenadores.
Paralelamente, el **Sistema de Notificaciones Pokémon (SNP)** escucha los eventos generados por la LCP y se encarga de **publicar notificaciones en tiempo real** hacia los entrenadores, informándoles sobre cambios en su ranking, nuevas penalizaciones o la apertura de nuevos torneos.
Este laboratorio pone énfasis en la implementación de **comunicación mixta (síncrona y asíncrona)** entre entidades, la **protección de datos cifrados**, la **integridad del sistema ante errores o fraudes**, y la **escalabilidad del diseño**. Los estudiantes deberán modelar los flujos de información y los eventos clave del torneo, asegurando que cada entidad cumpla su rol de forma coordinada y eficiente.

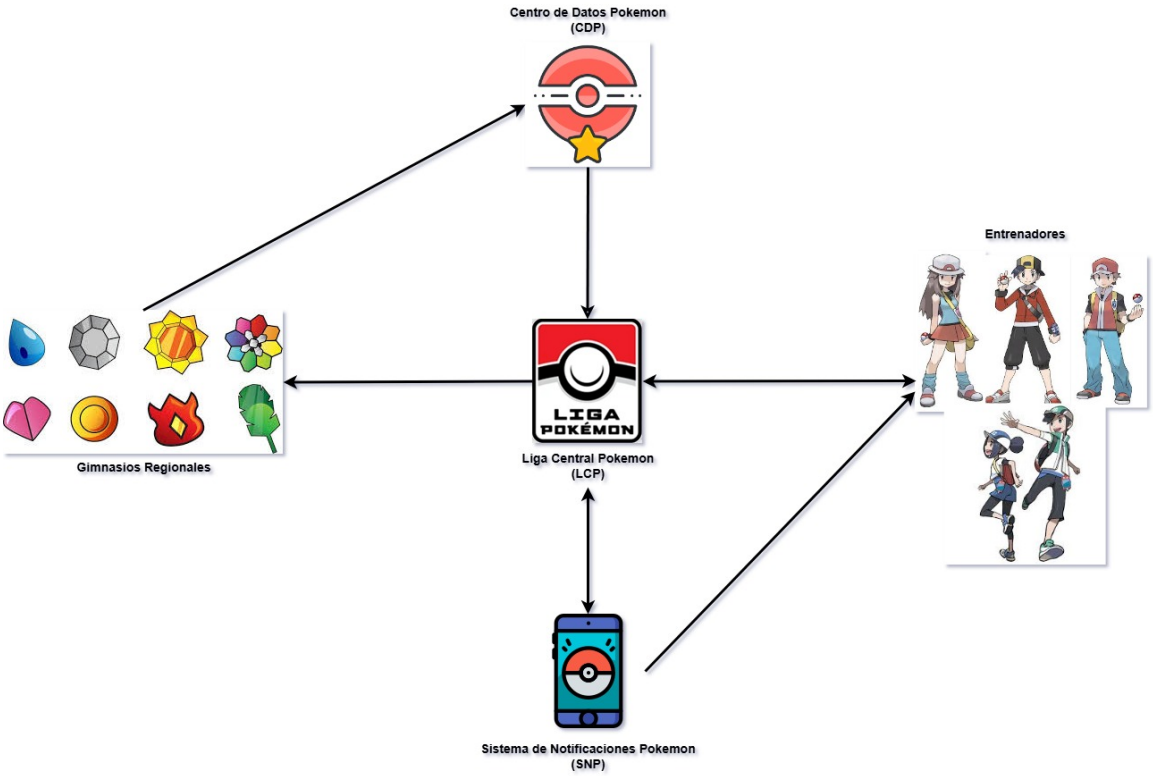

Figure 1: Entidades y sus interacciones con el entorno.

4.3 Liga Central Pokémon (LCP)

La **Liga Central Pokémon (LCP)** es la entidad central que regula y coordina los torneos oficiales entre entrenadores. Tiene como misión principal asegurar que los procesos de inscripción, validación de combates y actualización de rankings se realicen de forma justa, segura y sincronizada.

Actuando como núcleo del sistema distribuido, la LCP interactúa tanto con entrenadores como con el **Centro de Datos Pokémon (CDP)**, y también se comunica con el **Sistema de Notificaciones Pokémon (SNP)** para mantener informados a los participantes.

Funciones de la LCP:

- **Publicación de Torneos**: Genera y publica periódicamente listas de torneos disponibles para inscripción. Cada torneo posee un identificador único y puede estar asociado a una región específica.
- **Inscripción de Entrenadores**: Permite que entrenadores consulten e ingresen a torneos activos mediante solicitudes gRPC. Valida que el entrenador no esté inscrito en múltiples torneos simultáneamente.
- **Recepción de Resultados**: Recibe los resultados verificados de los combates desde el CDP vía RabbitMQ. Valida su estructura y determina si el resultado es nuevo, duplicado o inválido.
- **Actualización de Rankings**: Modifica el ranking oficial de entrenadores según los resultados recibidos, incrementando o disminuyendo puntuaciones dependiendo del desempeño.
- **Emisión de Eventos al SNP**: Informa al Sistema de Notificaciones sobre cambios importantes, como variaciones de ranking, penalizaciones o disponibilidad de nuevos torneos.
- **Rechazo de Resultados Inválidos**: Si se detecta un resultado corrupto, duplicado o asociado a entrenadores no registrados, la LCP lo rechaza y lo reporta para auditoría.

Registro de Entrenadores:
La LCP mantiene un registro en memoria con la información básica de cada entrenador, incluyendo:

ID Entrenador	Nombre	Región	Ranking	Estado	Suspensión
001	Ash Ketchum	Kanto	1570	Activo	0
002	Misty	Kanto	1490	Expulsado	0
003	Gary Oak	Johto	1535	Suspendido	3

Table 1: Ejemplo de registro de entrenadores activos en la LCP

Cada entrenador tiene un estado que refleja si puede participar en torneos, así como su ranking actual, el cual se ajusta en tiempo real según los resultados oficiales validados por el sistema. Los valores posibles del estado son:

- **Activo**: El entrenador puede inscribirse y participar en torneos sin ningún problema.
- **Suspendido**: El entrenador fue suspendido de la participación de torneos, por lo que no se podrá inscribir en los siguientes N torneos publicados en total (cualquier región), independiente de su Región. La cantidad de torneos en los que **NO** podrá participar se encuentran la columna de **Suspensión**.

- **Expulsado**: El entrenador **NO** puede participar de ningún torneo posterior. Se encuentra penalizado **PERMANENTEMENTE**.

La columna **Suspensión** indica el número de torneos restantes que el entrenador no podrá disputar (solo aplica si su estado es **Suspendido**).

Regla de Suspensión:

- Si un entrenador **Suspendido** intenta inscribirse en un torneo:
  1. Se decrementa su contador de Suspensión en 1.
  2. Si Suspensión == 0, su estado cambia a **Activo** y puede inscribirse.
  3. Si Suspensión > 0, se rechaza la inscripción y se notifica al SNP.
- **Ejemplo**: Gary (Suspendido, 3) intenta inscribirse → Suspensión = 2, rechazado.

4.4 Entrenador Pokémon (Consola)

El **Entrenador Pokémon** es el protagonista del sistema, implementado como una aplicación de consola interactiva. Este programa permite a un jugador humano interactuar con el sistema distribuido a través de un menú de terminal, mientras que los demás entrenadores son gestionados automáticamente mediante un archivo externo.

Funciones del Entrenador (Consola):

- **Menú Interactivo**: Interfaz por terminal con las siguientes opciones:
  - Consultar torneos disponibles (vía gRPC con LCP)
  - Inscribirse en un torneo seleccionado
  - Ver notificaciones recibidas (desde RabbitMQ)
  - Ver estado actual (ranking, penalizaciones)
  - Salir del programa
- **Comunicación con la LCP**:
  - Consulta síncrona de torneos disponibles mediante gRPC
  - Inscripción en torneos seleccionados por el usuario
- **Recepción de Notificaciones**:
  - Escucha pasiva de mensajes RabbitMQ del SNP
  - Muestra alertas sobre:
    - Cambios en su ranking
    - Penalizaciones
    - Nuevos torneos disponibles
- **Historial Local**: Almacena un registro local de sus participaciones y resultados

Implementación Técnica:

- Programa autocontenido que se ejecuta en un contenedor Docker
- Interfaz de línea de comandos con formato claro
- Suscripción a cola RabbitMQ personalizada para recibir notificaciones
- Conexión gRPC con la LCP para operaciones síncronas
- Los demás entrenadores son gestionados por un archivo JSON externo que se cargará al iniciar el sistema

Listing 1: Ejemplo de archivo JSON para entrenadores automáticos

```
[
  {
    "id": "001",
    "nombre": "Ash Ketchum",
    "region": "Kanto",
    "ranking": 1570,
    "estado": "Activo",
    "suspension": 0
  },
  {
    "id": "002",
    "nombre": "Misty",
    "region": "Kanto",
    "ranking": 1490,
    "estado": "Expulsado",
    "suspension": 0
  }
]
```

Flujo de Operación:

1. El programa inicia y muestra el menú principal
2. El usuario selecciona una opción del menú
3. Para consultas e inscripciones:
    - Se establece conexión gRPC con la LCP
    - Se muestran los resultados al usuario
4. Para notificaciones:
    - Escucha continua en segundo plano de RabbitMQ
    - Muestra alertas cuando llegan mensajes nuevos
5. El historial se guarda localmente en un archivo JSON

Consideraciones de Diseño:

- El entrenador consola debe manejar errores de conexión elegante
- Debe mostrar claramente las opciones disponibles
- Las notificaciones deben aparecer sin interrumpir la navegación del menú
- El estado del entrenador debe persistir entre ejecuciones

ID Entrenador	Nombre	Torneo ID	Resultado	Ranking Antes	Ranking Después
001	Ash Ketchum	05	Victoria	1520	1570
001	Ash Ketchum	06	Derrota	1570	1540
003	Gary Oak	06	Victoria	1500	1535
002	Misty	05	Derrota	1510	1490

Table 2: Ejemplo de historial competitivo de entrenadores en torneos oficiales

Estados de un Entrenador Durante un Torneo:
Los entrenadores, al participar de un torneo, pueden encontrarse en distintos estados según el flujo del evento:

- **Inscrito**: Ha sido registrado oficialmente en un torneo.
- **En Combate**: Se encuentra participando activamente en una batalla.
- **Esperando Resultado**: Su combate ya se ejecutó, pero el resultado está siendo procesado.
- **Finalizado**: Su participación ha terminado y el resultado ha sido validado.
- **Penalizado**: Ha cometido una falta (abandono, resultado inválido, etc.) y ha recibido una penalización.

4.5 Gimnasios Regionales

Los **Gimnasios Regionales** son entidades autónomas distribuidas a lo largo del mundo Pokémon. Su función principal es organizar y ejecutar combates entre entrenadores asignados por la Liga Central Pokémon (LCP). Cada gimnasio tiene la capacidad de operar localmente de forma independiente, pero debe integrarse con el sistema central para mantener la coherencia y validez de los resultados.

Estos gimnasios son responsables de generar resultados confiables, los cuales deben ser cifrados antes de ser enviados al Centro de Datos Pokémon (CDP) mediante comunicación asíncrona usando RabbitMQ.

Funciones de los Gimnasios:

- **Recepción de Combates**: Reciben asignaciones de combates directamente desde la LCP a través de gRPC, incluyendo información sobre los entrenadores, el torneo y el ID del combate.
- **Ejecución del Combate**: Simulan localmente la batalla entre los entrenadores registrados, determinando al ganador de manera pseudoaleatoria.

- **Generación de Resultados Cifrados**:
  - Una vez terminado el combate, el gimnasio genera una estructura de resultado.
  - El resultado se cifra utilizando **AES-256**, con una clave compartida con el CDP.
  - El mensaje cifrado se publica en un canal de **RabbitMQ**, destinado a ser consumido por el CDP.
  - Cada mensaje incluirá un `combate_id` único para garantizar idempotencia. El CDP descartará mensajes con IDs repetidos.
- **Gestión de Errores**:
  - En caso de fallos en la generación o envío del mensaje, el gimnasio debe reintentar tres veces el proceso, si no se logra, se debe loguear el error para revisión posterior.
  - Debe garantizar que cada combate se reporte exactamente una vez (idempotencia).

Listing 2: Estructura del mensaje de resultado antes del cifrado

```
{
  "torneo_id": 3,
  "id_entrenador_1": "001",
  "nombre_entrenador_1": "Ash",
  "id_entrenador_2": "003",
  "nombre_entrenador_2": "Gary",
  "id_ganador": "001",
  "nombre_ganador": "Ash",
  "fecha": "2024-10-23",
  "tipo_mensaje": "resultado_combate"
}
```

Consideraciones de Seguridad:

- Cada gimnasio tiene una **clave AES-256 única**, almacenada en una variable de entorno en su contenedor Docker.
- El CDP debe tener acceso a todas las claves de los gimnasios.
- Los resultados **no deben ser manipulables ni accesibles por entrenadores** u otras entidades externas.
- Las claves AES-256 se compartirán mediante variables de entorno en los contenedores Docker, garantizando que cada gimnasio tenga su clave única preconfigurada.

Consideraciones extra:

- Se debe contar con al menos **3 Gimnasios Regionales**.
- Todos los Gimnasios Regionales deben estar implementados dentro de la entidad "Gimnasios Regionales", por lo que se recomienda crear un arreglo de gimnasios dentro de la entidad.

4.6 Centro de Datos Pokémon (CDP)

El **Centro de Datos Pokémon (CDP)** es la entidad encargada de recibir, desencriptar y validar los resultados de combates enviados por los Gimnasios Regionales. Su rol es fundamental para garantizar la **integridad, autenticidad y unicidad** de los datos antes de que estos sean reportados oficialmente a la **Liga Central Pokémon (LCP)**.

Además de ser responsable de la seguridad de los datos cifrados, el CDP debe manejar la comunicación asíncrona y tolerar posibles errores o pérdida de mensajes durante la transmisión desde los gimnasios.

Funciones del CDP:

- **Consumo de Resultados Cifrados**:
  - Escucha continuamente una cola de mensajes en **RabbitMQ**.
  - Extrae los mensajes cifrados enviados por los gimnasios.
- **Desencriptación Segura**:
  - Utiliza **AES-256** con una clave previamente compartida con cada gimnasio.
- **Validación del Mensaje**:
  - Verifica que el mensaje tenga una estructura válida (campos esperados).
  - Comprueba que no se haya recibido previamente (verificación contra duplicados).
  - Asegura que ambos entrenadores estén registrados y activos en la LCP.
- **Reporte a la LCP**:
  - Si el mensaje es válido, lo transmite mediante **RabbitMQ** a la Liga Central Pokémon.
  - Si es inválido o duplicado, se registra como fallo.
- **Gestión de Errores**:
  - Aplica reintentos limitados para asegurar la entrega del mensaje procesado.

Validación de Entrenadores:

- El CDP debe enviar una solicitud gRPC a la LCP para confirmar que:
  1. Ambos entrenadores existen.
  2. Su estado es **Activo**.
- Si la validación falla, el resultado se rechaza y se notifica al SNP.

Flujo General de Procesamiento:

1. Mensaje cifrado es recibido desde RabbitMQ.
2. El mensaje es desencriptado con la clave AES correspondiente.
3. Se valida el contenido del mensaje.
4. El resultado es enviado a la LCP por RabbitMQ.
5. Se registra el evento (aceptado o rechazado) para trazabilidad.

4.7 Sistema de Notificaciones Pokémon (SNP)

El **Sistema de Notificaciones Pokémon (SNP)** es el componente encargado de mantener a los entrenadores informados en tiempo real sobre eventos importantes del sistema. Su función es asegurar la comunicación eficiente y asincrónica de eventos provenientes de la **Liga Central Pokémon (LCP)** hacia los entrenadores que están participando en torneos o activos dentro del sistema.

El SNP utiliza **RabbitMQ** para emitir mensajes que pueden ser recibidos de forma pasiva por los entrenadores suscritos, facilitando la arquitectura orientada a eventos.

Funciones del SNP

- **Escucha de Eventos del Sistema**:
  - Suscrito a eventos clave generados por la LCP (como cambios de ranking, penalizaciones o apertura de nuevos torneos).
- **Publicación de Notificaciones**:
  - Emite mensajes mediante RabbitMQ a colas correspondientes a cada entrenador o grupo de entrenadores.
  - Puede gestionar múltiples tipos de eventos (ranking actualizado, penalización, etc.).
- **Tipos de Notificaciones**:
  - **Actualización de Ranking**: Indica al entrenador su nuevo puntaje.
  - **Penalización**: Informa sobre sanciones por abandono, fraude o combate inválido.
  - **Nuevo Torneo Disponible**: Comunica la apertura de un nuevo torneo disponible para inscripción.
  - **Confirmación de Inscripción**: Mensaje automático tras el registro exitoso en un torneo.
- **Tolerancia a Fallos**:
  - Diseñado para resistir pérdida temporal de conexión o caídas de servicios de escucha.
  - Implementa reintentos y logs para asegurar la entrega eventual de mensajes.

Listing 3: Estructura del mensaje de ranking actualizado

```
{
  "tipo_mensaje": "ranking_actualizado",
  "id_entrenador": "001",
  "nombre_entrenador": "Ash",
  "nuevo_ranking": 1600,
  "fecha": "2024-10-23"
}
```

Consideraciones Técnicas:

- Los mensajes deben ser **estructurados y codificados en JSON**, y pueden contener metadatos adicionales como ID de torneo o motivo de penalización.
- Los entrenadores deben tener algún mecanismo de escucha pasiva (e.g., suscripción a su cola en RabbitMQ).

4.8 Aclaraciones Técnicas Clave

4.8.1 Asignación de Combates

- **Mecanismo**: La LCP asigna combates a los Gimnasios mediante **gRPC** usando el método `AsignarCombate(torneo_id, entrenador_1, entrenador_2)`. Cada gimnasio expone este servicio como parte de su API.
- **Estructura del mensaje**:

```
{
  "combate_id": "uuidv4",
  "torneo_id": "123",
  "entrenador_1": {"id": "001", "nombre": "Ash", "ranking": 1570},
  "entrenador_2": {"id": "003", "nombre": "Gary", "ranking": 1535},
  "region": "Kanto"
}
```

- **Reglas**:
  - Los combates se asignan a gimnasios de la misma región que los entrenadores.
  - La LCP garantiza que ambos entrenadores estén **Activos** y no tengan combates pendientes.

4.8.2 Validación de Fraudes en el CDP

El CDP implementa un **sistema de verificación en 3 pasos**:

1. **Validación Estructural**:
   - Campos obligatorios: `torneo_id`, `id_entrenador_1/2`, `id_ganador`, `fecha`.
   - Formato de fecha: `YYYY-MM-DD`.
2. **Reglas de Negocio**:
   - El `id_ganador` debe coincidir con uno de los entrenadores del combate.
   - Los entrenadores deben existir en la LCP y estar **Activos**.
   - El `combate_id` no debe estar registrado previamente (evita duplicados).
3. **Análisis de Patrones**:
   - Tiempos de combate anómalos (ej: menos de 2 segundos) generan alertas.

4.8.3 Generación de Resultados Pseudoaleatorios

Los Gimnasios determinan el ganador usando un **algoritmo ponderado por ranking** con normalización numérica:

- **Fórmula Normalizada**: Probabilidad de victoria del Entrenador A:

$$P(A) = \frac{1}{1 + e^{-(R_A - R_B)/k}}$$ (donde R_A, R_B son rankings y $k = 100$ es factor de escala)

- **Ejemplo**: Ash (1570) vs. Gary (1535):

$$P(\text{Ash}) = \frac{1}{1 + e^{-(1570-1535)/100}} \approx 58.7\%$$

- **Implementación Optimizada**:

```
func SimularCombate(ent1, ent2 Entrenador) string {
    diff := float64(ent1.Ranking - ent2.Ranking)
    k := 100.0 // Factor de escala
    prob := 1.0 / (1.0 + math.Exp(-diff/k))

    if rand.Float64() <= prob {
        return ent1.ID
    }
    return ent2.ID
}
```

- **Notas Técnicas**:
  - Usa `math.Exp` con valores acotados ($|\frac{diff}{k}| < 5$ para la mayoría de casos)
  - El factor $k = 100$ hace que diferencias de 100 puntos den $\approx 73\%$ de probabilidad
  - Para diferencias > 300 puntos, se puede retornar directamente al mejor rankeado

5 Uso de Docker

Todos los procesos del laboratorio deben ejecutarse en contenedores Docker dentro de las máquinas virtuales. Cada entidad, como la LCP, Entrenadores Pokémon, Gimnasios Regionales, CDP y SNP, deben estar aislados en contenedores separados.

Ejemplo de distribución de las entidades en las máquinas virtuales:

- MV1: LCP
- MV2: Entrenadores / CDP
- MV3: Gimnasios Regionales
- MV4: SNP

6 Restricciones

Las librerías de Golang permitidas son:

- `time`
- `strconv`
- `strings`
- `math`
- `math/rand`
- `net`
- `context`
- `fmt`
- `log`
- `os`
- `os/signal`
- `sync`
- `bufio`
- `grpc`
- `amqp`
- `encoding/json`
- `crypto/aes`
- `crypto/cipher`
- `encoding/base64`

Todo uso de librerías externas que no se han mencionado en el enunciado debe ser consultado con los ayudantes.

7 Consideraciones

- Se debe garantizar que los **mensajes asíncronos (RabbitMQ)** puedan tolerar:
  - Pérdida temporal de conexión.
  - Reintentos de entrega.
  - Verificación de duplicados por parte del CDP.
- Todos los **resultados deben ser cifrados correctamente** antes de su publicación en RabbitMQ.
- El **CDP debe rechazar** cualquier resultado con errores de estructura o que no pueda ser desencriptado.
- Cada entidad debe tener su propio contenedor Docker, con comandos Makefile como:
  - `make docker-lcp`
  - `make docker-entrenadores-cdp`
  - `make docker-gimnasio`
  - `make docker-snp`
- El sistema debe generar:
  - Logs claros de eventos (combates realizados, mensajes rechazados, ranking actualizado).
  - Archivos de salida o logs opcionales para verificación posterior.
- Revisión y evaluación del laboratorio:
  - El laboratorio será evaluado en las máquinas virtuales proporcionadas, por lo que todos los archivos necesarios deben estar correctamente almacenados en ellas.
  - No se aceptarán entregas que no puedan ejecutarse desde una consola de comandos.
  - Se aplicará un descuento en la nota por cada error de ejecución o fallo en la implementación de la comunicación entre entidades.
- Consultas y soporte:
  - Se realizará una ayudantía para explicar los detalles del laboratorio y responder dudas.
  - Consultas deben realizarse en el foro habilitado en Aula o a los siguientes correos electrónicos:
    - `felipe.marchantv@usm.cl`
    - `maximiliano.tapiac@usm.cl`
  - Se responderán consultas hasta **48 horas antes de la fecha de entrega**.

8 Reglas de Entrega

- La tarea se entrega en grupos de 2 personas previamente asignados en aula.
- La fecha de entrega es el día **miércoles 28 de mayo 23:59 hrs. sin posibilidad de extensión de plazo**.
- La tarea se revisará en las máquinas virtuales, por lo que los archivos necesarios para la correcta ejecución de esta deben estar en ellas. Recuerde que el código debe estar identado, comentado, sin warnings y sin errores. **Si hay alguna entidad que no esté subida a las VMs, se calificará con nota 0 y SIN POSIBILIDAD DE RECORRECCIÓN.**
- Se aplicará un descuento de 5 puntos al total de la nota por cada Warning, Error o Problema de Ejecución.
- Además de los códigos en las máquinas virtuales y github, deberán subir un archivo comprimido que contenga todos los códigos desarrollados en carpetas separadas por entidad en formato ***.zip*** con el nombre **GrupoXX-LabY.zip**. Donde XX es el número de su grupo e Y es el número del laboratorio. Ejemplo: *Grupo08-Lab2.zip*. El no cumplimiento (no tener los archivos en el repositorio) implica un descuento del **50% de la nota final**.
- **Debe** dejar un `MAKEFILE` o similar en cada máquina virtual asignada a su grupo para la ejecución de cada entidad. Este debe manejarse de la siguiente forma:
  - `make docker-lcp`: Iniciará el código hecho en Docker para el sistema de la LCP.
  - `make docker-entrenadores-cdp`: Iniciará el código hecho en Docker para el sistema de los entrenadores y el sistema del CDP.
  - `make docker-gimnasio`: Iniciará el código hecho en Docker para los gimnasios.
  - `make docker-snp`: Iniciará el código hecho en Docker para el SNP.
- Debe dejar un **README** en la entrega asignada a su grupo con nombre y rol de cada integrante, además de la información necesaria para ejecutar los archivos. La no entrega de este archivo, o un archivo incompleto (sin nombres, roles, etc.) conlleva un descuento de 20 puntos en la nota final.
- No se aceptan entregas que no puedan ser ejecutadas desde una consola de comandos. Incumplimiento de esta regla significa nota 0.
- Cada hora o fracción de atraso se penalizará con un descuento de 10 puntos.
- El uso de **Docker** es **obligatorio**. Teniendo nota 0 aquellas implementaciones que no hagan uso de Docker.
- Copias serán evaluadas con nota 0. Serán notificadas al profesor y las autoridades pertinentes.
- **A la hora de levantar los contenedores en las máquinas virtuales, solo se debe usar un comando make, por lo que no debe ser necesario aplicar variados comandos como make clean, creación de redes, etc. (no cuentan los comandos de cambio de directorio "cd"). En resumen, deberíamos usar un solo comando make por cada VM para poder ejecutar todo correctamente.** En caso de no cumplimiento, se aplicará un descuento de 5 puntos por comando extra, junto con ser llevado inmediatamente a recorrección!

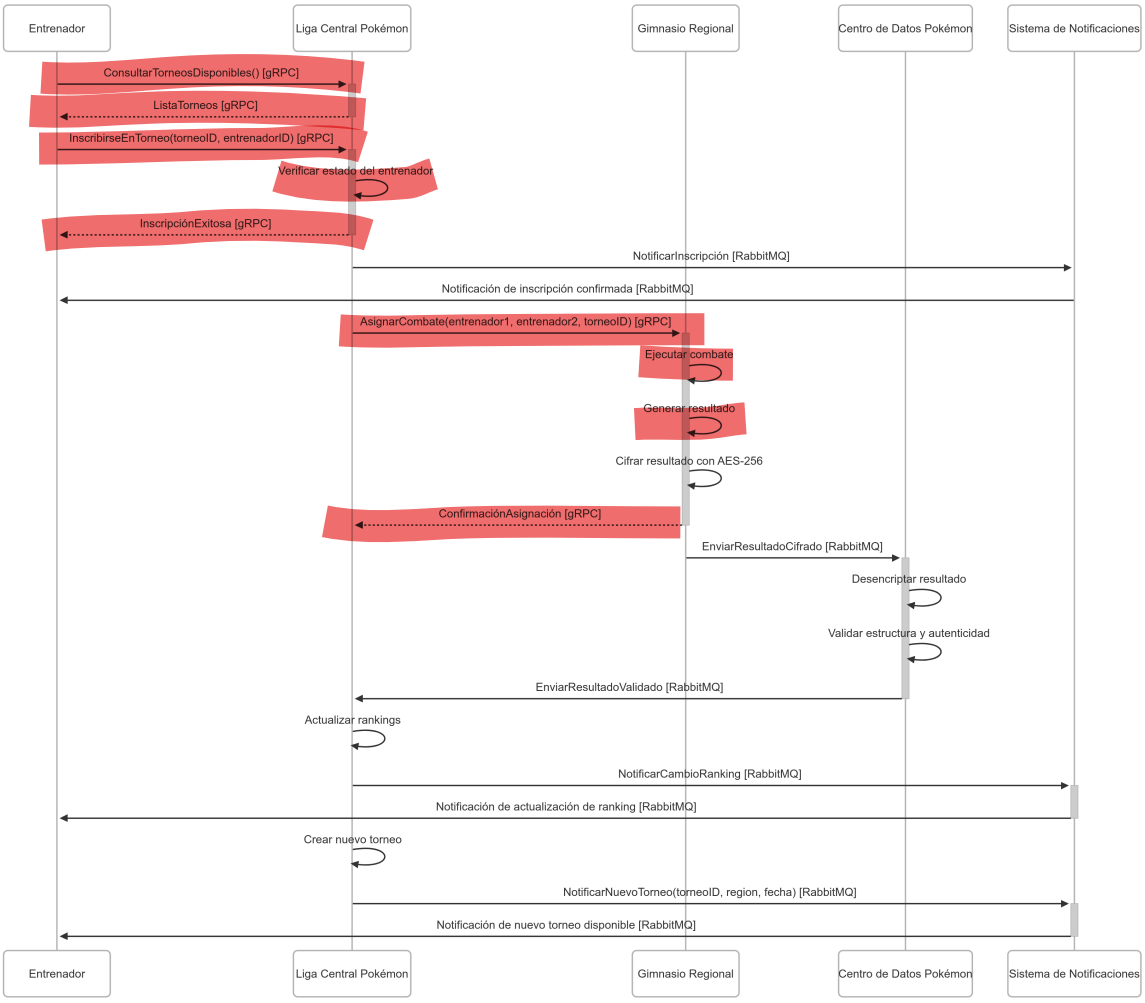

Figure 2: Diagrama de secuencia de una iteración completa en el flujo del sistema a desarrollar.