



TPO Final — **eScrims**: Plataforma de organización de scrims y partidas amistosas de eSports

Propósito: Diseñar y desarrollar (ADOO) una app móvil + backend que permita organizar scrims/partidas amistosas para distintos eSports (ej.: Valorant, LoL, CS2), gestionando emparejamientos por nivel/rango, estados del encuentro y notificaciones multi-canal.

Condiciones generales

- Armar grupos de hasta 5 (cinco) personas.
- Se debe entregar un documento PDF con todas las consignas solicitadas. El mismo deberá contener la carátula con el nombre, apellido y LU de los integrantes del equipo.
- Fechas de entrega: 11/11/25

Entrega final: ver cronograma

- En los días de entrega se debe exponer la solución propuesta. o Si bien el trabajo práctico es grupal, la evaluación es individual y lleva nota
-

1) Objetivos del sistema

- Facilitar que jugadores creen, encuentren y se unan a scrims en su región.
- Emparejar jugadores por **rango/MMR, rol y latencia**, con algoritmos intercambiables.
- Gestionar el **ciclo de vida** del scrim, desde su creación hasta el registro de estadísticas.
-
- Enviar **notificaciones** por múltiples canales (push, email, Discord/Slack) según eventos clave.

2) Alcance

- App móvil (vista usuario) y panel web (vista organizador/capitán de equipo).
 - Un **scrim** puede ser 1v1, 5v5 u otros formatos definidos por el juego.
 - Integraciones externas opcionales (autenticación OAuth de plataformas gaming, bots de Discord, proveedores de email/push).
-

3) Requerimientos Funcionales

1. Registro y autenticación de usuarios

- Alta mediante **usuario, email, contraseña**. Opcional OAuth (Steam, Riot, Discord).
- Perfil editable: **juego principal, rango** (ej.: Hierro–Radiante / Iron–Radiant), **roles** (ej.: Duelist/Support/Jungla), **servidor/región, disponibilidad horaria**.
- Verificación de email (estado: *Pendiente* → *Verificado*).

2. Búsqueda de scrims

- Filtros por **juego, formato (5v5, 3v3, 1v1), rango mínimo/máximo, región, fecha/hora, latencia máxima**.
- Guardar **búsquedas** favoritas y crear **alertas** (Observer) cuando aparezcan coincidencias.

3. Creación de scrim

- Un usuario creador define:
 - a) **Juego y formato** (cantidad de jugadores por lado).
 - b) **Cantidad total de jugadores requeridos y roles** por lado (opcional).
 - c) **Región/servidor, límites de rango** (min/max) y **latencia máxima**.
 - d) **Fecha/hora, duración estimada y modalidad** (ranked-like / casual / práctica de estratos).

e) Al crear, el scrim inicia en estado “**Buscando jugadores**”.

4. Estados del scrim (*patrón State*)

- **Buscando jugadores**: faltan plazas. Cuando se completa el cupo → “**Lobby armado**”.
- **Lobby armado**: todos los cupos llenos; cada jugador debe **confirmar**. Si todos confirman → “**Confirmado**”.
- **Confirmado**: el sistema programa cambio automático a “**En juego**” al llegar fecha/hora.
- **En juego**: una vez iniciado, puede pasar a “**Finalizado**” (cierre manual o por cron)
- **Finalizado**: se habilita **carga de resultados/estadísticas y feedback**.
- **Cancelado**: el organizador cancela antes del inicio. (Reglas para reembolsos/penalizaciones opcionales.)

Transiciones automáticas por tiempo (scheduler) y reglas de negocio.

5. Estrategias de emparejamiento (*patrón Strategy*)

- Soportar estrategias conmutables:
 - **Por rango/MMR** (diferencia máxima configurable).
 - **Por cercanía/latencia** (ping promedio dentro de umbral).
 - **Por historial/compatibilidad** (sinergia de roles, abandono previo, fair play).
- Scrim configurables para **cualquier nivel** o con **mínimo/máximo y roles obligatorios**.

6. Gestión de equipos y roles

- El organizador puede **asignar roles** y **swap** entre jugadores antes del inicio (Command).
- Sistemas de **suplentes**: si un jugador se baja, se notifica a **lista de espera**.

7. Notificaciones (*Observer + Abstract Factory/Adapter*)

- Eventos que disparan notificaciones:
 - i. Scrim creado que coincide con **preferencias** del usuario.
 - ii. Cambio a **Lobby armado** (cupos completos).
 - iii. **Confirmado** por todos.
 - iv. Cambio a **En juego, Finalizado o Cancelado**.
- Canales: **Push** (Firebase), **Email** (JavaMail/SendGrid), **Discord/Slack** (webhook/bot).

8. Estadísticas y feedback

- Al finalizar, el sistema permite **cargar resultado, MVP, kills/assists**, etc.
- **Rating** de jugadores y comentarios (con moderación básica: *pendiente/aprobado/rechazado*).

9. Moderación y penalidades

- Registro de **abandono** o **no-show**. Sistema de **strikes** y **cooldown** para reincidentes.
- Reportes de conducta (Chain of Responsibility para resolución/derivación: auto-resolver, bot, moderador humano).

10. Calendario y recordatorios

- Sincronización iCal (Adapter). Recordatorios automáticos N horas antes.

11. Multijuego y multirregión

- Un scrim siempre pertenece a un **juego** y **región/servidor**; las reglas de emparejamiento pueden variar por juego.

4) Requerimientos No Funcionales

- **Arquitectura:** seguir **MVC**. Capa de Dominio separada.
- **Patrones:** usar **al menos cuatro**; se sugiere: **State, Strategy, Observer, Abstract Factory**. Extra (opcional): **Builder, Command, Adapter, Chain of Responsibility, Template Method, Repository**.
- **Persistencia:** ORM/JPA o equivalente.
- **Escalabilidad:** colas para notificaciones (ej.: RabbitMQ/Kafka, simulado si no se implementa).
- **Disponibilidad:** manejo de fallos de proveedores de notificación con reintentos exponenciales.
- **Seguridad:** hashing de contraseñas, roles (USER, MOD, ADMIN), rate limiting básico.
- **Rendimiento:** emparejamiento debe ejecutarse en < 2s para lotes de 500 candidatos.
- **Trazabilidad:** logs de auditoría para cambios de estado y acciones de moderación.
- **Testing:** unit tests, tests de integración para estrategias, y tests de estado.

5) Patrones de Diseño (mapa sugerido)

- **State:** ciclo de vida del Scrim (Buscando, LobbyArmado, Confirmado, EnJuego, Finalizado, Cancelado).
- **Strategy:** algoritmos de emparejamiento (MMR, latencia, historial/compatibilidad).
- **Observer:** suscriptores a eventos de dominio → despachan notificaciones.
- **Abstract Factory:** creación de **Notifiers** por canal/entorno (dev/prod) y por región.
- **Builder** (*opcional*): armado incremental de Scrim con validaciones.
- **Command** (*opcional*): acciones como AsignarRol, InvitarJugador, SwapJugadores, con **undo** antes de confirmar.
- **Adapter** (*opcional*): integración con Discord/Slack/SendGrid/iCal.
- **Chain of Responsibility** (*opcional*): pipeline de moderación de reportes.

- **Template Method** (*opcional*): validación por juego (distintas reglas de composición de equipos).
-

6) Modelo de Dominio (sugerido)

- **Usuario**(id, username, email, passwordHash, rangoPorJuego, rolesPreferidos, region, preferencias)
 - **Scrim**(id, juego, formato, region, rangoMin, rangoMax, latenciaMax, fechaHora, duracion, estado, cupos, reglasRoles)
 - **Equipo**(id, lado, jugadores[0..n])
 - **Postulacion**(id, usuario, scrim, rolDeseado, estado: Pendiente/Aceptada/Rechazada)
 - **Confirmacion**(id, usuario, scrim, confirmado: bool)
 - **Notificacion**(id, tipo, canal, payload, estado)
 - **Estadistica**(id, scrim, usuario, mvp, kda, observaciones)
 - **ReporteConducta**(id, scrim, reportado, motivo, estado, sancion)
-

7) Casos de Uso (resumen)

1. **CU1 – Registrar usuario**
2. **CU2 – Autenticar usuario**
3. **CU3 – Crear scrim**
4. **CU4 – Postularse a scrim**
5. **CU5 – Emparejar y armar lobby** (auto/manual)
6. **CU6 – Confirmar participación**
7. **CU7 – Iniciar scrim (scheduler)**
8. **CU8 – Finalizar y cargar estadísticas**
9. **CU9 – Cancelar scrim**
10. **CU10 – Notificar eventos**
11. **CU11 – Moderar reportes**

Cada CU debe incluir: **Actores, Precondiciones, Flujo principal, Flujos alternativos, Reglas de negocio, Postcondiciones.**

8) Diagrama de Estados (texto)

- Buscando → (cupos completos) → LobbyArmado
- LobbyArmado → (todos confirman) → Confirmado
- Confirmado → (fechaHora alcanzada) → EnJuego

- EnJuego → (fin) → Finalizado
 - Cualquier estado **antes** de EnJuego → (cancelar) → Cancelado
-

9) Diagrama de Clases UML (guía de componentes)

- **Contexto State:** ScrimContext con ScrimState (interface) y concreciones: BuscandoState, LobbyArmadoState, ConfirmadoState, EnJuegoState, FinalizadoState, CanceladoState.
- **Strategy:** MatchmakingStrategy (interface) con ByMMRStrategy, ByLatencyStrategy, ByHistoryStrategy.
- **Observer:** DomainEventBus (Subject), NotificationSubscriber (Observer), implementaciones: PushNotifier, EmailNotifier, DiscordNotifier.
- **Abstract Factory:** NotifierFactory → crea notifiers según canal y entorno.
- **Command:** ScrimCommand con AsignarRolCommand, InvitarJugadorCommand, SwapJugadoresCommand.
- **Builder:** ScrimBuilder con validaciones encadenadas.
- **Adapter:** DiscordAdapter, SendGridAdapter, ICalAdapter.

Entrega: incluir el diagrama con estereotipos marcando cada patrón.

10) API (sugerida)

- POST /api/auth/register
 - POST /api/auth/login
 - GET /api/scrims?juego=®ion=&rangoMin=&rangoMax=&fecha=&latenciaMax=
 - POST /api/scrims (crear)
 - POST /api/scrims/{id}/postulaciones (postularse)
 - POST /api/scrims/{id}/confirmaciones (confirmar)
 - POST /api/scrims/{id}/acciones/{command} (Command pattern)
 - POST /api/scrims/{id}/cancelar
 - POST /api/scrims/{id}/finalizar
 - POST /api/scrims/{id}/estadisticas
-

11) Esqueleto de Código (Java, sugerido)

```
// Strategy
public interface MatchmakingStrategy {
    List<Usuario> seleccionar(List<Usuario> candidatos, Scrim scrim);
}

public class ByMMRStrategy implements MatchmakingStrategy { /* ... */ }
```

```

public class ByLatencyStrategy implements MatchmakingStrategy { /* ... */ }
public class ByHistoryStrategy implements MatchmakingStrategy { /* ... */ }

// State
public interface ScrimState {
    void postular(ScrimContext ctx, Usuario u, Rol rol);
    void confirmar(ScrimContext ctx, Usuario u);
    void iniciar(ScrimContext ctx);
    void finalizar(ScrimContext ctx);
    void cancelar(ScrimContext ctx);
}

public class ScrimContext {
    private ScrimState state;
    public void setState(ScrimState s){ this.state = s; }
    public void postular(Usuario u, Rol r){ state.postular(this, u, r); }
    // ... demás delegaciones
}

// Observer
public interface DomainEvent {}
public record ScrimStateChanged(UUID scrimId, String nuevoEstado) implements
DomainEvent {}

public interface Subscriber { void onEvent(DomainEvent e); }

public class DomainEventBus {
    private final List<Subscriber> subs = new ArrayList<>();
    public void subscribe(Subscriber s){ subs.add(s); }
    public void publish(DomainEvent e){ subs.forEach(s -> s.onEvent(e)); }
}

// Abstract Factory
public interface Notifier { void send(Notificacion n); }

public interface NotifierFactory { Notifier createPush(); Notifier
createEmail(); Notifier createChat(); }

public class DevNotifierFactory implements NotifierFactory { /* crea
fakes/loggers */ }
public class ProdNotifierFactory implements NotifierFactory { /* integra
Firebase/SendGrid/Discord */ }

// Builder
public class ScrimBuilder {
    private Scrim s = new Scrim();
    public ScrimBuilder juego(String j){ s.setJuego(j); return this; }
    public ScrimBuilder rango(int min, int max){ s.setRangoMin(min);
s.setRangoMax(max); return this; }
}

```

```

    public ScrimBuilder formato(String f){ s.setFormato(f); return this; }
    public ScrimBuilder fecha(LocalDateTime dt){ s.setFechaHora(dt); return
this; }
    public Scrim build(){ /* validar invariantes */ return s; }
}

// Command
public interface ScrimCommand { void execute(ScrimContext ctx); void
undo(ScrimContext ctx); }
public class AsignarRolCommand implements ScrimCommand { /* ... */ }

```

12) Historias de Usuario (ejemplos)

- **HU1:** Como jugador, quiero **buscar** scrims por rango y región para unirme a partidas con buen ping.
- **HU2:** Como organizador, quiero **crear** un scrim 5v5 con límites de rango para equilibrar el lobby.
- **HU3:** Como participante, quiero **recibir notificaciones** cuando el lobby se complete.
- **HU4:** Como moderador, quiero **procesar reportes** con un flujo escalonado.

Criterios de Aceptación (ej.)

- Dado un scrim con rango [Gold–Plat], cuando un **Player Silver** se postula, **entonces** el sistema **rechaza** la postulación.
 - Dado un scrim con latencia máx. 80ms, cuando un jugador promedio 120ms se postula, **entonces** no es admitido por la estrategia de latencia.
 - Dado un lobby completo, cuando **todos confirman**, **entonces** cambia a **Confirmado** y se envía notificación a todos los miembros.
-

13) Plan de Pruebas (resumen)

- **Unitarias:** ByMMRStrategyTest, ScrimStateTransitionsTest, NotifierFactoryTest.
 - **Integración:** flujo *crear* → *postular* → *armar lobby* → *confirmar* → *iniciar* → *finalizar*.
 - **E2E:** desde UI móvil: búsqueda, postulación, confirmación.
 - **Carga:** emparejamiento con 500 candidatos en < 2s.
-

14) Entregables

1. **Diagrama de clases UML** (con estereotipos de patrones).
2. **Diagrama de estados** del scrim.

3. **Modelo de dominio y casos de uso** documentados.
 4. **Código fuente** (mínimo capas: Controller–Service–Domain–Infra) y README.
 5. **Suite de tests** y evidencias (reportes).
 6. **Video demo (≤5 min)** mostrando patrones en ejecución.
-

15) Rúbrica de Evaluación (sugerida)

- Correctitud del **modelo** y justificación de patrones (10%).
 - Calidad del **diseño UML** y trazabilidad a código (10%).
 - Completitud del **ciclo de vida** (estados y transiciones) (10%).
 - **Notificaciones** y desacoplo con Abstract Factory/Adapter (10%).
 - **Tests** y calidad de código (10%).
 - Documentación y demo - funcionalidad- (10%).
 - Presentación oral (40%).
-

16) Extensiones opcionales (para bonus)

- **Matchmaking híbrido** (ponderar MMR + latencia + historial).
- **Rank decay** y recalcular de MMR por desempeño en scrims.
- **Colas** para notificaciones y retrys.
- **Sistema de reputación** con antifraude (detección de smurfing básica).

Nota: Mantener el alcance similar al enunciado original, pero con foco en eSports. Deben verse con claridad **State, Strategy, Observer, Facade** (mínimo 4 patrones).