Instituto de Formación Técnica Superior N°9

Trabajo Práctico: Proyecto de vasija dorada

Materia: Modelado y diseño de software

Profesor: Martín Duhalde

Integrantes: Silvestre Pieres Rawson Paz, Agustín Dietz, Candela Maisner, Alejandro Silva

Año y Carrera: 1° Año de Análisis de Sistemas

Índice

---

1. Introducción

Documento que modela en UML el sistema de control de acceso y juego para un casino, centrado en la máquina tragamonedas "Vasija Dorada" cuya mecánica acumula monedas en una vasija virtual y puede disparar un bonus.

2. Alcance y Supuestos

Alcance: Modelado del dominio, casos de uso, comportamiento de la máquina Vasija Dorada y componentes principales (controladores, servicios, BD, logs). No incluye implementación de cliente o servidor.

Supuestos clave:

- Usuarios con cuentas verificadas por email.
- Monedas virtuales gestionadas por el sistema (saldo por usuario).
- La máquina Vasija Dorada tiene una capacidad fija (ej. 100 monedas).
- Al llegar a la capacidad, existe una probabilidad (30%) de activar bonus.
- Auditoría inmutable de eventos críticos.

3. Requisitos

3.1 Requisitos Funcionales (selección)

- RF1: Registro de usuario con confirmación por email.
- RF2: Autenticación y recuperación de contraseña.
- RF3: RBAC: gestión de roles y permisos.
- RF4: Jugar en máquina tragamonedas (insertar moneda, jugar, mostrar resultado).
- RF5: Máquina Vasija Dorada: incrementar vasija; evaluar y activar bonus.
- RF6: Gestión administrativa: crear/editar/borrar usuarios, asignar roles.
- RF7: Auditoría: registrar eventos de acceso y juego.
- RF8: Mantenimiento por rol Técnico (recaudación, estado máquina).

3.2 Requisitos No Funcionales

- RNF1: Seguridad: cifrado de contraseñas; TLS para transporte.
- RNF2: Rendimiento: respuesta < 2s para operaciones de juego.
- RNF3: Disponibilidad: alta disponibilidad para servicio de juego.
- RNF4: Escalabilidad: horizontal para controladores de juego.
- RNF5: Trazabilidad y audit logs inmutables.

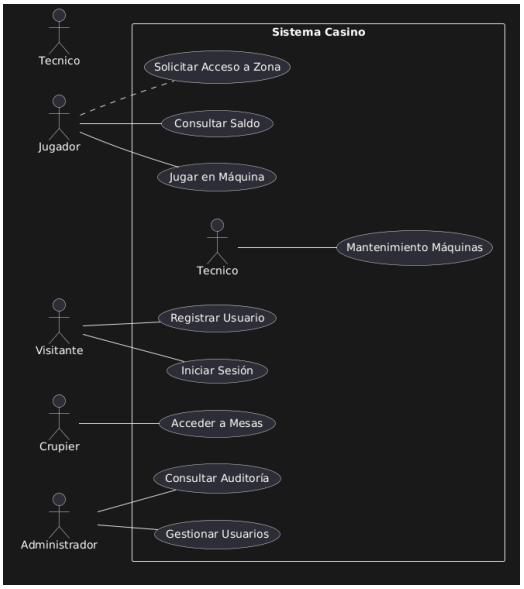
4. Actores

- Visitante
- Jugador
- Crupier
- Técnico
- Administrador
- Sistema de Juego (componente)

5. Casos de Uso

5.1 Diagrama de Casos de Uso (PlantUML)

```
@startuml
left to right direction
actor Visitante
actor Jugador
actor Crupier
actor Tecnico as Técnico
actor Administrador
rectangle "Sistema Casino" {
  Visitante -- (Registrar Usuario)
  Visitante -- (Iniciar Sesión)
  Jugador -- (Jugar en Máquina)
  Jugador -- (Consultar Saldo)
  Crupier -- (Acceder a Mesas)
  Tecnico -- (Mantenimiento Máquinas)
  Administrador -- (Gestionar Usuarios)
  Administrador -- (Consultar Auditoría)
  (Solicitar Acceso a Zona) .up. Jugador
}
@enduml
```



UC-01: Registrar Usuario

- **Actor:** Visitante
- **Precondición:** Ninguna
- **Postcondición:** Usuario creado en estado PENDIENTE_CONFIRMACION; email enviado.
  Flujo principal: Ingresar datos -> Validar -> Guardar -> Enviar email -> Confirmación por link.
- **Flujos alternativos:** Email inválido, usuario ya existe.

UC-02: Iniciar Sesión

- **Actor:** Visitante/Jugador/Administrador
- **Precondición:** Usuario registrado y confirmado
- **Postcondición:** Sesión iniciada; token JWT emitido.
- **Flujos alternativos:** Credenciales inválidas, cuenta bloqueada.

UC-03: Jugar en Máquina

- **Actor:** Jugador
- **Precondición:** Sesión activa; saldo suficiente
- **Postcondición:** Saldo actualizado; evento de juego registrado; bonus si aplica.
- **Flujos alternativos:** Saldo insuficiente; máquina en mantenimiento.

UC-04: Gestionar Usuarios

- **Actor:** Administrador
- **Precondición:** Rol Admin
- **Postcondición:** Usuarios CRUD y roles asignados
- **Flujos alternativos:** Permisos insuficientes.

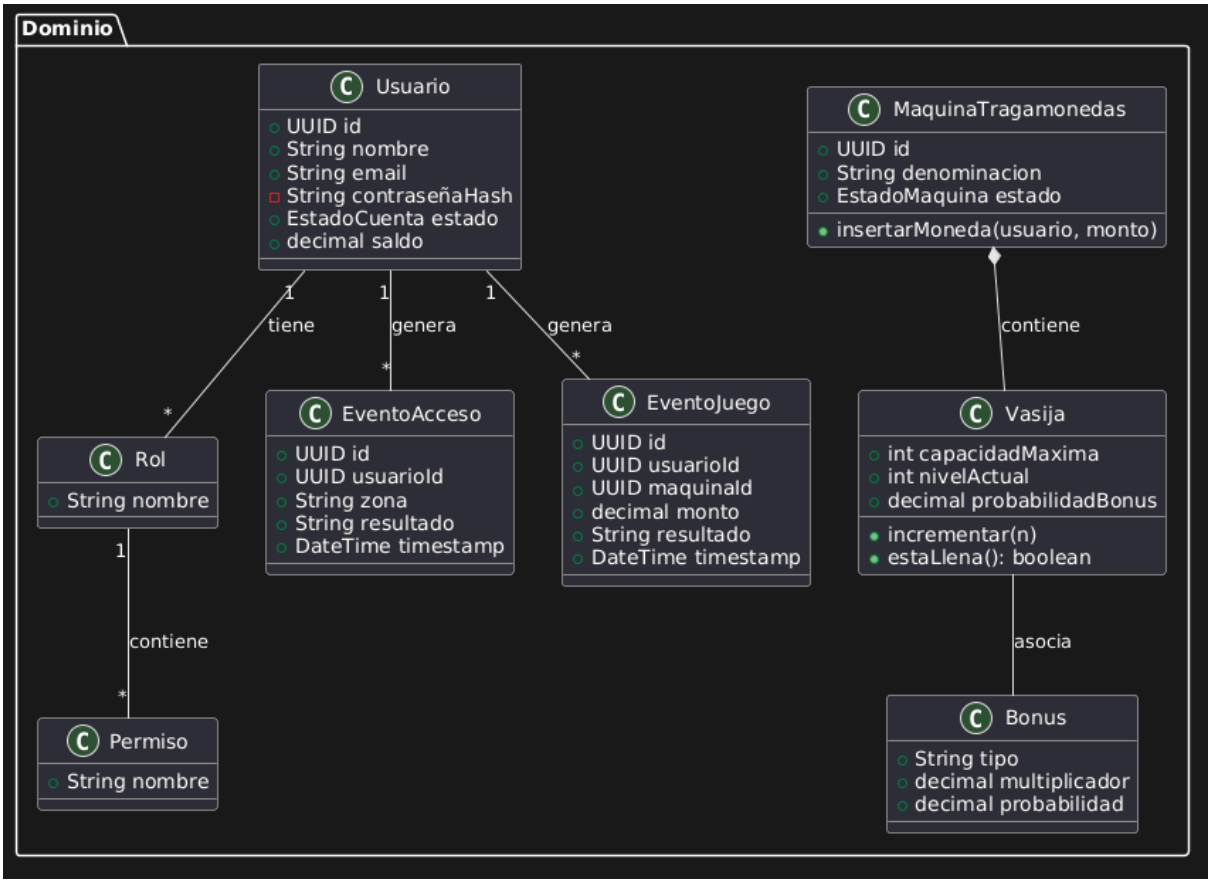
UC-05: Consultar Auditoría

- **Actor:** Administrador
- **Precondición:** Permiso Auditoría
- **Postcondición:** Listado de eventos filtrables.

6. Diagrama de Clases — Modelo de Dominio

```
@startuml
package Dominio {
  class Usuario {
    +UUID id
    +String nombre
    +String email
    -String contraseñaHash
    +EstadoCuenta estado
    +decimal saldo
  }
  class Rol {
    +String nombre
  }
  class Permiso {
    +String nombre
  }
  class MaquinaTragamonedas {
    +UUID id
    +String denominacion
    +EstadoMaquina estado
    +insertarMoneda(usuario, monto)
  }
  class Vasija {
    +int capacidadMaxima
    +int nivelActual
    +decimal probabilidadBonus
    +incrementar(n)
    +estaLlena(): boolean
  }
  class Bonus {
    +String tipo
    +decimal multiplicador
    +decimal probabilidad
  }
  class EventoAcceso {
    +UUID id
    +UUID usuarioId
    +String zona
    +String resultado
    +DateTime timestamp
  }
  class EventoJuego {
    +UUID id
    +UUID usuarioId
    +UUID maquinaId
    +decimal monto
    +String resultado
    +DateTime timestamp
  }

  Usuario "1" -- "*" Rol : tiene
  Rol "1" -- "*" Permiso : contiene
  Usuario "1" -- "*" EventoAcceso : genera
  Usuario "1" -- "*" EventoJuego : genera
  MaquinaTragamonedas *-- Vasija : contiene
  Vasija -- Bonus : asocia
}
@enduml
```



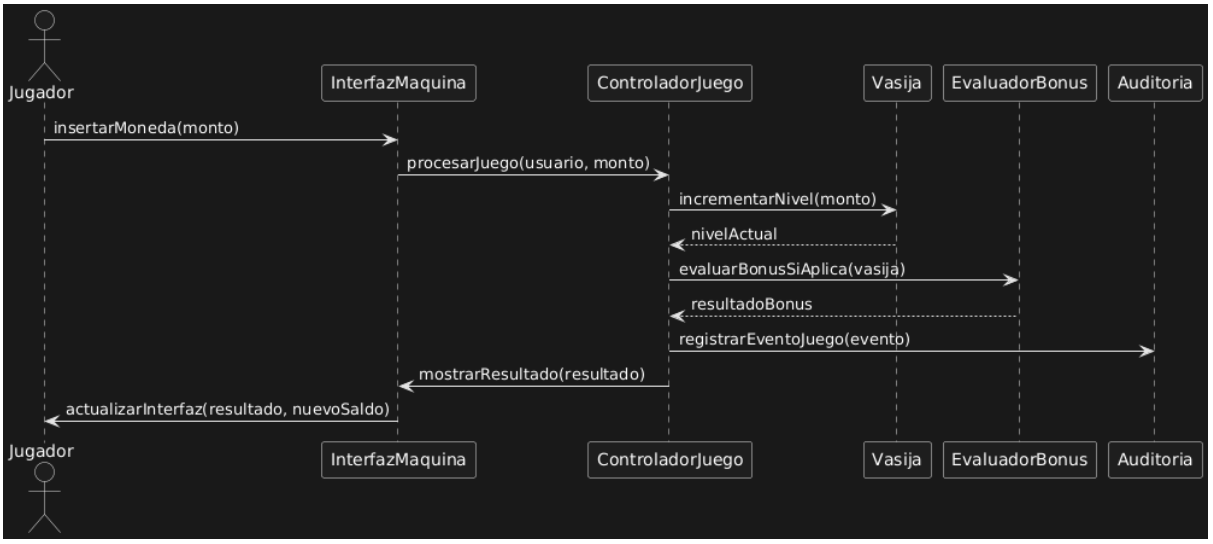
Notas de diseño de clases

- `Usuario` mantiene `saldo` y referencias a roles.
- `MaquinaTragamonedas` delega la lógica de vasija a la clase `Vasija`.
- `EventoJuego` y `EventoAcceso` se escriben en un log append-only.

7. Diagrama de Secuencia — "Jugar en Máquina Vasija Dorada"

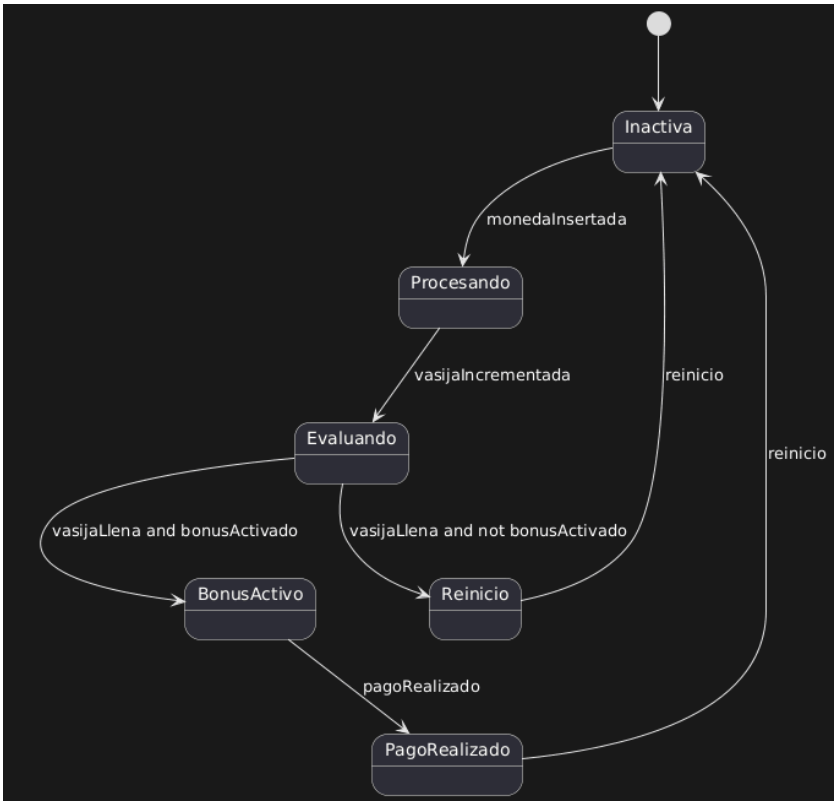
@startuml
actor Jugador
participant InterfazMaquina
participant ControladorJuego
participant Vasija
participant EvaluadorBonus
participant Auditoria

Jugador -> InterfazMaquina : insertarMoneda(monto)
InterfazMaquina -> ControladorJuego : procesarJuego(usuario, monto)
ControladorJuego -> Vasija : incrementarNivel(monto)
Vasija --> ControladorJuego : nivelActual
ControladorJuego -> EvaluadorBonus : evaluarBonusSiAplica(vasija)
EvaluadorBonus --> ControladorJuego : resultadoBonus
ControladorJuego -> Auditoria : registrarEventoJuego(evento)
ControladorJuego -> InterfazMaquina : mostrarResultado(resultado)
InterfazMaquina -> Jugador : actualizarInterfaz(resultado, nuevoSaldo)
@enduml



8. Diagrama de Estados — Máquina "Vasija Dorada"

@startuml
[*] --> Inactiva
Inactiva --> Procesando : monedaInsertada
Procesando --> Evaluando : vasijaIncrementada
Evaluando --> BonusActivo : vasijaLlena and bonusActivado
Evaluando --> Reinicio : vasijaLlena and not bonusActivado
BonusActivo --> PagoRealizado : pagoRealizado
PagoRealizado --> Inactiva : reinicio
Reinicio --> Inactiva : reinicio
@enduml

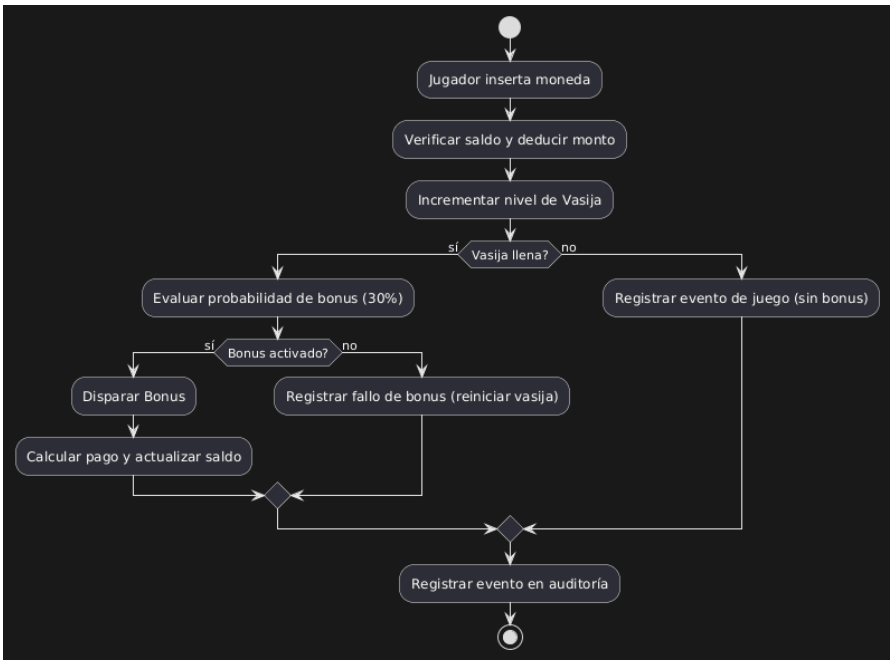


Estados explicados

- Inactiva: Espera interacción.
- Procesando: Validación e incremento de vasija.
- Evaluando: Determina si la vasija está llena; calcula probabilidad de bonus.
- BonusActivo: Ejecuta la lógica del bonus (pago, animaciones).
- PagoRealizado / Reinicio: Limpia estado y vuelve a Inactiva.

9. Diagrama de Actividades — Flujo de Juego Completo

```
@startuml
start
:Jugador inserta moneda;
:Verificar saldo y deducir monto;
:Incrementar nivel de Vasija;
if (Vasija llena?) then (sí)
  :Evaluar probabilidad de bonus (30%);
  if (Bonus activado?) then (sí)
    :Disparar Bonus;
    :Calcular pago y actualizar saldo;
  else (no)
    :Registrar fallo de bonus (reiniciar vasija);
  endif
else (no)
  :Registrar evento de juego (sin bonus);
endif
:Registrar evento en auditoría;
stop
@enduml
```



10. Componentes del Sistema (Arquitectura propuesta)

- **Frontend**: SPA (React/Vue) — InterfazJugador, InterfazAdmin, InterfazTécnico
- **API Gateway**: Autenticación, routing
- **Servicios**:
    - Servicio de Usuarios (CRUD, verificación email)
    - Servicio de Autenticación (JWT, sesiones)
    - Servicio RBAC (roles y permisos)
    - Servicio Juego / ControladorJuego (lógica de máquinas)
    - Servicio EvaluadorBonus (módulo estadístico)
    - Servicio Auditoría (append-only log, inmutable)
    - Servicio Mantenimiento (Técnico)
- **Persistencia**:
    - Base de datos relacional para dominio (Postgres)
    - Almacenamiento de eventos / audit log (EventStore o append-only table)
- **Infra**: Balanceadores, colas (RabbitMQ/Kafka) para desacoplar logs y notificaciones

11. Reglas de negocio y supuestos

- RB1: Cuando `Vasija.nivelActual >= capacidadMaxima` se evalúa el bonus.
- RB2: Probabilidad de bonus por defecto: 30% (configurable por Admin).
- RB3: El pago del bonus aplica un multiplicador definido en `Bonus`.
- RB4: Las transacciones de saldo son atómicas y auditadas.
- RB5: Solo Administrador puede cambiar configuración de máquinas y probabilidades.

12. Matriz de trazabilidad (Algunas filas de ejemplo)

Requisito	Caso de Uso	Clase(s) involucradas	Secuencia	Estado
RF1 Registro	UC-01 Registrar Usuario	Usuario, ServicioUsuarios	N/A	N/A
RF3 RBAC	UC-04 Gestionar Usuarios	Usuario, Rol, Permiso, ServicioRBAC	N/A	N/A
RF4 Jugar	UC-03 Jugar en Máquina	MaquinaTragamonedas, Vasija, EvaluadorBonus, EventoJuego	Secuencia Jugar	Estado Vasija
RF7 Auditoría	UC-05 Consultar Auditoría	EventoJuego, EventoAcceso, ServicioAuditoría	Secuencia Jugar	N/A

13. Plan de auditoría y seguridad

- **Auditoría**: Todos los eventos de acceso, transacciones de saldo, cambios de configuración y acciones administrativas se escriben con timestamp, usuario, origen IP y correlación de operación.

- **Seguridad**:

    - Hash de contraseñas con bcrypt/scrypt/argon2.
    - TLS en transporte.
    - JWT con expiración corta y refresh tokens revocables.
    - RBAC para control de operaciones sensibles.

.