

# Planificación de entregables por fase

Motor RPG data-driven (servidor, transacciones atómicas, snapshots, cálculo de stats)

## Propósito

Definir, por fase y por slice vertical, el entregable verificable esperado (documentos, JSON, OpenAPI, código y tests). Este plan es independiente de la implementación concreta y sirve como checklist de avance.

## Convenciones de artefactos

- Los artefactos se agrupan en carpetas: schemas/, examples/, openapi/, docs/, src/, tests/.
- Cada slice incluye: código + tests + actualización de OpenAPI si aplica + golden files.
- Las decisiones de semántica (equipamiento, restricciones, sets, clamp, idempotencia, durabilidad) quedan fijadas antes de consolidar APIs.

## Fase 0 - Preparación y criterios de éxito

### Entregables

- docs/00\_project\_charter.md: alcance (in/out), supuestos, definiciones y criterios de éxito por hito.
- docs/invariants.md: invariantes (ownership, atomicidad, unicidad de equipamiento, etc.).
- Repositorio inicial: estructura base + pipeline de tests + CI ejecutando tests.
- tests/test\_smoke.\*: primer test que valida el runner y el empaquetado mínimo.

## Fase 1 - Concreción final de requisitos y esquemas

### 1.1 Semántica del dominio

- docs/01\_semantics\_addendum.md: decisiones cerradas: conflicto al equipar (estricto/swap), semántica de restricciones de nivel, conteo de sets con gear multi-slot, clamp/negativos/redondeo, durabilidad (pérdida aceptable vs log), idempotencia (txId) y modelo de extensibilidad (DSL vs catálogo vs plugins).

### 1.2 Esquemas y ejemplos golden

- schemas/game\_config.schema.json
- schemas/game\_state.schema.json
- schemas/transaction.schema.json
- schemas/transaction\_result.schema.json
- examples/: config mínima, estado vacío, transacciones de ejemplo y resultados esperados (golden).
- docs/02\_migration\_policy.md: reglas best-effort config↔state (stats/slots/clases/gearDef desaparecidos).

## Fase 2 - Diseño de arquitectura y contrato de API

## Entregables

- openapi/openapi.yaml (v1): endpoint de transacciones y endpoints de lectura (config, stats, estado de jugador). Incluye modelos y ejemplos.
- docs/03\_error\_codes.md: catálogo de errorCode + significado + casos típicos.
- docs/04\_internal\_components.md: interfaces conceptuales (ConfigLoader, StateStore, TxProcessor, RulesEngine, StatsCalculator, SnapshotManager) y responsabilidades.
- (Opcional) diagrams/component\_view.\*: diagrama de componentes y dependencias.

# Slices verticales de implementación

Cada slice se considera completado cuando existen artefactos verificables y tests reproducibles. Se recomienda que cada slice publique un paquete de aceptación con secuencia de transacciones y estado final esperado.

## Slice 1 - Instancia + carga de config + health + snapshot mínimo

### Entregables

- Código (src/): servidor escucha en IP:puerto; carga GameConfig al arranque; crea/gestiona gameInstanceId; SnapshotManager guarda/carga GameState.
- OpenAPI: GET /health, GET /{gameInstanceId}/config, GET /{gameInstanceId}/stateVersion.
- Tests: arranque con config válida/ inválida; snapshot→reinicio→restore.
- Examples: examples/config\_minimal.json, examples/state\_snapshot\_minimal.json.

## Slice 2 - Jugadores y ownership (invariantes)

### Entregables

- Código: transacción CreatePlayer; estructura de players en GameState; enforcement de ownership.
- OpenAPI: POST /{gameInstanceId}/tx (CreatePlayer); GET /{gameInstanceId}/state/player/{playerId}.
- Tests: no duplicar playerId; rechazo atómico ante payload inválido.
- Golden: tx\_create\_player.json + expected\_result\_create\_player.json.

## Slice 3 - Personajes (clase + nivel) + stats base escaladas

### Entregables

- Código: CreateCharacter(classId); LevelUpCharacter; StatsCalculator (solo personaje).
- OpenAPI: tx CreateCharacter/LevelUpCharacter; GET /{gameInstanceId}/character/{characterId}/stats.
- Tests: golden de crecimiento por nivel; validación classId; maxLevel.
- Golden: config\_with\_classes.json + golden\_stats\_character\_levels.json.

## Slice 4 - Gear instanciable + inventario del jugador

### Entregables

- Código: CreateGear(gearDefId); LevelUpGear; inventario del jugador (equipado vs no equipado).
- OpenAPI: tx CreateGear/LevelUpGear; (opcional) query de listado de gear por player.
- Tests: golden de crecimiento gear por nivel; ownership; validación gearDefId.
- Golden: config\_with\_geardefs.json + tx\_create\_gear.json + expected\_result\_create\_gear.json.

## Slice 5 - Equipamiento 1 slot + stats sumadas (pj + gear)

### Entregables

- Código: EquipGear (1 slot); UnequipGear; StatsCalculator suma pj +  $\Sigma$  gear equipados.
- OpenAPI: tx EquipGear/UnequipGear; stats endpoint refleja equipo.
- Tests: SLOT\_OCCUPIED; gear no puede estar en 2 personajes; atomicidad (si falla no hay cambios).
- Golden: golden\_equipped\_stats.json.

## Slice 6 - Gear de 2 slots + conflictos + swap (si aplica)

### Entregables

- Código: soporte multi-slot (ocupación exacta según gearDef); política de conflicto (rechazo o swap explícito).
- OpenAPI: si existe modo swap, tx adicional (p.ej. EquipGearSwap).
- Tests: equip 2 slots con uno ocupado; unequip libera ambos; consistencia del estado.
- Golden: golden\_two\_slot\_equip.json.

## Slice 7 - Restricciones de equipamiento

### Entregables

- Código: evaluador de restricciones (clase y nivel) integrado en EquipGear/EquipGearSwap.
- OpenAPI: errores RESTRICTION\_FAILED con detalles mínimos.
- Tests: matriz permitido/denegado por clase/level; rechazo atómico.
- Golden: golden\_restrictions\_matrix.json.

## Slice 8 - Sets (2 piezas, 4 piezas)

### Entregables

- Código: setId en gearDef; activación por umbrales; bonus sumado a stats finales.
- OpenAPI: config incluye sets; (opcional) breakdown de stats por fuente (pj/gear/set).
- Tests: activación 2/4 y edge cases; regla de conteo para multi-slot aplicada.
- Golden: golden\_set\_bonus\_2.json + golden\_set\_bonus\_4.json.

## Slice 9 - Migración best-effort config↔state

### Entregables

- Código: migrator al cargar snapshot con config distinta (stats nuevas a 0, desaparecidas ignoradas; slots desaparecidos según política; clases/gearDef faltantes según política).
- Docs: (opcional) docs/05\_migration\_report\_format.md para warnings/reportes.
- Tests: fixtures de migración (v1 state + v2 config → expected state) + invariantes post-migración.
- Golden: fixtures/migration/\*.

## Fase 4 - Extensibilidad sin tocar código

Los entregables de esta fase dependen del modelo elegido en Fase 1: DSL declarativa, catálogo parametrizable o plugins. La fase se considera completada cuando añadir un algoritmo/regla nuevo se logra únicamente modificando datos (JSON) y pasando los tests.

### Entregables mínimos comunes

- docs/06\_extensibility\_model.md: decisión tomada (DSL/catálogo/plugins) y ejemplos de uso.
- schemas/: schemas adicionales para algoritmos y reglas.
- examples/algorithms/ y examples/rules/: ejemplos funcionales.
- Tests: suite de evaluación (golden) + límites/seguridad (si aplica).

## Fase 5 - Observabilidad y operación

### Entregables

- docs/07\_operability.md: logging, métricas, configuración runtime y troubleshooting básico.
- Código: logs estructurados por transacción (aceptada/rechazada + motivo) y métricas básicas (latencia, tamaño estado, tx count).
- Tests: smoke tests de endpoints + reinicio con snapshot + carga ligera.

### Paquete de aceptación recomendado (transversal)

- docs/release\_notes\_slice\_X.md
- examples/tx\_sequence\_slice\_X.json (secuencia reproducible)
- examples/expected\_final\_state\_slice\_X.json
- Script de prueba (tests/) que ejecute la secuencia y compare estado final con el esperado.