1️⃣ OBJETIVO DEL PLAN DE PRUEBAS

Garantizar que el sistema EcoRent Norte:

- Cumple todos los **Requisitos Funcionales (RF-01 a RF-14)**
- Cumple los **Requisitos No Funcionales críticos (RNF-01 a RNF-09)**
- Implementa correctamente las **Reglas de Negocio (RN-01 a RN-05)**
- Es estable bajo uso concurrente básico (≤5 usuarios)
- No permite errores críticos como:
  - Duplicidad de reservas
  - Pagos superiores al importe
  - Reservas en mantenimiento
  - Solapamientos de fechas

El plan busca **minimizar riesgo operativo** antes de despliegue interno.

2️⃣ ESTRATEGIA DE TESTING

Se aplicará estrategia híbrida basada en **Test Pyramid**:

```
text

        E2E (pocos, críticos)
      Sistema / API
    Integración (Service + Repo)
  Unitarios (base amplia)
```

Copy

2.1 Pruebas Unitarias (Backend)

Responsables: Backend Developer + QA validación
Herramientas: JUnit 5 + Mockito

Cobertura objetivo:

- ☑ RentalService (crítico)
- ☑ PaymentService
- ☑ EquipmentService
- ☑ Validaciones solapamiento
- ☑ Validación pagos > importe

Casos críticos:

- RN-01 Validación solapamiento
- RN-02 Bloqueo mantenimiento
- RN-03 Cálculo días naturales
- RN-05 Pago superior rechazado

Objetivo cobertura:

- **≥ 80% branch coverage en Service Layer**

2.2 Pruebas de Integración (SpringBootTest)

Verifican:

- Controller + Service + Repository + H2 real
- Transacciones @Transactional
- Persistencia correcta en H2 file

Casos clave:

- Crear alquiler → actualiza estado equipo
- Registrar devolución → cambia estado
- Registrar pago → afecta estado financiero

2.3 Pruebas de Sistema (Manual + Swagger)

Validación funcional completa vía:

- Swagger UI
- Postman

Objetivo:

- Validar contrato REST real
- Validar mensajes de error (BusinessException)

2.4 Pruebas End-to-End (Frontend + Backend)

Herramienta recomendada:

- Cypress (ideal)
- Alternativa manual estructurada

Flujos críticos:

1. Crear cliente → Crear alquiler → Registrar pago → Devolver equipo
2. Intentar crear solapamiento
3. Intentar reservar equipo en mantenimiento
4. Intentar pago superior

3️⃣ ALCANCE Y EXCLUSIONES

✅ Alcance

- RF-01 a RF-14
- RN-01 a RN-05
- RNF-01 rendimiento básico
- RNF-06 usabilidad básica
- Persistencia H2 file
- Integración completa SPA ↔ REST

❌ Exclusiones

- Seguridad avanzada (actualmente permitAll)
- JWT
- Ataques de seguridad complejos
- Escalabilidad horizontal
- Carga >5 usuarios

4️⃣ CASOS DE PRUEBA FUNCIONALES

Formato profesional ejecutable.

4.1 Gestión de Equipos

ID	Caso	Precondición	Acción	Resultado Esperado
TC-EQ-01	Crear equipo válido	Usuario autenticado	POST equipo válido	Equipo creado, estado AVAILABLE

ID	Caso	Precondición	Acción	Resultado Esperado
TC-EQ-02	Código duplicado	Existe código	Crear con mismo código	Error 400 duplicidad
TC-EQ-03	Cambiar estado mantenimiento	Equipo existe	PUT status MAINTENANCE	Estado actualizado
TC-EQ-04	Reservar equipo en mantenimiento	Equipo MAINTENANCE	Crear alquiler	Error negocio

---

4.2 Gestión de Clientes

ID	Caso	Acción	Resultado
TC-CL-01	Crear cliente válido	POST cliente	Cliente guardado
TC-CL-02	Consultar historial vacío	GET rentals	Lista vacía
TC-CL-03	Consultar historial con datos	GET rentals	Lista correcta

---

4.3 Gestión de Alquileres

ID	Caso	Resultado Esperado
TC-RE-01	Crear alquiler válido	Estado equipo = RENTED
TC-RE-02	Solapamiento fechas	Error "Existe solapamiento"
TC-RE-03	Cálculo importe	total = precio × días
TC-RE-04	Devolución	returned=true y equipo AVAILABLE

4.4 Pagos

ID	Caso	Resultado
TC-PAY-01	Registrar pago parcial	Estado = Parcial
TC-PAY-02	Pago completo	Estado = Completo
TC-PAY-03	Pago superior	Error negocio

4.5 Reportes

ID	Caso	Resultado
TC-REP-01	Ingresos periodo	Suma correcta
TC-REP-02	Top equipos	Orden descendente
TC-REP-03	Top clientes	Ranking correcto

5️⃣ CASOS NO FUNCIONALES

5.1 Rendimiento (RNF-01)

Escenario:

- 5 usuarios simultáneos
- 20 consultas consecutivas GET /equipments

Criterio:

☑ Tiempo respuesta < 2 segundos

Herramienta sugerida:

- Apache JMeter

5.2 Persistencia (RNF-08)

Caso:

1. Crear datos
2. Parar servidor
3. Reiniciar
4. Verificar datos

Resultado esperado:

☑ Datos persisten en fichero H2

5.3 Usabilidad (RNF-06)

Escenario:

- Usuario entrenado crea alquiler

Tiempo máximo:

☑ < 2 minutos

Medición manual con cronómetro.

---

6 ENTORNO DE PRUEBAS

Componente	Configuración
Backend	Spring Boot 3.2.5
DB	H2 file ./data/ecorentdb
Frontend	React 19 + Vite
Navegador	Chrome 121+
OS	Windows 11
Java	JDK 21

Ambiente idéntico a producción local.

---

7 CRITERIOS DE ENTRADA Y SALIDA

✅ Entrada

- Código compilable
- Build sin errores
- Endpoints documentados en Swagger
- H2 inicializada

✅ Salida

- 100% RF probados
- 0 defectos críticos abiertos
- <5% defectos severidad media
- Cobertura ≥80% en Services
- Pruebas de regresión completadas

---

8 MÉTRICAS DE CALIDAD

Métrica	Objetivo
Cobertura servicios	≥80%
Defectos críticos	0
MTTR	<24h
Tasa defectos escapados	<5%
Tiempo medio respuesta	<2s
Flakiness E2E	<2%

---

9 RIESGOS DE TESTING

Riesgo	Impacto	Mitigación
Error lógica solapamiento	Duplicidad reservas	Tests unitarios exhaustivos
Error cálculo días	Facturación incorrecta	Tests límite fechas
H2 corrupción	Pérdida datos	Backup
Frontend no valida	UX deficiente	Validación backend obligatoria
Concurrencia	Estado inconsistente	@Transactional

---

10 PLAN DE EJECUCIÓN REAL (2 DÍAS)

Día 1

- Unit tests Services
- Integración REST
- Validación reglas negocio críticas

Día 2

- E2E críticos
- Pruebas manuales completas
- Validación rendimiento básica
- Smoke test final

✅ CONCLUSIÓN COMO QA LEAD

El sistema:

- Tiene reglas críticas bien aisladas en Service Layer
- Usa transacciones
- Usa DTOs
- Tiene manejo global de errores
- Está listo para pruebas estructuradas

Riesgo principal:
👉 Validación de solapamiento y estados financieros.

Con este plan:

✓ Se minimiza riesgo operativo
✓ Se protege reputación con clientes
✓ Se valida estabilidad antes de despliegue
✓ Se garantiza cumplimiento funcional

Si deseas, puedo ahora generar:

- ☑ Casos de prueba en formato Excel listo para importar a Jira/Xray
- ☑ Feature files Gherkin (BDD)
- ☑ Plan de pruebas automatizadas con JUnit + Cypress
- ☑ Checklist de Go-Live
- ☑ Estrategia de regresión continua