1️⃣ Decisiones Arquitectónicas Clave

✅ 1. Arquitectura Monolítica Modular

Decisión: Implementar un monolito estructurado por capas.

Justificación:

- Proyecto de 2 días.
- Única sede, baja concurrencia (≤5 usuarios).
- Simplifica despliegue (un único `.jar`).
- Reduce complejidad operativa (sin orquestación, sin red interna).
- Facilita mantenimiento por equipo pequeño.

Se diseña como monolito modular con separación clara por dominios para permitir futura evolución a microservicios si el negocio crece.

✅ 2. Arquitectura en Capas (RNF-09)

Estructura:

```
text

Controller → Service → Repository → Database
```

Copiar

Justificación:

- Cumple requisito RNF-09.
- Aísla reglas de negocio (Service).
- Facilita pruebas unitarias.
- Reduce acoplamiento.

✅ 3. Base de datos H2 en modo file

Justificación:

- Cumple RNF-08.
- Persistencia tras reinicio.
- Instalación simple (sin servidor externo).
- Ideal para entorno interno.

Diseñado con JPA para permitir migración futura a PostgreSQL/MySQL sin cambiar lógica.

✅ 4. Frontend React servido como estático desde Spring Boot

Justificación:

- Simplifica despliegue (un único artefacto).
- No requiere servidor web adicional.
- Reduce complejidad de red.
- Adecuado para aplicación interna.

2️⃣ Tipo de Arquitectura

✅ Monolito Modular + SPA (Single Page Application)

- Backend: Spring Boot (REST API)
- Frontend: React + Vite
- Base de datos embebida
- Despliegue local

3️⃣ Diagrama Lógico (descrito en texto)

```
text

[Usuario Interno]
        ↓
[React SPA - Vite]
        ↓ (Axios HTTP)
[REST Controllers - Spring Boot]
        ↓
[Service Layer]
        ↓
[Repository Layer - JPA]
        ↓
[H2 Database (modo file)]
```

Copiar

Flujo típico: Crear alquiler

1. Usuario crea alquiler en React.
2. Axios envía POST /api/rentals.
3. RentalController recibe petición.
4. RentalService:
    - Valida solapamiento (RN-01).
    - Valida estado equipo (RN-02).
    - Calcula importe (RN-03).
5. Repository guarda entidad.
6. Respuesta JSON al frontend.

4️⃣ Modelo de Datos

🟢 Entidades Principales

1️⃣ Equipment

Campo	Tipo	Descripción
id	Long	PK
name	String	Nombre
category	String	Categoría
internalCode	String (unique)	Código único
pricePerDay	BigDecimal	Precio
status	Enum	AVAILABLE / RENTED / MAINTENANCE

Relación:

- 1 Equipment → N Rentals

2️⃣ Client

Campo	Tipo
id	Long
name	String
dni	String

Campo	Tipo
phone	String
email	String

Relación:

- 1 Client → N Rentals

---

3 Rental

Campo	Tipo
id	Long
startDate	LocalDate
endDate	LocalDate
totalAmount	BigDecimal
returned	Boolean
equipment	ManyToOne
client	ManyToOne

Relación:

- 1 Rental → N Payments

---

4 Payment

Campo	Tipo
id	Long
amount	BigDecimal
paymentDate	LocalDate
rental	ManyToOne

---

5 User

Campo	Tipo
id	Long
username	String
password	String (BCrypt)

Campo	Tipo
role	Enum

5️⃣ Diseño de APIs

Base URL: `/api`

🔹 Equipment

Crear equipo

```
text

POST /api/equipments
```

Copiar

Listar equipos

```
text

GET /api/equipments
```

Copiar

Cambiar estado

```
text

PUT /api/equipments/{id}/status
```

Copiar

🔹 Client

Crear cliente

```
text

POST /api/clients
```

Copiar

Historial cliente

```
text
```

```
GET /api/clients/{id}/rentals
```
Copiar

Rental

Crear alquiler

```
text

POST /api/rentals
```
Copiar

Registrar devolución

```
text

PUT /api/rentals/{id}/return
```
Copiar

Payment

Registrar pago

```
text

POST /api/rentals/{id}/payments
```
Copiar

Reportes

Ingresos por periodo

```
text

GET /api/reports/income?start=2026-01-01&end=2026-01-31
```
Copiar

Equipos más alquilados

```
text

GET /api/reports/top-equipments
```
Copiar

---

6 Stack Tecnológico y Justificación

Componente	Tecnología	Justificación
Backend	Spring Boot	Productivo, rápido setup
Persistencia	Spring Data JPA	Abstracción DB
DB	H2 File	Ligero y persistente
Seguridad	Spring Security	Cumple RNF-03/05
Frontend	React + Vite	Rápido build
HTTP Client	Axios	Simplicidad
Build	Maven	Estándar
Deploy	java -jar	Simple operación

---

7 Estrategia de Escalabilidad

Actualmente:

- Escalabilidad vertical (más RAM/CPU).
- Hasta 5 usuarios concurrentes (RNF-01).

Diseñado para futura evolución:

☑ Separación por dominio
☑ JPA desacoplado de H2
☑ Posible migración a:

- PostgreSQL
- Docker
- Microservicios
- Despliegue cloud

---

8 Disponibilidad

Requisito RNF-02:

- Servicio ejecutado en servidor interno.
- Backup diario automático del fichero H2.
- Restart manual sencillo.

Opcional mejora:

- Script Windows service.

9️⃣ Seguridad Técnica

✅ Autenticación

- Login con JWT o sesión básica.
- Password cifrada con BCrypt.

✅ Protección endpoints

- Configuración Spring Security.
- Roles:
  - ADMIN
  - EMPLOYEE

✅ Validaciones backend obligatorias

Nunca confiar solo en frontend:

- Validación solapamiento
- Validación estado equipo
- Validación pagos > importe

🔟 Riesgos Técnicos y Mitigación

Riesgo	Impacto	Mitigación
Corrupción fichero H2	Pérdida datos	Backup diario
Error en lógica de solapamiento	Duplicidad alquiler	Test unitarios exhaustivos
Falta de experiencia React	Retraso	UI simple, sin router complejo
Concurrencia	Estado inconsistente	Transacciones @Transactional
Crecimiento futuro	Limitación monolito	Diseño modular preparado

---

1️⃣1️⃣ Cumplimiento con Requisitos

☑ RF-01 a RF-14 cubiertos por módulos
☑ RNF-01 rendimiento simple
☑ RNF-03 seguridad implementada
☑ RNF-08 persistencia file
☑ RNF-09 arquitectura en capas

1️⃣ 2️⃣ Roadmap Técnico Realista (2 días)

Día 1 – Backend completo

- Entidades JPA
- Repositories
- Services con reglas
- Controllers
- Seguridad básica
- Pruebas manuales con Postman

Día 2 – Frontend + integración

- Vistas CRUD
- Formulario alquiler
- Validaciones UI
- Dashboard simple
- Build y deploy

✅ Conclusión

La arquitectura propuesta:

- Es simple pero robusta.
- Cumple todos los requisitos.
- Minimiza riesgo.
- Es implementable en 2 días.
- Permite crecimiento futuro.

Si deseas, el siguiente paso puede ser:

- ☑ Generar estructura de paquetes exacta
- ☑ Diseñar entidades JPA completas
- ☑ Diseñar SecurityConfig
- ☑ Diseñar lógica detallada de solapamiento
- ☑ Crear checklist técnico de implementación

Indícame cuál desarrollamos ahora.