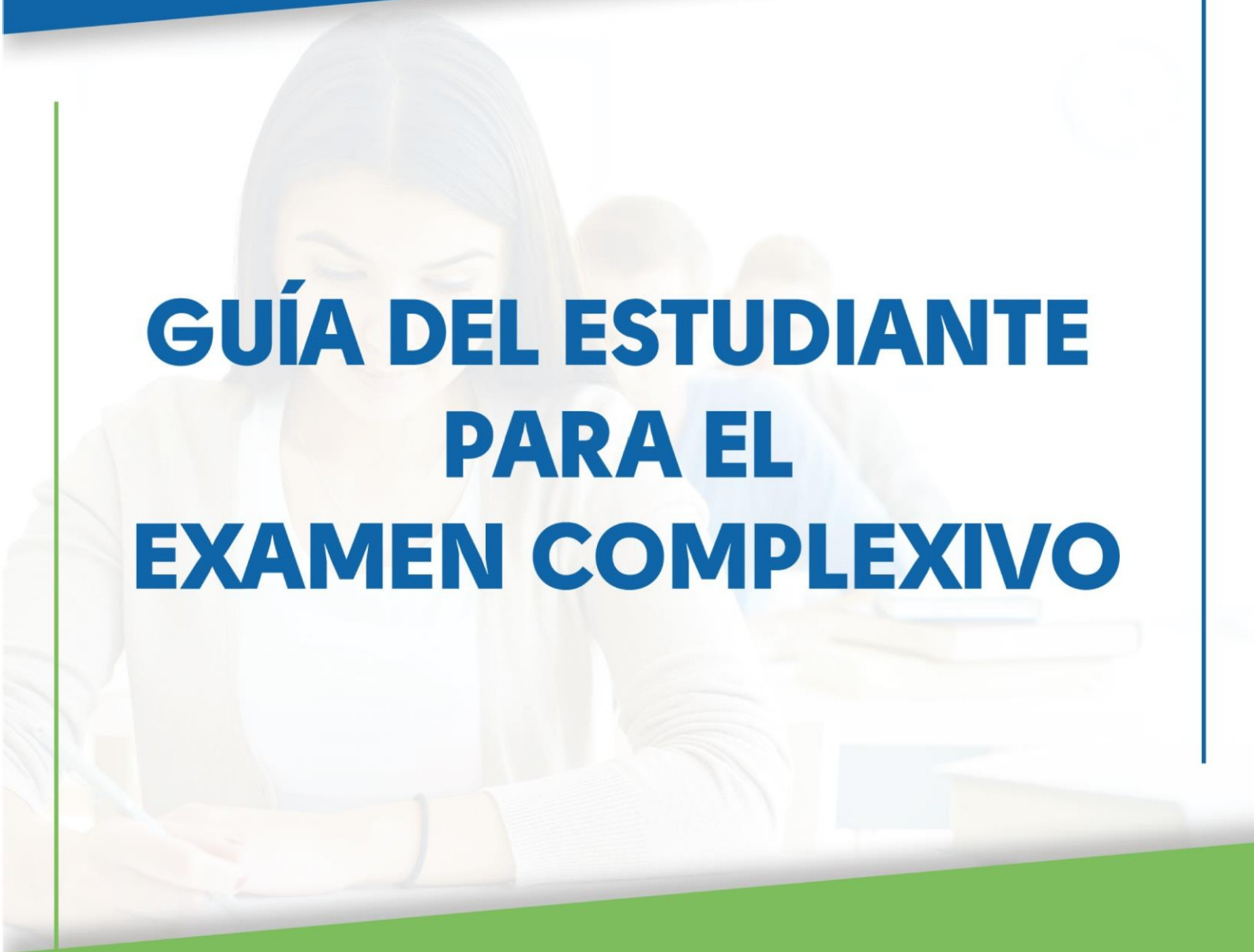
UNIDAD DE INTEGRACIÓN CURRICULAR

CARRERA: TECNOLOGÍA SUPERIOR EN DESARROLLO DE SOFTWARE



GUÍA DEL ESTUDIANTE PARA EL EXAMEN COMPLEXIVO

UTE ESCUELA DE TECNOLOGÍAS

PERÍODO ACADÉMICO

2025-II

EXAMEN PRÁCTICO : Sistema Básico de Gestión de Reservas y Operaciones para un Cine

1. t ítulo del caso:	**Sistema Básico de Gestión de Reservas y Operaciones para un Cine**
2. N2 Áreas de la carrera:	**1.** Tecnologías de la Información **2.** Diseño y Desarrollo de Software
3. N3 Subáreas de la carrera relacionadas con el perfil de egreso:	**4.** Análisis de Datos y Sistemas **5.** Base de Datos **6.** Programación **7.** Calidad y Gestión
8. N4 Materias:	1. Base de Datos I 2. Base de Datos II 3. Programación III 4. Programación IV 5. Sistemas Operativos
9. Identificación de resultados de aprendizaje relacionados con el caso:	El estudiante demuestra que es capaz de: • Diseñar y administrar bases de datos SQL y NoSQL. • Implementar un backend con API REST. • Consumir servicios desde aplicaciones web y móviles. • Aplicar principios básicos de seguridad y validación. • Ejecutar pruebas unitarias. • Configurar y ejecutar un sistema en Linux. • Integrartodas las capas del sistema de forma funcional.
10. Objetivo	Desarrollar, documentar y presentar una solución de software completa que permita gestionar servicios internos de una organización, integrandotodas las capas del stacktecnológico (front-end, back-end, bases de datos SQL y NoSQL, seguridad, pruebas y despliegue en Linux), mediante metodologías de gestión de proyectos y herramientas colaborativas.

11. Planteamiento:	Diseñe e implemente un Sistema Básico de Gestión de Alquileres y Operaciones para una Agencia de Alquiler de Vehículos, integrando una base de datos relacional, una base de datos no relacional, un backend con API, una interfaz web, una aplicación móvil y su ejecución en un entorno Linux. El sistema debe permitir registrar órdenes de producción, consultar su estado y almacenar eventos operativos, demostrando la integración completa del stack tecnológico y la aplicación de buenas prácticas de desarrollo.
12. Descripción de entregables	**Back-end funcional** • API REST u otra arquitectura definida • Validaciones, control de acceso, sanitización **Front-end funcional** • Interfaz implementada • Integración con API • Manejo de estados y errores **Proyecto Móvil funcional** • Interfaz implementada • Integración con API • Manejo de estados y errores **Base de datos relacional (SQL)** • Scripts de creación • Consultas necesarias **Base de datos NoSQL** • Colecciones, documentos • Consultas **Integración completa del sistema** • Front-end ↔ API • Proyecto Móvil↔ API • API ↔ SQL y NoSQL • Flujo end-to-end funcionando **Ejecución en Linux** • Comandos utilizados • Configuración necesaria • Evidencia de funcionamiento enterminal

CASO: Sistema Básico de Gestión de Reservas y Operaciones para un Cine

Regla de negocio: gestionar funciones de cine y reservas de asientos con estados (RESERVED, CONFIRMED, CANCELLED), y registrar operación NoSQL (cartelera/bitácora y eventos) vinculada a cada reserva. Misma estructura técnica: 2 tablas (PostgreSQL) + 2 colecciones (MongoDB), ReactJS consume tablas, React Native consume colecciones, integración por reservation_id (id SQL) en eventos NoSQL.

MODELO DE DATOS OBLIGATORIO (2 tablas + 2 colecciones)

Tablas PostgreSQL (2):

Tabla shows (funciones):

id BIGSERIAL PRIMARY KEY
movie_title VARCHAR(120) NOT NULL
room VARCHAR(20) NOT NULL
price NUMERIC(10,2) NOT NULL
available_seats INTEGER NOT NULL

Tabla reservations (reservas):

id BIGSERIAL PRIMARY KEY
show_id BIGINT NOT NULL REFERENCES shows(id)
customer_name VARCHAR(120) NOT NULL
seats INTEGER NOT NULL
status VARCHAR(20) NOT NULL (RESERVED, CONFIRMED, CANCELLED)
created_at TIMESTAMP NOT NULL DEFAULT NOW()

Colecciones MongoDB (2):

Colección movie_catalog (cartelera / catálogo):
_id ObjectId
movie_title string
genre string
duration_min int
rating string
is_active bool

Colección reservation_events (eventos operativos):

_id ObjectId
reservation_id long
event_type string (CREATED, CONFIRMED, CANCELLED, CHECKED_IN)
source string (WEB, MOBILE, SYSTEM)
note string
created_at date

Nota de integración: reservation_events.reservation_id debe guardar el id de la reserva creada en PostgreSQL.

SECUENCIA LÓGICA (PASO A PASO, APOYÁNDOSE EN MIGRACIONES DJANGO)

1. Crear BD y usuario en PostgreSQL
2. Crear proyecto Django y modelos
3. Migrar (Django crea tablas)
4. Verificar tablas en psql
5. Crear endpoints SQL (shows, reservations)
6. Crear DB y usuario en Mongo
7. Crear colecciones e índices
8. Crear endpoints Mongo (movie-catalog, reservation-events)
9. Integrar: al crear reservation en SQL generar evento en Mongo
10. ReactJS consume SQL (shows y reservations)
11. Mobile consume Mongo (movie_catalog y reservation_events)
12. Evidencia en Linux + Git.

MATERIAS Y PREGUNTAS (MISMA CANTIDAD QUE EL MODELO ORIGINAL)

BASE DE DATOS I – PostgreSQL (8 preguntas)

Objetivo: Crear y configurar una base de datos relacional en PostgreSQL para el backend, asegurando gestión de usuarios, permisos, integridad y optimización.

1. Crear la base de datos cinema_db en PostgreSQL desde terminal/psql.
2. Crear el usuario backend_user con contraseña segura y sin privilegios de superusuario.
3. Asignar permisos mínimos para migraciones Django (conectar, usar schema, crear/alterar objetos).
4. Verificar desde psql conexión con backend_user y existencia de cinema_db.
5. Ejecutar migraciones Django y verificar que existen las tablas shows y reservations (listar tablas y un select).
6. Crear un índice en reservations(status) y demostrar su uso con consultas (select * from reservations; select * from shows;).
7. Crear una vista vw_active_reservations que liste únicamente reservas en estado RESERVED o CONFIRMED.
8. Crear una función o trigger (uno): función que retorne total de reservas por estado o trigger que impida seats <= 0 o total <= 0 a nivel DB (además de validación en backend).

BASE DE DATOS II – MongoDB (7 preguntas)

Objetivo: Diseñar e implementar una base de datos NoSQL para eventos y registros del sistema.

1. Crear/definir la base de datos cinema_logs (evidenciar use cinema_logs en mongosh).
2. Crear el usuario mongo_backend_user con contraseña exa_2026_ute.
3. Asignar roles mínimos para leer/escribir en cinema_logs (sin permisos administrativos globales).
4. Probar autenticación conectándose con mongo_backend_user y verificar que puede operar sobre cinema_logs.
5. Verificar/crear colecciones movie_catalog y reservation_events con los campos definidos.
6. Crear índice en reservation_events(reservation_id) y evidenciar con getIndexes().
7. Ejecutar 2 consultas: eventos por reservation_id y eventos por rango de fechas (created_at).

PROGRAMACIÓN III – Backend Django + Frontend React (9 preguntas)

Backend – Django REST

1. Crear proyecto Django cinema_backend.
2. Crear app cinema.
3. Configurar conexión a PostgreSQL (cinema_db).
4. Configurar conexión a MongoDB (cinema_logs).
5. Crear modelos Django para shows y reservations (según campos).
6. Crear endpoint GET /shows que liste funciones (SQL).
7. Crear endpoints GET /reservations y POST /reservations (SQL), y en POST registrar evento en reservation_events (Mongo) con reservation_id.
   Frontend – ReactJS (consume TABLAS SQL)
8. Crear proyecto React.
9. Crear vistas que consuman GET /shows y GET /reservations y muestren funciones y reservas con estado, manejo de carga/error y actualización tras crear reserva.

PROGRAMACIÓN IV – React Native (9 preguntas)

Objetivo: Consumir la API desde una app móvil básica enfocada a NoSQL.

1. Crear proyecto React Native.
2. Configurar conexión con la API del backend.
3. Crear pantalla principal.
4. Consumir endpoint GET /movie-catalog (Mongo: colección movie_catalog) y mostrar cartelera.
5. Consumir endpoint GET /reservation-events (Mongo: colección reservation_events) y mostrar eventos (por reservation_id o últimos).
6. Mostrar cartelera y eventos en listas (puede ser navegación simple).
7. Manejar estado de carga.
8. Manejar errores de conexión.
9. Evidenciar ejecución en emulador o dispositivo.

SISTEMAS OPERATIVOS – Ubuntu 24.04 (8 preguntas)

Objetivo: Ejecutar el sistema en Linux.

Trabajar desde el HOME (~).

1. Crear estructura del proyecto
   Crear una carpeta llamada examen, dentro de ella una carpeta cine, y dentro de cine crear las carpetas: backend, frontend, movil, docs. Luego verificar la estructura.
   Comandos a usar:
   cd, mkdir -p, tree

2. Navegación y verificación
   Entrar a la carpeta cine, verificar la ubicación actual y listar contenido.
   Comandos a usar:
   cd, pwd, ls, ls -la

3. Crear archivos de evidencia
   Dentro de docs crear los archivos cine_comandos.txt y evidencia.txt, luego registrar la fecha actual dentro de evidencia.txt y verificar contenido.
   Comandos a usar:
   touch, date >>, cat

4. Redirección de salida
   Guardar la salida del comando who en un archivo y luego agregar la salida de ls -la al mismo archivo. Verificar contenido.
   Comandos a usar:
   who, ls -la, >, >>, cat

5. Buscar palabra dentro de archivo

Dentro de la carpeta docs, crear o sobrescribir el archivo cine_comandos.txt con el siguiente contenido (copiar y pegar exactamente):
cat << EOF > cine_comandos.txt
Sistema Cine - Backend
GET /api/movies/
GET /api/reservations/
POST /api/reservations/
DELETE /api/reservations/20/
INFO: reservation created successfully
INFO: reservations service running
WARN: reservations timeout detected
EOF

Verificar que el contenido se guardó correctamente.
Luego buscar la palabra **reservations** dentro del archivo y mostrar también el número de línea donde aparece.

Comandos a usar:
cat << EOF, cat, grep, grep -n

6. Localizar archivo

Crear dentro de la carpeta frontend un archivo llamado README.md y luego localizarlo desde la carpeta cine.
Comandos a usar:
touch, find, locate (opcional)

7. Copiar y mover archivos

Copiar el archivo evidencia.txt para crear un respaldo y luego mover ese respaldo a otra carpeta del proyecto.
Comandos a usar:
cp, mv, ls -la

8. Permisos y Sticky Bit

Crear una carpeta llamada shared_cine, aplicar permisos 1777 (Sticky Bit. y verificar que aparezca la letra t en los permisos.
Comandos a usar:
mkdir, chmod, ls -l