Universidad Católica Andrés Bello
Escuela de Ingeniería Informática
Desarrollo de APIs con Python

Proyecto Final

"El Buen Sabor" es un restaurante ficticio que busca modernizar su sistema de reservas, permitiendo a los comensales:

- Reservar mesas en horarios específicos.
- Pre-ordenar platos del menú al momento de reservar (¡sin preocuparse por precios, pues la cuenta será una "sorpresa"!).
- Autenticación segura con OAuth2 + JWT.

El objetivo del proyecto es desarrollar el backend de esta plataforma utilizando FastAPI, aplicando buenas prácticas como DDD, TDD y Arquitectura Hexagonal, junto con tecnologías como PostgreSQL, SQLModel, Docker y JWT.

Entregables del Proyecto

Repositorio GitHub

- Estructura DDD/Hexagonal (por módulo):

```
/src
  /auth
    /domain       # Entidades y lógica de negocio
    /infrastructure # Repositorios y seguridad
    /api          # Endpoints
  /restaurants    # Misma estructura por módulo
  /reservations
  /menu
  /notificactions
  /dashboard
```

- Código:
  - Commits descriptivos (ej: feat: add OAuth2 token endpoint).
  - Tests con ≥70% cobertura (pytest).
- README.md:
  - Instrucciones para ejecutar con Docker.
  - Explicación de la arquitectura (DDD/Hexagonal).

Funcionalidades Mínimas

Módulo	**Detalle**
Autenticación	OAuth2 (Password Flow) + JWT. Roles: Cliente, Admin.
Restaurantes	CRUD de restaurantes y mesas (solo Admin)
Reservas	Agendar/cancelar con validación de horarios y capacidad.
Menú	Pre-ordenar platos al reservar

Módulo	Detalle
Notificaciones	Solo con un print por cónsola
Dashboard de reservas	Solo los endpoints necesarios (GET /stats)

Informe Escrito:

- Archivo PDF indicando los integrantes del equipo, enlace al repositorio GitHub donde está el proyecto. Cualquier otra información que sea de utilidad para ejecutar el proyecto

Alcances

- Incluye:
  - Autenticación: OAuth2 con JWT y scopes.
  - DB: PostgreSQL con SQLModel + Alembic.
  - Arquitectura: DDD/Hexagonal (separación clara de capas).
  - Deployment: Docker Compose (FastAPI + Postgres).

- No Incluye:
  - Frontend o pasarela de pagos.
  - Notificaciones reales (solo print).

Limitaciones

- Máximo 1 reserva/cliente por horario.
- Sin precios o ingredientes en el menú.
- No hay recuperación de contraseña.

Rúbrica de Evaluación

Criterio	Puntos	Detalle
Funcionalidad Completa	40	Todos los módulos básicos funcionando. • Autenticatión: 8 pts • Restaurantes: 8 pts • Reservas: 12 pts • Menú: 6 pts • Notificaciones: 3 pts • Dashboard: 3 pts
Validaciones	15	Reglas de negocio: - Reservas: Solapamiento de horarios, límite de platos pre-ordenados (5). - Restaurantes: Horarios de apertura/cierre válidos (5). - Menú: Platos únicos por restaurante (5).
Arquitectura (DDD/Hexagonal)	10	Carpeta bien organizadas (5), inversión de dependencias (repositorios abstractos) (5).
TDD y Tests	20	Tests unitarios/integración: - Cobertura: ≥70% con pytest (10). - Pruebas significativas: - Validación de reglas de negocio (5). - Integración entre módulos (5).
Docker y DB	10	- Contenedores: FastAPI + Postgres funcionando con docker-compose (5). - Migraciones: Alembic aplicado correctamente (5).
Trabajo en Equipo	5	Commits significativos, PRs revisados, división equitativa.

Estructura de Carpetas sujerido (DDD/Hexagonal)

```
/src
│
├── /auth                 # Ejemplo de módulo
│   ├── /domain           # User (entidad), AuthService (lógica)
│   ├── /infrastructure   # UserRepository, JWT/OAuth2 utils
│   └── /api              # FastAPI routers (/login, /register)
│
├── /restaurants         # Misma estructura
├── /reservations
├── /menu
├── /notifications          # Módulo nuevo
│   └── services.py         # Función `notify()`
├── /dashboard              # Módulo nuevo
│   ├── /domain             # Lógica de métricas
│   └── /api                # Endpoints
│
└── /shared              # Database.py, exceptions.py
```

Convenciones de nombrado:

- **Idiomas**: Utilizar inglés para todos los nombres de carpetas, funciones, clases, métodos y variables.
- **Mayúsculas y minúsculas**: Utilizar snake_case para nombres de variables y funciones, y PascalCase para nombres de clases.
- **Significatividad**: Los nombres deben ser claros y descriptivos.

Reglas de Negocio

Módulo auth

Rol	**Permisos**
Cliente	- Crear/modificar su perfil. - Reservar mesas y pre-ordenar menú.
Admin	- Gestionar restaurantes, mesas y menú. - Ver todas las reservas.

Reglas Específicas
Registro de Usuarios:

- Solo se requiere: email (único), password, nombre.
- Default: Todos los nuevos usuarios son Clientes.
- Admin: Solo puede ser creado manualmente en la DB (vía migración o script).

Login (OAuth2 Password Flow):

- Validar que el email exista y la password coincida.
- Retornar access_token (JWT) con scopes según rol. Por ejemplo:

```
{
  "sub": "user@email.com",
  "scopes": ["client:read", "client:write"]
}
```

Protección de Endpoints:

- Usar OAuth2PasswordBearer en FastAPI
- Verificar scopes en endpoints sensibles

Seguridad Adicional:

- Passwords hasheadas con bcrypt.
- Tokens JWT con expiración corta (ej: 15 mins) + refresh_token opcional.

Errores Comunes a Validar

- 401 Unauthorized: Token inválido o expirado.
- 403 Forbidden: Cliente intenta acceder a endpoint de admin.
- 409 Conflict: Email ya registrado.

Pruebas (TDD)

- Test de Registro:
  - Verificar que un Cliente no pueda asignarse rol admin.
- Test de Login:
  - Validar que retorne 403 si las credenciales son incorrectas.

Módulo de Resaturantes

Rol	**Permisos**
Cliente	- Solo puede listar restaurantes y ver mesas disponibles.
Admin	- Crear/editar/eliminar restaurantes y mesas. - Listar todos los restaurantes.

Reglas Específicas

1. Gestión de Restaurantes
   - Creación:
     - Campos obligatorios: nombre, ubicación, hora_apertura, hora_cierre.
     - Validar que hora_cierre > hora_apertura.
     - El admin es el único que puede crear restaurantes.
   - Edición:
     - Solo el admin puede modificar datos del restaurante.
     - No se puede cambiar el id del restaurante.
   - Eliminación:
     - Si un restaurante tiene mesas asociadas, no se puede eliminar (primero eliminar mesas).

2. Gestión de Mesas
   - Creación:
     - Campos obligatorios: capacidad (número de personas), ubicacion (ej: "terraza", "interior").
     - capacidad debe ser ≥ 2 y ≤ 12 personas.
     - Un restaurante puede tener múltiples mesas, pero no puede haber dos mesas con el mismo numero_mesa en el mismo restaurante.
   - Disponibilidad:
     - Las mesas deben poder filtrarse por capacidad y ubicacion.

Validaciones Clave

- Horario del Restaurante:
  - Un restaurante no puede tener hora_cierre ≤ hora_apertura.
- Capacidad de Mesas:
  - Las mesas deben tener capacidad entre 2 y 12 personas.
- Número de Mesa Único:
  - No puede haber dos mesas con el mismo numero_mesa en un restaurante.
- Restricción de Eliminación:
  - Si un restaurante tiene mesas, no se puede eliminar (integrar con módulo de reservas).

Pruebas (TDD)

- Test de Creación de Restaurante:
  - Verificar que un Cliente no pueda crear restaurantes (403 Forbidden).
  - Validar que hora_cierre > hora_apertura (400 Bad Request).
- Test de Mesas:
  - Asegurar que una mesa con capacidad 1 o 13 sea rechazada.

Posibles Errores
- 403 Forbidden: Cliente intenta crear/editar restaurantes.
- 400 Bad Request: Datos inválidos (ej: capacidad de mesa incorrecta).
- 409 Conflict: Número de mesa duplicado.

Módulo Reservas

Rol	**Permisos**
Cliente	- Crear/cancelar sus reservas. - Pre-ordenar menú. - Ver sus reservas activas.
Admin	- Ver todas las reservas. - Cancelar cualquier reserva. - Filtrar por fecha/restaurante.

Reglas Específicas

1. Creación de Reservas
- Validaciones:
  - El cliente no puede tener más de 1 reserva activa en el mismo horario.
  - La mesa debe estar disponible en el horario solicitado.
  - El horario de reserva debe estar dentro del horario de apertura/cierre del restaurante.
  - Máximo 4 horas por reserva.
- Pre-orden:
  - Opcional: El cliente puede asociar platos del menú a la reserva.
  - Los platos deben existir en el menú del restaurante.

2. Cancelación
- Cliente: Solo puede cancelar sus propias reservas con al menos 1 hora de anticipación.
- Admin: Puede cancelar cualquier reserva en cualquier momento.

3. Estados de una Reserva:

```
PENDIENTE = "pendiente"   # Recién creada (default)
CONFIRMADA = "confirmada" # Opcional: si el restaurante confirma
CANCELADA = "cancelada"
COMPLETADA = "completada" # Después de la fecha/hora
```

Validaciones Clave
- Disponibilidad de Mesa:
  - No puede haber dos reservas activas para la misma mesa en el mismo horario.
- Límite de Reservas por Cliente:

    - Un cliente no puede tener dos reservas en el mismo horario, incluso en diferentes restaurantes.
- Pre-orden:
    - Los platos deben pertenecer al menú del restaurante de la mesa reservada.
- Cancelación:
    - Un cliente no puede cancelar una reserva que ya pasó.

Pruebas (TDD)

1. Test de Solapamiento:
    - Intentar crear dos reservas para la misma mesa en el mismo horario → 409 Conflict.
2. Test de Pre-orden Inválido:
    - Asociar un plato que no existe → 400 Bad Request.

Posibles Errores

- 400 Bad Request: Datos inválidos (ej: horario fuera de rango).
- 403 Forbidden: Cliente intenta cancelar reserva de otro.
- 409 Conflict: Mesa ya reservada en ese horario.

Módulo Menú

Rol	**Permisos**
Cliente	- Ver menú de un restaurante. - Pre-ordenar platos al reservar.
Admin	- Crear/editar/eliminar platos. - Gestionar categorías del menú.

Reglas Específicas

1. Gestión de Platos
    - Creación:
        - Campos obligatorios: nombre, descripcion, categoria (ej: "Entrada", "Principal").
        - nombre debe ser único dentro del mismo restaurante.
        - Opcional: imagen_url (simulada como string).
    - Edición:
        - Solo el admin puede modificar platos.
        - No se puede cambiar el restaurante_id de un plato.
    - Eliminación:
        - Si un plato está asociado a una reserva futura, no se puede eliminar (solo marcar como no disponible).

2. Pre-orden en Reservas
    - Validaciones:
        - Los platos pre-ordenados deben existir en el menú del restaurante de la mesa reservada.
        - Máximo 5 platos por reserva (para simplificar).

3. Categorías del Menú
    - Lista fija: ["Entrada", "Principal", "Postre", "Bebida"].

Validaciones Clave

1. Unicidad de Platos:
   - No puede haber dos platos con el mismo nombre en un restaurante.
2. Pre-orden Válido:
   - Los platos pre-ordenados deben estar disponible=True y pertenecer al restaurante de la mesa.
3. Eliminación Segura:
   - Si un plato está en reservas futuras, se marca como disponible=False en lugar de eliminarlo.

Pruebas (TDD)

1. Test de Creación de Plato:
   - Verificar que un Cliente no pueda crear platos (403 Forbidden).
   - Validar que nombre sea único (409 Conflict).
2. Test de Pre-orden:
   - Intentar pre-ordenar un plato de otro restaurante → 400 Bad Request.

Módulo de Notificaciones y Dashboard

Notificaciones : Reglas:

- Simular envío de notificaciones vía print en consola para:
  - Reserva creada: "Notificación: Reserva confirmada para {fecha} en {restaurante}."
  - Reserva cancelada: "Notificación: Reserva cancelada (ID: {id_reserva})."
  - Pre-orden registrado: "Notificación: Pre-orden con {n_platos} platos."

Dashboard: endpoints

Endpoint	Descripción	Responsable
GET /dashboard/reservas	Total de reservas por día/semana.	Admin
GET /dashboard/platos	Platos más pre-ordenados (top 5).	Admin
GET /dashboard/ocupacion	% de ocupación por restaurante (mesas usadas/totales).	Admin

Métricas Clave:

- Reservas por día: Agrupadas por fecha_hora.
- Top platos: Contar apariciones en reservas.platos_preordenados.
- Ocupación: (mesas_reservadas / mesas_totales) * 100.

Condiciones:

- El proyecto debe ser realizado en equipo. Cada equipo debe tener un máximo de cuatro integrantes y un mínimo de dos integrantes. **No se permitirá equipos individuales,** ya que se está evaluando la competencia de trabajo en equipo.
- Los equipos deben estar conformados a más tardar el día **03 de junio de 2025** , ingresando el archivo en la tarea definida para este caso. Esta notidficación tiene un valor de 5% de la nota final. Ver las condiciones en la tarea publicada en M7

- El informe escrito del proyecto debe ser entregado a más tardar el martes **7 de julio de 2025, a las 11:59 pm**, subiendo un archivo pdf al sitio indicado del proyecto. Este informe tiene un valor de 15% del valor de la nota final. Ver las condiciones en la tarea correspondiente en M7.
- La presentación se realizará el miércoles **8 de julio de 2025 en las horas de clase.** La presentación tiene un peso de 80% de la nota y se evaluará según la rúbrica presentada en el presente documento.