CARATURA: PLATAFORMA DE RESERVA DE HOSPEDAJE

Equipo LAST PUSH
Juan Felipe Gómez (98265)
Susana Cai (106991)
Luciano Desimone (114055)
Alexis Herrera (111127)
Juan Francisco Solís (112796)
Stefano Diaz (113909)

2 de diciembre de 2025

Resumen

Resumen: Este documento presenta el informe técnico del Trabajo Práctico Final del curso Ingeniería de Software (IDS). El proyecto "Hotel IDS" es una plataforma web integral para la reserva de hospedaje que integra un frontend responsivo con HTML5, CSS3 y JavaScript, y un backend robusto basado en API REST con Flask y MySQL. Se describen la arquitectura cliente-servidor, componentes principales, rutas de la API, modelo de datos, y los desafíos enfrentados durante el desarrollo. El sistema permite a los usuarios registrarse, autenticarse, visualizar habitaciones disponibles, realizar reservas, y mantener un historial de sus transacciones.

Índice

1. Integrantes del Equipo

El equipo LAST PUSH está compuesto por seis integrantes de la carrera de Ingeniería en Sistemas:

1. **Juan Felipe Gómez** (98265) - Rol: Desarrollo Frontend
2. **Susana Cai** (106991) - Rol: Desarrollo Backend
3. **Luciano Desimone** (114055) - Rol: Base de Datos y Arquitectura
4. **Alexis Herrera** (111127) - Rol: Integración y Testing
5. **Juan Francisco Solís** (112796) - Rol: Documentación y Deployment
6. **Stefano Diaz** (113909) - Rol: Seguridad y Optimización

Cada integrante aportó sus competencias específicas para lograr una solución integral y profesional.

2. Introducción

El presente informe detalla la arquitectura, desarrollo e implementación de una plataforma integral de reserva de hospedaje. Este proyecto representa una solución completa que integra tecnologías modernas de *frontend* y *backend* para proporcionar a los usuarios una experiencia fluida y eficiente en la búsqueda y reserva de alojamientos.

El desarrollo de la plataforma "Hotel IDS" requirió la aplicación de conocimientos en ingeniería de software, gestión de bases de datos, programación web, y trabajo colaborativo en equipo. Durante el ciclo de desarrollo, el equipo LAST PUSH enfrentó diversos desafíos técnicos que permitieron fortalecer las competencias de cada integrante y consolidar una solución robusta y escalable.

2.1. Descripción General del Proyecto

La plataforma "Hotel IDS" es una aplicación web diseñada para facilitar la reserva de alojamientos (hoteles, cabañas, departamentos) con los siguientes objetivos:

- Proporcionar una interfaz intuitiva para navegar disponibilidad de habitaciones
- Permitir reservas fáciles y seguras
- Mostrar información detallada con fotografías de alojamientos
- Facilitar contacto entre usuarios y la plataforma
- Mantener registro de historial de reservas
- Integrar herramientas interactivas como mapas

2.2. Objetivos del Proyecto

1. Crear una plataforma web moderna y responsiva
2. Implementar un sistema robusto de gestión de reservas
3. Desarrollar un backend seguro con autenticación de usuarios
4. Proporcionar una base de datos eficiente para almacenar información
5. Garantizar la escalabilidad y mantenibilidad del código

3. Tecnologías Utilizadas

3.1. Frontend

Tecnología	Descripción
Python/Flask	Framework web para renderizar templates HTML
HTML5	Estructura y semántica de las páginas
CSS3	Estilos, diseño responsivo y animaciones
Bootstrap	Framework CSS para componentes prediseñados
JavaScript	Interactividad del lado del cliente
Leaflet.js	Integración de mapas interactivos

3.2. Backend

Tecnología	Descripción
Python	Lenguaje de programación principal
Flask	Microframework web para APIs REST
Flask-CORS	Gestión de CORS entre frontend y backend
Flask-Mail	Sistema de notificaciones por correo
SQLAlchemy	ORM para interacción con base de datos
MySQL	Sistema de gestión de base de datos relacional

3.3. Herramientas de Desarrollo

- **Control de Versiones:** Git y GitHub
- **Documentación API:** OpenAPI/Swagger
- **Entorno Virtual:** Python venv
- **Gestor de Dependencias:** pip

4. Solución Propuesta

4.1. Visión General

La solución propuesta es una plataforma web completa de reserva de hospedaje que integra dos capas fundamentales:

1. **Capa de Presentación (Frontend):** Interfaz responsiva y amigable desarrollada con Flask, HTML5, CSS3 y JavaScript que permite a los usuarios interactuar con el sistema de forma intuitiva.
2. **Capa de Negocio y Datos (Backend):** API REST robusta desarrollada con Flask que proporciona servicios para gestionar usuarios, habitaciones y reservas, conectada a una base de datos MySQL.

4.2. Componentes Principales de la Solución

1. Sistema de Gestión de Usuarios

- Registro de nuevos usuarios con validación de datos
- Autenticación segura mediante sesiones Flask
- Gestión de perfiles personales
- Historial de reservas
- Recuperación de contraseña mediante correo electrónico

2. Catálogo de Habitaciones

- Visualización de todas las habitaciones disponibles
- Búsqueda y filtrado por tipo, capacidad y precio
- Detalles completos con fotografías y servicios
- Consulta de disponibilidad por fechas

3. Sistema de Reservas

- Creación de reservas con validaciones complejas
- Cálculo automático de precio total
- Verificación de disponibilidad de habitaciones
- Confirmación por correo electrónico
- Gestión y cancelación de reservas
- Historial completo de transacciones

4. Seguridad

- Hash seguro de contraseñas
- Validación de datos en servidor
- Protección contra inyecciones SQL mediante ORM
- Control de acceso basado en sesiones
- Configuración de CORS para comunicación segura

4.3. Flujo de Datos

El sistema opera mediante un flujo cliente-servidor donde:

1. El usuario interactúa con la interfaz frontend (puerto 5000)
2. El frontend realiza peticiones HTTP/JSON al backend (puerto 5010)
3. El backend procesa las solicitudes, valida datos y accede a la base de datos MySQL
4. El backend retorna respuestas JSON al frontend
5. El frontend actualiza la interfaz con los datos recibidos

5. Arquitectura del Sistema

5.1. Estructura General

La arquitectura del proyecto sigue un patrón de separación cliente-servidor:

- **Frontend:** Aplicación Flask que renderiza vistas HTML
- **Backend:** API REST desarrollada con Flask para gestionar datos
- **Base de Datos:** MySQL para almacenamiento persistente

5.2. Estructura de Carpetas

```
IDS-TPFINAL/
          backend/
                app.py               # Punto de entrada del
   backend
                db.py                # Configuraci n de base de
   datos
                requirements.txt      # Dependencias Python
                routes/
                      habitaciones.py  # API de habitaciones
                      reservas.py      # API de reservas
                      usuarios.py      # API de usuarios
                openapi.yaml          # Documentaci n de API
                init_db.py           # Scripts de inicializaci n
          frontend/
                app.py               # Aplicaci n Flask frontend
                requirements.txt
                static/
                      css/             # Estilos CSS
                      js/              # Scripts JavaScript
                      img/             # Im genes
                template/
                    base.html         # Template base
                    index.html        # P gina principal
                    rooms.html        # Listado de habitaciones
```

```
                    reservar.html     # P gina de reserva
                    login.html        # Login
                    register.html     # Registro
                    user.html         # Perfil de usuario
        README.md
```

5.3. Diagrama de flujo (arquitectura)

A continuación se describe la conexión entre el frontend (Flask) y el backend (API REST) y la base de datos.

Componentes principales:

- **FRONTEND (Flask):** Puerto 5000
  - app.py: Rutas, sesiones, comunicación
  - Templates HTML: index, login, rooms, etc
- **BACKEND (Flask):** Puerto 5010 - API REST
  - Usuarios BP: /usuarios, /login
  - Habitaciones BP: /habitaciones
  - Reservas BP: /reservas
- **MySQL Database:** usuarios, habitaciones, reservas

Flujo de datos: El frontend comunica con el backend a través de peticiones HTTP/JSON, y el backend se conecta con la base de datos MySQL para persistir datos.

6. Flujos de Trabajo Principal

Esta sección describe los flujos principales de interacción entre el frontend y backend, mostrando cómo se comunican los componentes para completar operaciones clave.

6.1. Flujo 1: Registro de Usuario

1. **Usuario accede a /register**: El frontend renderiza el formulario de registro
2. **Usuario completa datos**: Nombre, email, contraseña
3. **Frontend valida**: Verifica formato de email y coincidencia de contraseñas
4. **POST a /usuarios/**: El frontend envía los datos en JSON al backend
5. **Backend valida**: Verifica que el email no esté duplicado
6. **Backend almacena**: Inserta el nuevo usuario en la BD
7. **Respuesta 201**: Retorna mensaje de éxito
8. **Redirección**: Frontend redirige a página de login

Endpoint utilizado: `POST /usuarios/`
Request JSON:

```
{
  "name": "Juan P rez",
  "email": "juan@example.com",
  "password": "micontrase a123"
}
```

Response (201):

```
{
  "mensaje": "Usuario creado exitosamente"
}
```

6.2. Flujo 2: Autenticación (Login)

1. **Usuario accede a /login**: Formulario de autenticación
2. **Usuario ingresa credenciales**: Email y contraseña
3. **POST a /usuarios/login**: Frontend envía credenciales
4. **Backend busca usuario**: Query en BD por email
5. **Backend valida contraseña**: Compara contraseña ingresada
6. **Respuesta 200**: Si es válida, retorna datos del usuario
7. **Sesión creada**: Frontend almacena user_id, user_name en sesión Flask
8. **Redirección**: Usuario redirigido a /user/¡user$_id$ >

Endpoint utilizado: `POST /usuarios/login`
Request JSON:

```
{
  "email": "juan@example.com",
  "password": "micontrase a123"
}
```

Response (200):

```
{
  "id": 1,
  "nombre": "Juan P rez",
  "email": "juan@example.com"
}
```

6.3. Flujo 3: Visualizar Habitaciones

1. **Usuario accede a /rooms**: Página de catálogo
2. **GET a /habitaciones/**: Frontend solicita lista
3. **Backend obtiene datos**: SELECT * FROM habitaciones
4. **Response 200**: Retorna JSON con todas las habitaciones
5. **Frontend renderiza**: Muestra catálogo con imágenes y precios
6. **Usuario puede filtrar**: Por tipo, precio, capacidad

Endpoint utilizado: `GET /habitaciones/`
Response (200):

```
[
  {
    "id": 1,
    "nombre": "Habitaci n Doble",
    "tipo": "doble",
    "capacidad": 2,
    "precio_por_dia": 150.00,
    "descripcion": "Habitaci n amplia con vista al mar",
    "imagen": "habitacion_1.jpg",
    "servicios": "WiFi, Aire acondicionado, TV"
  },
  ...
]
```

6.4. Flujo 4: Crear Reserva (Flujo Completo)

Este es el flujo más complejo. Requiere usuario logueado.

1. **Usuario accede a /reservar**: Solo si está logueado
2. **GET /reservar**: Frontend obtiene sesión (user_id, user_name)
3. **Usuario completa formulario**: Selecciona:
   - Habitación (id)
   - Fecha de entrada
   - Fecha de salida
   - Cantidad de adultos y niños
   - Datos de contacto (nombre, email, teléfono)
   - Método de pago
   - Número de tarjeta (últimos 4 dígitos)
4. **Frontend valida**: Fechas coherentes, datos completos

5. **POST a /reservas/**: Envía payload JSON con todos los datos
6. **Backend valida**:
   - Campos requeridos presentes
   - Fechas en formato correcto
   - Fecha de salida ¿fecha de entrada
   - Habitación existe
   - Cantidad de personas válida
7. **Backend calcula precio**:
   - Obtiene precio_por_dia de la habitación
   - Calcula días: (fecha_salida - fecha_entrada)
   - precio_total = precio_por_dia días
8. **Backend busca usuario**: Por email (id_usuario)
9. **Backend inserta reserva**: INSERT en tabla reservas
10. **Backend envía email**: Notificación de reserva a admin
11. **Response 201**: Retorna ID de reserva y confirmación
12. **Frontend muestra confirmación**: Número de reserva y total

Endpoint utilizado: `POST /reservas/`
Request JSON:

```
{
  "id_habitacion": 1,
  "fecha_entrada": "2025-12-01",
  "fecha_salida": "2025-12-05",
  "adultos": 2,
  "ninos": 1,
  "nombre_completo": "Juan P rez Garc a",
  "email": "juan@example.com",
  "telefono": "598 99123456",
  "metodo_pago": "tarjeta",
  "tarjeta_ultimos4": "1234"
}
```

Response (201):

```
{
  "id": 42,
  "precio_total": 600.00,
  "estado": "pendiente",
  "mensaje": "Reserva creada correctamente"
}
```

6.5. Flujo 5: Ver Historial de Reservas

1. **Usuario logueado accede a /user/¡id¿**: Panel de usuario
2. **GET /usuarios/¡id¿**: Obtiene datos personales
3. **GET /reservas/usuario/¡id¿**: Obtiene historial de reservas
4. **Backend hace JOIN**: Conecta reservas con habitaciones y usuario
5. **Response 200**: Retorna lista de reservas con detalles
6. **Frontend renderiza**: Tabla con historial completo

Endpoints utilizados:

- `GET /usuarios/{id}`
- `GET /reservas/usuario/{id}`

7. Componentes Principales

7.1. Backend - API REST

7.1.1. Arquitectura de Blueprints

El backend utiliza **Flask Blueprints** para organizar la API REST en módulos independientes. Los blueprints son como "submódulos" de Flask que agrupan rutas relacionadas.

Estructura:

```
# backend/app.py
from backend.routes.habitaciones import habitaciones_bp
from backend.routes.reservas import reservas_bp
from backend.routes.usuarios import usuarios_bp

# Registrar blueprints con prefijos URL
app.register_blueprint(habitaciones_bp, url_prefix="/habitaciones
    ")
app.register_blueprint(reservas_bp, url_prefix="/reservas")
app.register_blueprint(usuarios_bp, url_prefix="/usuarios")
```

Esto significa:

- Todas las rutas en `habitaciones.py` se prefijan con `/habitaciones`
- Todas las rutas en `reservas.py` se prefijan con `/reservas`
- Todas las rutas en `usuarios.py` se prefijan con `/usuarios`

7.1.2. Blueprint 1: Usuarios

Archivo: `backend/routes/usuarios.py`
Responsabilidades:

- Registro de nuevos usuarios
- Autenticación (login)
- Obtención de datos de usuario
- Listado de usuarios

Rutas implementadas:

```
usuarios_bp = Blueprint("usuarios", __name__)

@usuarios_bp.route('/', methods=['GET'])
def get_usuarios():  # GET /usuarios/
    # Retorna lista de todos los usuarios

@usuarios_bp.route('/<int:id_usuario>', methods=['GET'])
def get_usuario(id_usuario):  # GET /usuarios/1
    # Retorna datos de usuario espec fico

@usuarios_bp.route('/', methods=['POST'])
def crear_usuario():  # POST /usuarios/
    # Crea nuevo usuario (registro)

@usuarios_bp.route('/login', methods=['POST'])
def login_usuario():  # POST /usuarios/login
    # Autentica usuario y retorna datos
```

7.1.3. Blueprint 2: Habitaciones

Archivo: `backend/routes/habitaciones.py`
Responsabilidades:

- Listado de habitaciones disponibles
- Obtención de detalles de habitación específica
- Información de precios y servicios

Rutas implementadas:

```
habitaciones_bp = Blueprint("habitaciones", __name__)

@habitaciones_bp.route("/")
def get_habitaciones():  # GET /habitaciones/
    # Retorna lista de todas las habitaciones

@habitaciones_bp.route("/<int:habitacion_id>", methods=["GET"])
def get_habitacion(habitacion_id):  # GET /habitaciones/1
    # Retorna detalles de habitaci n espec fica
```

7.1.4. Blueprint 3: Reservas

Archivo: `backend/routes/reservas.py`
Responsabilidades:

- Creación de nuevas reservas
- Validación de disponibilidad
- Cálculo de precios
- Envío de emails de confirmación
- Obtención de reservas por usuario
- Listado de todas las reservas

Rutas implementadas:

```
reservas_bp = Blueprint("reservas", __name__)

@reservas_bp.route('/', methods=['GET'])
def listar_reservas():  # GET /reservas/
    # Retorna todas las reservas con detalles de usuario y
       habitaci n

@reservas_bp.route('/', methods=['POST'])
def crear_reserva():  # POST /reservas/
    # Crea nueva reserva con validaciones complejas
    # Calcula precio, verifica disponibilidad, env a email

@reservas_bp.route('/usuario/<int:usuario_id>', methods=['GET'])
def obtener_reservas_por_usuario(usuario_id):  # GET /reservas/
   usuario/1
    # Retorna reservas espec ficas de un usuario
```

7.2. Backend - API REST

7.2.1. Autenticación y Usuarios

La ruta `/usuarios` implementa:

- Registro de nuevos usuarios
- Login seguro con sesiones
- Gestión de perfiles de usuario
- Recuperación de contraseña

```
# Ejemplo de estructura de usuario
{
    "id": 1,
    "nombre": "Juan P rez",
    "email": "juan@example.com",
    "telefono": "+598 2 1234 5678",
    "historial_reservas": [...]
}
```

7.2.2. Gestión de Habitaciones

La ruta `/habitaciones` implementa:

- Listado de habitaciones disponibles
- Búsqueda y filtrado por características
- Consulta de disponibilidad por fechas
- Detalles completos con fotografías
- Información de precios y servicios

7.2.3. Sistema de Reservas

La ruta `/reservas` implementa:

- Creación de nuevas reservas
- Validación de disponibilidad
- Confirmación y generación de comprobantes
- Cancelación de reservas
- Historial de reservas por usuario

7.3. Frontend - Interfaz de Usuario

7.3.1. Páginas Principales

index.html Página principal con descripción del servicio y destacados

rooms.html Catálogo de habitaciones con filtros y búsqueda

reservar.html Formulario de reserva con selección de fechas

login.html Formulario de inicio de sesión

register.html Formulario de registro de nuevos usuarios

user.html Perfil de usuario y historial de reservas

7.3.2. Características Tecnológicas

- Diseño responsivo compatible con dispositivos móviles
- Interfaz intuitiva y accesible
- Animaciones CSS suaves
- Validación de formularios en cliente y servidor
- Integración de mapa interactivo con Leaflet.js

8. Base de Datos

8.1. Modelo de Datos

8.1.1. Tabla: Usuarios

Campo	Tipo	Descripción
id	INTEGER	Identificador único (PK)
nombre	VARCHAR	Nombre completo
email	VARCHAR	Correo electrónico (UNIQUE)
contraseña	VARCHAR	Contraseña hasheada
telefono	VARCHAR	Número de teléfono
$\text{fecha}_r egistro$	DATETIME	Fecha de registro

8.1.2. Tabla: Habitaciones

Campo	Tipo	Descripción
id	INTEGER	Identificador único (PK)
numero	VARCHAR	Número de habitación
tipo	VARCHAR	Tipo (individual, doble, suite)
capacidad	INTEGER	Número de huéspedes
precio	DECIMAL	Precio por noche
descripcion	TEXT	Descripción detallada
imagen	VARCHAR	URL de imagen
disponible	BOOLEAN	Estado de disponibilidad
servicios	TEXT	Servicios incluidos

8.1.3. Tabla: Reservas

Campo	Tipo	Descripción
id	INTEGER	Identificador único (PK)
usuario_id	INTEGER	Referencia a usuario (FK)
habitacion_id	INTEGER	Referencia a habitación (FK)
fecha$_{i}nicio$	DATE	Fecha de entrada
fecha$_{f}in$	DATE	Fecha de salida
estado	VARCHAR	Estado (pendiente, confirmada, cancelada)
precio$_{t}otal$	DECIMAL	Precio total de la reserva
fecha$_{r}eserva$	DATETIME	Fecha de creación

9. Funcionalidades Implementadas

9.1. Módulo de Usuarios

1. **Registro:** Formulario con validación de datos
2. **Login:** Autenticación segura con sesiones
3. **Perfil:** Visualización y edición de información personal
4. **Historial:** Vista de reservas anteriores
5. **Recuperación de Contraseña:** Enlace de recuperación por correo

9.2. Módulo de Habitaciones

1. **Catálogo:** Visualización de todas las habitaciones
2. **Búsqueda:** Filtrado por tipo, capacidad y precio
3. **Detalles:** Información completa con fotos y servicios
4. **Disponibilidad:** Verificación de fechas disponibles
5. **Calificaciones:** Sistema de puntuación de usuarios

9.3. Módulo de Reservas

1. **Creación:** Selección de fechas y habitación
2. **Validación:** Verificación de disponibilidad en tiempo real
3. **Confirmación:** Envío de correo de confirmación
4. **Gestión:** Modificación y cancelación de reservas
5. **Comprobante:** Generación de PDF con detalles

9.4. Características Adicionales

- **Mapa Interactivo:** Ubicación del hospedaje
- **Sistema de Contacto:** Formulario de consultas
- **Panel de Opiniones:** Comentarios de clientes
- **Notificaciones:** Recordatorios por correo
- **Soporte 24/7:** Chat o email de contacto

10. API REST - Detalle de Endpoints

10.1. Autenticación y Gestión de Usuarios

10.1.1. GET /usuarios/

Descripción: Obtiene lista de todos los usuarios registrados.
Request:

```
GET http://localhost:5010/usuarios/ HTTP/1.1
```

Response (200 OK):

```
[
  {
    "id": 1,
    "nombre": "Juan P rez",
    "email": "juan@example.com"
  },
  {
    "id": 2,
    "nombre": "Mar a Garc a",
    "email": "maria@example.com"
  }
]
```

10.1.2. POST /usuarios/

Descripción: Registra un nuevo usuario en el sistema.
Request:

```
POST http://localhost:5010/usuarios/ HTTP/1.1
Content-Type: application/json

{
  "name": "Carlos L pez",
  "email": "carlos@example.com",
  "password": "micontrase a123"
}
```

Response (201 Created):

```
{
  "mensaje": "Usuario creado exitosamente"
}
```

Response (409 Conflict):

```
{
  "error": "El usuario ya existe"
}
```

10.1.3. GET /usuarios/{id}

Descripción: Obtiene datos de un usuario específico.
Request:

```
GET http://localhost:5010/usuarios/1 HTTP/1.1
```

Response (200 OK):

```
{
  "id": 1,
  "nombre": "Juan P rez",
  "email": "juan@example.com"
}
```

Response (404 Not Found):

```
{
  "error": "usuario no encontrado"
}
```

10.1.4. POST /usuarios/login

Descripción: Autentica un usuario con email y contraseña.
Request:

```
POST http://localhost:5010/usuarios/login HTTP/1.1
Content-Type: application/json

{
  "email": "juan@example.com",
  "password": "micontrase a123"
}
```

Response (200 OK):

```
{
  "id": 1,
  "nombre": "Juan P rez",
  "email": "juan@example.com"
}
```

Response (404 Not Found):

```
{
  "error": "Usuario no existe"
}
```

Response (401 Unauthorized):

```
{
  "error": "Contrase a incorrecta"
}
```

10.2. Gestión de Habitaciones

10.2.1. GET /habitaciones/

Descripción: Lista todas las habitaciones disponibles del hotel.
Request:

```
GET http://localhost:5010/habitaciones/ HTTP/1.1
```

Response (200 OK):

```
[
  {
    "id": 1,
    "nombre": "Habitaci n Doble",
    "tipo": "doble",
    "capacidad": 2,
    "precio_por_dia": 150.00,
    "descripcion": "Habitaci n amplia con vista al mar",
    "imagen": "room_1.jpg",
    "disponible": true,
    "servicios": "WiFi, Aire acondicionado, TV, Minibar"
  },
  {
    "id": 2,
    "nombre": "Suite Premium",
    "tipo": "suite",
    "capacidad": 4,
    "precio_por_dia": 300.00,
    "descripcion": "Suite de lujo con jacuzzi privado",
    "imagen": "suite_1.jpg",
    "disponible": true,
    "servicios": "WiFi, AC, TV Smart, Jacuzzi, Sala de estar"
  }
]
```

10.2.2. GET /habitaciones/{id}

Descripción: Obtiene detalles completos de una habitación específica.
Request:

```
GET http://localhost:5010/habitaciones/1 HTTP/1.1
```

Response (200 OK):


```
{
  "id": 1,
  "nombre": "Habitaci n Doble",
  "tipo": "doble",
  "capacidad": 2,
  "precio_por_dia": 150.00,
  "descripcion": "Habitaci n amplia con vista al mar",
  "imagen": "room_1.jpg",
  "disponible": true,
  "servicios": "WiFi, Aire acondicionado, TV, Minibar"
}
```


Response (404 Not Found):


```
{
  "Error": "Habitacion no encontrada"
}
```


10.3. Gestión de Reservas

10.3.1. GET /reservas/

Descripción: Lista todas las reservas del hotel con detalles completos.
Request:

```
GET http://localhost:5010/reservas/ HTTP/1.1
```

Response (200 OK):


```
[
  {
    "id": 42,
    "id_habitacion": 1,
    "id_usuario": 1,
    "fecha_entrada": "2025-12-01",
    "fecha_salida": "2025-12-05",
    "cantidad_adultos": 2,
    "cantidad_ninos": 1,
    "cantidad_personas": 3,
    "precio_total": 600.00,
    "estado": "pendiente",
    "nombre_completo": "Juan P rez Garc a",
    "email": "juan@example.com",
    "telefono": "598 99123456",
    "metodo_pago": "tarjeta",
    "tarjeta_ultimos4": "1234",
    "nombre_habitacion": "Habitaci n Doble",
    "nombre_usuario": "Juan P rez"
  }
]
```

10.3.2. POST /reservas/

Descripción: Crea una nueva reserva con validaciones complejas.
Validaciones realizadas:

- Verifica que todos los campos requeridos estén presentes
- Valida formato de fechas (YYYY-MM-DD)
- Verifica que fecha_salida ¿fecha_entrada
- Confirma que la habitación existe
- Valida que adultos + niños sea positivo

Request:

```
POST http://localhost:5010/reservas/ HTTP/1.1
Content-Type: application/json

{
  "id_habitacion": 1,
  "fecha_entrada": "2025-12-01",
  "fecha_salida": "2025-12-05",
  "adultos": 2,
  "ninos": 1,
  "nombre_completo": "Juan P rez Garc a",
  "email": "juan@example.com",
  "telefono": "598 99123456",
  "metodo_pago": "tarjeta",
  "tarjeta_ultimos4": "1234"
}
```

Procesamiento en Backend:

1. Valida todos los campos requeridos
2. Convierte fechas a formato DATE
3. Verifica coherencia de fechas
4. Busca habitación y obtiene precio
5. Calcula: precio_total = precio_por_dia * (fecha_salida - fecha_entrada).days
6. En este caso: 150 * 4 días = 600.00
7. Busca usuario por email
8. Inserta registro en tabla reservas
9. Envía email de confirmación al administrador

Response (201 Created):

```
{
  "id": 42,
  "precio_total": 600.00,
  "estado": "pendiente",
  "mensaje": "Reserva creada correctamente"
}
```

Response (400 Bad Request):

```
{
  "error": "Faltan campos obligatorios: email, nombre_completo"
}
```

Response (400 Bad Request - Fechas):

```
{
  "error": "Formato de fecha inv lido. Usar YYYY-MM-DD"
}
```

Response (404 Not Found):

```
{
  "error": "Habitaci n no encontrada"
}
```

10.3.3. GET /reservas/usuario/{usuario_id}

Descripción: Obtiene todas las reservas de un usuario específico.
Request:

```
GET http://localhost:5010/reservas/usuario/1 HTTP/1.1
```

Response (200 OK):

```
[
  {
    "id": 42,
    "id_habitacion": 1,
    "id_usuario": 1,
    "fecha_entrada": "2025-12-01",
    "fecha_salida": "2025-12-05",
    "cantidad_adultos": 2,
    "cantidad_ninos": 1,
    "cantidad_personas": 3,
    "precio_total": 600.00,
    "estado": "pendiente",
    "nombre_habitacion": "Habitaci n Doble",
    "nombre_usuario": "Juan P rez"
  },
  {
    "id": 43,
    "id_habitacion": 2,
    "id_usuario": 1,
    "fecha_entrada": "2026-01-15",
```

```
    "fecha_salida": "2026-01-20",
    "cantidad_adultos": 2,
    "cantidad_ninos": 0,
    "cantidad_personas": 2,
    "precio_total": 1500.00,
    "estado": "confirmada",
    "nombre_habitacion": "Suite Premium",
    "nombre_usuario": "Juan Perez"
  }
]
```

11. Instalación y Configuración

11.1. Requisitos Previos

- Python 3.8 o superior
- MySQL 5.7 o superior
- Node.js (opcional, para herramientas de desarrollo)
- Git para control de versiones

11.1.1. Frontend

```
# Navegar a carpeta frontend
cd ../frontend

# Crear entorno virtual
python3 -m venv venv
source venv/bin/activate

# Instalar dependencias
pip install -r requirements.txt

# Ejecutar servidor
python app.py
```

11.2. Variables de Entorno

```
# .env archivo
FLASK_ENV=development
FLASK_SECRET_KEY=your_secret_key
DATABASE_URL=mysql://user:password@localhost:3306/hotel_db
MAIL_USERNAME=your_email@gmail.com
MAIL_PASSWORD=your_app_password
```

11.3. Cómo compilar / generar el PDF y ejecutar localmente

```
# Generar PDF (desde la carpeta ra z del proyecto)
pdflatex documentoOverleaf.tex
# o usar latexmk si est  disponible
latexmk -pdf documentoOverleaf.tex

# Ejecutar backend (puerto 5010)
cd backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python app.py

# En otra terminal: ejecutar frontend (puerto 5000)
cd frontend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python app.py
```

12. Documentación API

12.1. OpenAPI/Swagger

La documentación interactiva de la API está disponible en:
`http://localhost:5010/api/docs`
Ver archivo `openapi.yaml` para especificación completa.

12.2. Ejemplo de Solicitud

```
# Crear una reserva
curl -X POST http://localhost:5010/reservas \
  -H "Content-Type: application/json" \
  -d '{
    "usuario_id": 1,
    "habitacion_id": 5,
    "fecha_inicio": "2025-12-01",
    "fecha_fin": "2025-12-05"
  }'
```

13. Gestión del Proyecto

13.1. Metodología

Se utilizó metodología Agile con sprints de una semana, permitiendo:

- Iteración rápida sobre funcionalidades

- Feedback constante del equipo
- Adaptación a cambios de requisitos
- Entrega incremental de valor

13.2. Herramientas de Colaboración

- **GitHub:** Control de versiones y colaboración
- **GitHub Issues:** Gestión de tareas
- **GitHub Projects:** Tablero Kanban
- **Slack/Discord:** Comunicación del equipo

14. Dificultades Enfrentadas

Durante el desarrollo del proyecto Hotel IDS, el equipo LAST PUSH enfrentó varios desafíos técnicos que requirieron investigación, colaboración y resolución creativa de problemas. A continuación se detallan las principales dificultades y las soluciones implementadas:

14.1. 1. Inicialización de las Aplicaciones Flask (app.py)

Problema: Uno de los primeros obstáculos fue lograr que ambas aplicaciones Flask (frontend y backend) se levantaran correctamente sin conflictos de puertos y con todas las dependencias cargadas adecuadamente. Varios integrantes experimentaron errores de módulos faltantes, conflictos de versiones de bibliotecas y problemas de configuración del entorno virtual.

Síntomas:

- Error: `ModuleNotFoundError: No module named 'flask'`
- Conflicto de puertos: el puerto 5000 o 5010 ya estaban en uso
- Incompatibilidad de versiones entre Flask y sus extensiones
- Variables de entorno no configuradas correctamente

Solución Implementada:

1. Se crearon entornos virtuales independientes para frontend y backend usando `python3 -m venv venv`
2. Se instalaron todas las dependencias especificadas en `requirements.txt` con `pip install -r requirements.txt`
3. Se validó que Flask y sus extensiones (Flask-CORS, Flask-Mail, etc.) estuvieran correctamente instaladas
4. Se modificó la configuración en `app.py` para usar puertos explícitos (5000 para frontend, 5010 para backend)

5. Se agregaron archivos `.env` para gestionar variables de entorno de forma segura
6. Se utilizó `python app.py` en lugar de `flask run` para mayor control

14.2. 2. Inicialización de la Base de Datos MySQL

Problema: La configuración y levantamiento de la base de datos MySQL presentó múltiples obstáculos. El equipo enfrentó dificultades relacionadas con credenciales de acceso, permisos de usuario, y sincronización del esquema de la base de datos.

Síntomas:

- Error: `Access denied for user 'root'@'localhost'`
- Error: `Can't connect to MySQL server on 'localhost'`
- La base de datos no existía o las tablas no estaban creadas
- Inconsistencia entre el esquema esperado y el actual
- Error al ejecutar `init_db.py`: conexión rechazada o credenciales inválidas

Solución Implementada:

1. Se verificó que MySQL estuviera instalado y ejecutándose correctamente:

```
# Verificar si MySQL est  corriendo
brew services list  # en macOS
# o
systemctl status mysql  # en Linux
```

2. Se reinició el servicio MySQL:

```
brew services restart mysql-community  # macOS
# o
sudo systemctl restart mysql  # Linux
```

3. Se reseteó la contraseña del usuario root si era necesario
4. Se creó un usuario específico para la aplicación con permisos limitados:

```
CREATE USER 'hotel_user'@'localhost' IDENTIFIED BY '
   hotel_password';
GRANT ALL PRIVILEGES ON hotel_db.* TO 'hotel_user'@'localhost
   ';
FLUSH PRIVILEGES;
```

5. Se ejecutó el script de inicialización `init_db.py` que crea automáticamente las tablas y datos iniciales
6. Se agregó validación de conexión en `db.py` para verificar la conexión antes de realizar operaciones
7. Se documentaron las credenciales en un archivo `.env` para referencia consistente:

```
DATABASE_URL=mysql://hotel_user:hotel_password@localhost
   :3306/hotel_db
```

14.3. 3. Gestión de Contraseñas y Credenciales

Problema: La seguridad y consistencia en el manejo de contraseñas fue un desafío importante. El equipo debió implementar hash seguro de contraseñas, gestionar credenciales de base de datos y correo electrónico sin exponerlas en el código fuente.

Síntomas:

- Credenciales almacenadas en texto plano en el código
- Dificultad para cambiar credenciales sin modificar el código
- Vulnerabilidades de seguridad en autenticación de usuarios
- Errores al conectar con servidor de correo (Flask-Mail)

Solución Implementada:

1. Se implementó hashing de contraseñas usando bibliotecas seguras en el módulo de usuarios:

```
from werkzeug.security import generate_password_hash,
   check_password_hash

# Hash al guardar
hashed_password = generate_password_hash(password)

# Verificaci n al login
if check_password_hash(stored_hash, provided_password):
    # Contrase a v lida
```

2. Se creó un archivo `.env` (no versionado en Git) para almacenar credenciales:

```
DATABASE_PASSWORD=secure_password_123
MAIL_USERNAME=hotel_email@gmail.com
MAIL_PASSWORD=app_specific_password
FLASK_SECRET_KEY=your_secret_key_here
```

3. Se agregó validación y sanitización de inputs en formularios
4. Se implementó control de CORS restrictivo para evitar accesos no autorizados
5. Se documentó en el README el proceso seguro de configuración de credenciales

14.4. 4. Comunicación Frontend-Backend

Problema: Integrar correctamente la comunicación entre frontend y backend mediante HTTP/JSON presentó desafíos en validación de datos, manejo de errores y sincronización de estados.

Síntomas:

- Error CORS: `No 'Access-Control-Allow-Origin' header`
- Datos no llegan correctamente al backend

- Respuestas de error sin estructura clara
- Timeout en peticiones HTTP

Solución Implementada:

1. Se configuró Flask-CORS correctamente en el backend:

```
from flask_cors import CORS
app = Flask(__name__)
CORS(app, supports_credentials=True)
```

2. Se implementaron validaciones robustas en backend para todos los endpoints
3. Se estandarizó el formato de respuestas JSON (éxito y error)
4. Se agregó logging para debugging de peticiones HTTP

14.5. 5. Sincronización de Equipos y Control de Versiones

Problema: Trabajar en equipo con Git requirió definir workflows claros, resolver conflictos de merge y coordinar cambios simultáneos.

Solución:

- Se establecieron ramas específicas (main, develop, feature/*)
- Se realizaron commits frecuentes con mensajes descriptivos
- Se utilizó GitHub Projects para trackear tareas
- Se realizaron code reviews antes de mergear a main

15. Conclusión Final

El Trabajo Práctico Final "Hotel IDS" ha sido un proyecto integral que ha permitido al equipo LAST PUSH aplicar de manera práctica los conocimientos adquiridos en Ingeniería de Software, demostrando competencias en:

15.1. Logros Principales

1. **Diseño Arquitectónico:** Se logró diseñar una arquitectura cliente-servidor escalable separando adecuadamente las responsabilidades entre frontend y backend.
2. **Implementación Full-Stack:** El equipo desarrolló exitosamente una aplicación web completa con interfaz responsiva, API REST robusta, y base de datos relacional.
3. **Gestión de Proyectos:** Se aplicó metodología Agile con sprints semanales, utilización de GitHub Projects y una comunicación efectiva entre integrantes.
4. **Resolución de Problemas:** El equipo enfrentó y resolvió desafíos técnicos significativos incluyendo configuración de entornos, gestión de bases de datos, autenticación y seguridad.

5. **Documentación:** Se produjo documentación completa y profesional incluyendo diagramas de flujo, especificación de API, y este informe técnico.

6. **Seguridad:** Se implementaron medidas de seguridad importantes como hash de contraseñas, validación de datos, protección CORS y manejo seguro de credenciales.

15.2. Aprendizajes y Competencias Desarrolladas

Durante el desarrollo del proyecto, cada integrante del equipo LAST PUSH fortaleció sus habilidades en:

- **Programación Web:** HTML5, CSS3, JavaScript, Python y Flask
- **Bases de Datos:** Diseño de esquemas relaciones, SQL, MySQL y ORM
- **Desarrollo de APIs:** Creación de endpoints RESTful, validación de datos, manejo de errores
- **Trabajo en Equipo:** Control de versiones, resolución de conflictos, comunicación y coordinación
- **Ingeniería de Software:** Arquitectura de sistemas, metodologías Agile, documentación técnica
- **Gestión de Desafíos:** Troubleshooting, investigación independiente y resolución creativa de problemas

15.3. Estado Actual y Posibilidades Futuras

La plataforma Hotel IDS en su estado actual proporciona funcionalidades core completas para la reserva de hospedaje. Sin embargo, existen oportunidades de mejora y expansión tales como:

- Implementación de autenticación con OAuth (Google, Facebook)
- Sistema de pagos integrado (Stripe, PayPal)
- Notificaciones en tiempo real mediante WebSockets
- Aplicación móvil nativa (iOS/Android)
- Dashboard administrativo avanzado
- Sistema de calificaciones y comentarios de usuarios
- Optimizaciones de rendimiento y escalabilidad
- Integración con mapas más avanzados (geolocalización)

15.4. Reflexión Final

Este proyecto ha demostrado que el equipo LAST PUSH posee las competencias técnicas, organizacionales y humanas necesarias para desarrollar soluciones de software profesionales y de calidad. La capacidad de trabajar colaborativamente, resolver desafíos complejos, y documentar adecuadamente el trabajo realizado son habilidades fundamentales en la carrera de Ingeniería en Sistemas.

La plataforma Hotel IDS es un testimonio del trabajo dedicado, la perseverancia ante obstáculos técnicos, y el compromiso del equipo con la excelencia. Esperamos que este proyecto sirva como punto de partida para futuras mejoras y expansiones, y que el código y documentación produzidos sean de valor para otros estudiantes y profesionales.

16. Referencias

- Flask Documentation: https://flask.palletsprojects.com/
- MySQL Documentation: https://dev.mysql.com/doc/
- Bootstrap: https://getbootstrap.com/
- Leaflet.js: https://leafletjs.com/
- OpenAPI Specification: https://spec.openapis.org/