UNIVERSIDAD MAYOR DE SAN ANDRES

CARRERA DE INFORMATICA

PROGRAMACION WEB III

PROYECTO

Sistema de Reservas de Espacios Municipales

Estudiantes :

Apaza Patty Jurguin Cristhian	C.I.9117201	Nro:6
Flores Capriles Sergio Brandon	C.I.13754122	Nro:59
Guarachi Gutierrez Luis Manuel	C.I.9104176	Nro:66
Quispe Ticona Wiliam Herland	C.I.13763098	Nro:123
Vazquez Nicolas Belen America **(jefe de grupo)**	C.I.11104132	Nro:152

Docente:

Lic. Marcelo Aruquipa

Fecha:

1 de diciembre de 2025

LA PAZ - BOLIVIA

2025

1) **Título y descripción del proyecto**

   1.1 Objetivo del proyecto
   1.2 Funciones principales

2) **Requisitos y librerías adicionales**

   2.1 Requisitos
   2.2 Libreras y dependencias adicionales

3) **Instalación, estructura de archivos, código fuente, base de datos**

   3.1 Instalación paso a paso
   3.2 Estructura de archivos
   3.3 Codigo fuente clave
   3.4 Base de datos

4) **Manual de usuario**

   4.1 Roles
   4.2 Guía paso a paso
   4.3 Extra

5) **Documentación técnica**

   5.1 Diagrama de base de datos
   5.2 Configuración de variables de sistema
   5.3 Registro de progreso

6) **Bibliografía**

7) **Anexos**

1. Título y descripción del proyecto

1.1 Objetivo del proyecto

El objetivo del proyecto es desarrollar un sistema web que permita la **gestión de reservas de espacios municipales**, tales como auditorios, salones, canchas con tinglado u otros espacios públicos. Mediante esta plataforma, los solicitantes podrán pedir permiso para realizar eventos, subir justificación y una imagen del evento, mientras que los operadores municipales podrán aprobar o rechazar solicitudes. Además, se mostrará un **calendario público** con las reservas aprobadas, mejorando la transparencia y evitando conflictos de horarios.

1.2 Funcionalidades principales

- Registro de espacios municipales (nombre, ubicación, capacidad, descripción).
- Tipos de uso (evento, taller, reunión, etc.).
- Solicitud de reserva con formulario: solicitante, tipo de evento, justificación, fecha de inicio y fin, y carga de imagen.
- Estados de reserva: **pendiente**, **aprobado**, **rechazado**.
- Validación de disponibilidad (no permitir solapamiento de reservas).
- API REST para gestionar reservas.
- Calendario público con las reservas aprobadas, mostrando también la imagen del evento si fue subida.
- Interfaz para operadores que puedan ver las reservas pendientes y cambiar su estado.
- Notificaciones visuales (mensaje “solicitud enviada”).
- Almacenamiento de la imagen enviada por el solicitante, que se muestra en el calendario si la reserva se aprueba.

2. Requisitos, librerías adicionales

2.1 Requisitos

- Sistema operativo: compatible con Windows / Linux / macOS para desarrollo.
- Python 3.x.
- Base de datos SQL (por ejemplo, SQLite para desarrollo o MySQL / MariaDB para producción).

- Espacio para almacenamiento de imágenes (sistema de archivos o servicio de almacenamiento).
- Navegador moderno (Chrome, Firefox, Edge) para usar el frontend.

2.2 Librerías y dependencias adicionales

- **Django** (framework principal para backend).
- **Django REST Framework** para crear la API.
- Pillow para manejar imágenes (si usas ImageField).
- Configuración de archivos media (para guardar imágenes).
- En el frontend: **FullCalendar** (CDN) para el calendario público.
- JavaScript / CSS estándar para formularios.
- (Opcional) Librería para autenticación si quieres login para operadores (por ejemplo, Django contrib auth, django-allauth).

3. Instalación, estructura de archivos, código fuente, base de datos

3.1 Instalación paso a paso

1. Clonar el repositorio desde GitHub:
2. git clone https://github.com/tuUsuario/reservas_municipales.git
3. cd reservas_municipales
4. Crear entorno virtual:
5. python -m venv venv
6. source venv/bin/activate # Linux/Mac
7. venv\Scripts\activate # Windows
8. Instalar dependencias:
9. pip install -r requirements.txt
10. Ejecutar migraciones para la base de datos:
11. python manage.py makemigrations
12. python manage.py migrate

13. Ejecutar el servidor de desarrollo:
14. python manage.py runserver
15. Configurar la carpeta media/ para que Django guarde las imágenes subidas, en settings.py:
16. MEDIA_URL = '/media/'
17. MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
18. Configurar las URLs para servir medios durante el desarrollo.

3.2 Estructura de archivos

```
reservas_municipales/
│
├── manage.py
├── requirements.txt
├── municipal/
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│
├── reservas/
│   ├── migrations/
│   ├── models.py
│   ├── serializers.py
│   ├── views.py
│   ├── urls.py
│   └── templates/
│       ├── index.html
│       ├── reservar.html
│       └── panel_operador.html
│
├── media/  ← carpeta para imágenes subidas
│
└── docs/
    ├── bitacora.md
    └── manual_usuario.md
```

3.3 Código fuente clave

- **Models**: Espacio, Reserva, Solicitante, Uso.
- **Serializers**: para exponer API REST.
- **Views / ViewSets**: lógica para crear reserva, validar disponibilidad, cambiar estado.
- **Plantillas HTML**: calendario, formulario, panel de operador.
- **JavaScript**: fetch para consumir API y actualizar calendario.

3.4 Base de datos

- Se recomienda usar SQLite para desarrollo; para producción usar MySQL o similar.
- Tablas principales: Espacio, Solicitante, Uso, Reserva.
- Relaciones: reservas referencian espacio, solicitante y tipo de uso.
- Campo de imagen en Reserva para guardar la imagen subida por el solicitante (ImageField).

4. Manual de usuario

4.1 Roles

- **Solicitante**: Persona que pide reservar un espacio.
- **Operador**: Funcionario municipal que revisa las solicitudes y las aprueba o rechaza.
- **Invitado / público**: Cualquiera que visite el sitio para ver el calendario público.

4.2 Guía paso a paso

Como solicitante:

1. Ir a la página principal de “Reservar espacio” (reservar.html).
2. Llenar el formulario: nombre, tipo de evento, justificación, fecha inicio, fecha fin, y subir una imagen JPG si tienes.
3. Hacer clic en “Pedir autorización”.
4. Aparecerá un mensaje “Solicitud enviada” si todo sale bien.
5. Esperar a que un operador revise la solicitud; puede estar en estado “pendiente” hasta que se decida.

Como operador:

1. Inicar sesión (o ir a panel de administración si usas Django Admin).
2. Revisar la lista de reservas pendientes.
3. Evaluar la justificación, la imagen y las fechas.
4. Cambiar el estado a “aprobado” o “rechazado”.
5. Si aprueba, la reserva aparecerá en el calendario público con la imagen del evento.
6. Si rechaza, puedes dejar un comentario (opcional) o notificar al solicitante.

Para el público / invitado:

1. Ir a la página pública con el calendario.
2. Ver las reservas aprobadas, con su título, imagen, tipo de evento y quién lo solicitó (según cómo configures la visibilidad).
3. Hacer clic en un evento para ver más detalles en un “pop-up” (o ventana emergente).

4.3 Extra

- Puedes explicar cómo recargar el calendario después de que una reserva es aprobada.
- O cómo un solicitante puede ver el estado de su reserva (si añades una vista para eso).
- También puedes agregar una sección de “Preguntas frecuentes” para los solicitantes (“¿Qué pasa si mi solicitud es rechazada?”, “¿Puedo modificar mis fechas?”, etc.).

5. Documentación técnica

5.1 Diagrama de base de datos

- Incluye un diagrama ER (Entidad-Relación) con las tablas: Espacio, Solicitante, Uso, Reserva.
- Relación: Reserva tiene llave foránea a Espacio, Solicitante y Uso.
- Propiedades: campos principales (id, nombre, fecha_inicio, fecha_fin, estado, imagen…).

5.2 Configuración de variables del sistema

- Archivo .env o settings.py debe contener variables como:

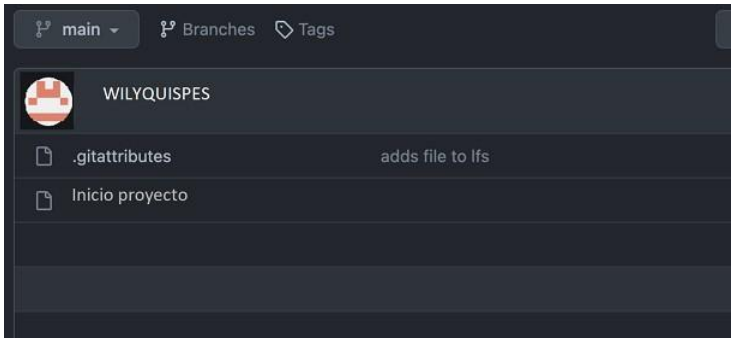
- DEBUG = True/False
- DATABASE_URL o configuraciones de base de datos (HOST, USER, PASSWORD)
- MEDIA_ROOT y MEDIA_URL para almacenamiento de imágenes
- (Opcional) variables para correo electrónico si envías notificaciones

5.3 Registro de progreso (Bitácora de desarrollo)

- Haz un registro cronológico de lo que hiciste:
  1. **Commit inicial**: creación del proyecto Django.
  2. **Commit modelo**: creación de modelos Espacio, Uso, Solicitante, Reserva.
  3. **Commit API**: serializadores y viewsets para API REST.
  4. **Commit frontend**: plantillas, formulario, calendario con FullCalendar.
  5. **Commit lógica de validación**: verificar solapamiento de reservas.
  6. **Commit subida de imagen**: añadir ImageField en modelo, guardar y servir imágenes.
  7. **Commit estado**: permitir cambiar estado de reserva desde operador.
  8. **Commit mejoras UI**: mensajes de “solicitud enviada”, formateo de interfaz, estilos CSS.
  9. **Commit documentación**: añadir manual, README, documentación técnica.

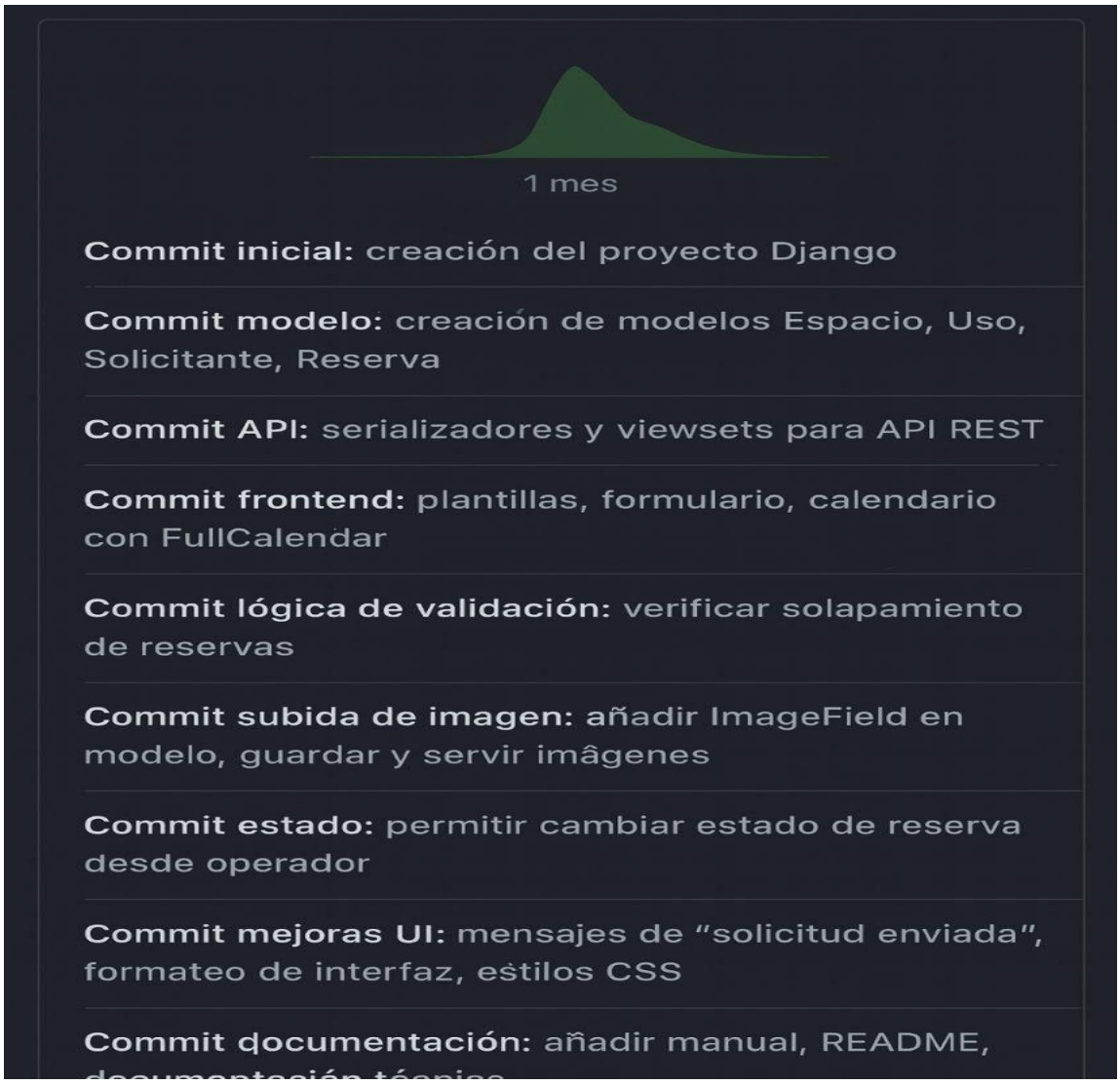
5.4 GitHub

Los colaboradores del proyecto fueron los mismos estudiantes



Donde en el repositorio del jefe de grupo se creó el proyecto y después de empezó a trabajar junto a los demás compañeros, donde se realizaron los diferentes commints

El trabajo duro casi un mes de trabajo, en donde se trabajo el proyecto de manera grupal.

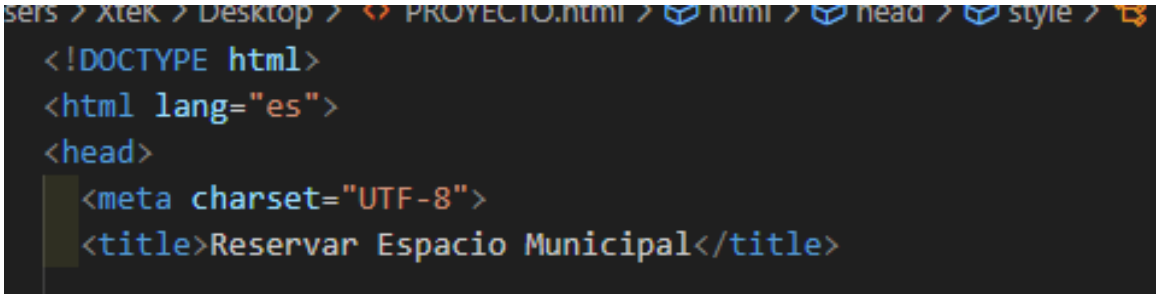


6. Bibliografía

- Manual técnico y guía de documentación: “Guía Práctica para el Desarrollo de Software: un Manual Técnico Completo”. (desarrollodesoftware)
- Plantilla para documento de requisitos de software (SRS), Asana. (Asana)
- Cómo documentar proyectos de software: “¿Cómo documentar proyectos de software? Guía sencilla.” (educaopen.com)
- Guía para la creación de especificación técnica: “Guía para crear documento técnico de especificación” (Document360). (Document360)
- Guía para documentación de proyectos (manuales de usuario, instalación, referencia): “DI06 – Documentación de aplicaciones”. (sarreplec.caib.es).

7. Anexos

Se realizo usando html y estilo css.



```
<!-- FullCalendar CSS (CDN) -->
<link href="https://cdn.jsdelivr.net/npm/fullcalendar@6.1.19/index.global.min.css" rel="stylesheet" />

<style>
  body {
    font-family: Arial, sans-serif;
    margin: 40px;
  }

  header {
    text-align: center;
    margin-bottom: 30px;
  }

  form {
    max-width: 600px;
    margin: 0 auto 40px auto;
    padding: 20px;
    border: 1px solid #ccc;
    border-radius: 8px;
  }

  form div {
    margin-bottom: 15px;
  }

  form label {
    display: block;
```

```
      .fc-event {
        cursor: pointer;
        /* Ajuste para imagen: la vamos a inyectar dinámicamente con eventContent */
      }
    </style>
  </head>
  <body>

    <header>
      <h1>Solicitud de Reserva</h1>
      <p>Pide autorización para reservar un espacio municipal</p>
    </header>

    <form id="reserva-form">
      <div>
        <label for="solicitante-nombre">Nombre del solicitante:</label>
        <input type="text" id="solicitante-nombre" name="nombre" required>
      </div>

      <div>
        <label for="uso-tipo">Tipo de evento (Uso):</label>
        <select id="uso-tipo" name="uso" required>
          <option value="">Selecciona una opción</option>
          <option value="Conferencia">Conferencia</option>
          <option value="Taller">Taller</option>
          <option value="Reunión">Reunión</option>
          <option value="AnimeKai">Anime Kai</option>
```

Hacemos la conexión Json

```
<!-- FullCalendar JS (CDN) -->
<script src="https://cdn.jsdelivr.net/npm/fullcalendar@6.1.19/index.global.min.js"></script>

<script>
  document.addEventListener("DOMContentLoaded", function() {
    const form = document.getElementById("reserva-form");
    const mensaje = document.getElementById("mensaje");

    form.addEventListener("submit", function(event) {
      event.preventDefault();
      mensaje.textContent = "";

      const nombre = document.getElementById("solicitante-nombre").value;
      const uso = document.getElementById("uso-tipo").value;
      const justificacion = document.getElementById("justificacion").value;
      const imagenInput = document.getElementById("imagen-evento");
      const imagenFile = imagenInput.files[0];

      const formData = new FormData();
      formData.append("nombre", nombre);
      formData.append("uso", uso);
      formData.append("justificacion", justificacion);
      if (imagenFile) {
        formData.append("imagen", imagenFile);
      }
```

Creamos la petición backend Djando

```
// Aquí hacemos la petición al backend Django (tu endpoint POST /api/reservas o uno dedicado)
fetch("/api/reservas/", {
  method: "POST",
  body: formData
})
.then(response => {
  if (!response.ok) throw new Error("Error al enviar solicitud");
  return response.json();
})
.then(data => {
  console.log("Respuesta del servidor:", data);
  mensaje.textContent = "Solicitud enviada";
  form.reset();
  // recargar el calendario para incluir (o marcar) la nueva reserva si es aprobada más adelante
  calendar.refetchEvents();
})
.catch(error => {
  console.error("Error:", error);
  mensaje.textContent = "Hubo un problema al enviar la solicitud";
});
});
```

Iniciamos el calendario, creando lo necesario para que este funcione.

```
// Inicializar calendario
const calendarEl = document.getElementById("calendar");
const calendar = new FullCalendar.Calendar(calendarEl, {
  initialView: "dayGridMonth",
  headerToolbar: {
    left: "prev,next today",
    center: "title",
    right: "dayGridMonth,timeGridWeek,timeGridDay"
  },
  navLinks: true,
  nowIndicator: true,
  weekNumbers: true,

  // inyectar custom content (como imagen) dentro del evento
  eventContent: function(arg) {
    // arg.event.extendedProps debe tener propiedades como imagen, estado, solicitante, uso
    const { imagen_url, estado, solicitante, uso } = arg.event.extendedProps;

    const container = document.createElement("div");

    // Si hay imagen, usar como fondo del mini-evento
    if (imagen_url) {
      const img = document.createElement("img");
      img.src = imagen_url;
      img.style.width = "100%";
      img.style.height = "auto";
      img.style.display = "block";
      img.style.borderRadius = "4px";
```

Verificamos el programa ya hecho y terminado donde se puede realizar la solicitud.

Al enviar eso se puede apreciar que nos sale que no hubo ningún problema al realizar dicha petición, también mostrando el calendario y fecha de hoy.