

Planificador Inteligente de Eventos

Informe del Proyecto de Programación

Guillermo Rodríguez Rodríguez

Ciencia de la Computación

Universidad de La Habana

`guillermo2rodriguez.06@gmail.com`

4 de febrero de 2026

Resumen

En este informe se presenta el desarrollo del proyecto de programación titulado “*Planificador Inteligente de Eventos*”. El objetivo es crear una aplicación de software que permita la planificación de eventos a partir de los recursos de un inventario limitado. El dominio elegido para desarrollar la aplicación es un cine-teatro, en el cual los principales eventos son: proyecciones fílmicas, obras teatrales y conciertos musicales. La aplicación permite al usuario efectuar las siguientes acciones: agregar eventos (evitando colisiones en el uso de recursos), ver detalles de eventos, eliminar eventos programados, ver información de los recursos del inventario y la recomendación de un horario para agregar un evento. La aplicación fue desarrollada con el lenguaje Python, utilizando el framework Streamlit. Para el control de versiones se empleó Git. El editor de código empleado fue Visual Studio Code. La aplicación cuenta con un archivo .json que permite la persistencia de los datos. El resultado ha sido satisfactorio, con una aplicación de software robusta que permite cumplir todos los objetivos propuestos, y que cuenta con una interfaz gráfica amigable para realizar las acciones descritas con anterioridad.

Índice

1. Introducción	4
2. Explicación detallada de las restricciones entre los eventos y los recursos	4
2.1. Recursos necesarios para cada evento	4
2.2. Restricciones entre recursos	4
2.3. Restricciones entre eventos	5
3. Tecnologías y Herramientas	5
4. Diseño del Sistema	5
4.1. Arquitectura General	5
4.2. Diagrama del sistema	7
5. Implementación	7
5.1. “Inicio.py”	8
5.2. “pages”	9
5.3. “Functions”	10
5.4. “Intellisense”	14
5.5. “Data”	15
6. Resultados	16
7. Conclusiones	18
8. Repositorio	18

1. Introducción

Este proyecto de programación consiste en una aplicación de software que permite planificar eventos a partir de un grupo limitado de recursos de un inventario. El dominio escogido para esta aplicación fue un cine-teatro, en el cual se pueden planificar tres tipos de eventos: proyección de películas, obras teatrales y/o conciertos musicales.

En principio, la idea de escoger un cine como dominio vino dada por la pasión cinéfila de mi familia, en especial de mi madre; también influyó el hecho de vivir relativamente cerca del cine Chaplin, de gran renombre en La Habana. Después de darle vueltas, me pareció insuficiente el hecho de solo gestionar proyecciones de películas, entonces decidí expandir el dominio a un cine-teatro, con el objetivo de agregar nuevos tipos de eventos y nuevas restricciones de recursos. Para completar la concepción de los eventos del dominio, agregué la opción de un concierto musical como otro tipo de evento, ya que en la actualidad suelen escogerse los escenarios de los cines-teatros para realizar estas funciones.

El cine-teatro (ficticio) que sirve como dominio de la aplicación se llama "La Cuarta Pared", el cual cuenta con los siguientes recursos a gestionar: 6 salas de cine con diferentes capacidades y equipos instalados, y los trabajadores del recinto (técnicos de sonido, de iluminación, del proyector, personal de limpieza y de seguridad). Cada sala de cine es tratada como un recurso independiente, mientras que los trabajadores son gestionados por cantidad. Un aspecto que influye en el uso de cada recurso es la asistencia del público, ya que no se puede escoger una sala en la que no quepan todos los asistentes, además de que según esa cantidad se pueden necesitar más o menos trabajadores.

2. Explicación detallada de las restricciones entre los eventos y los recursos

2.1. Recursos necesarios para cada evento

Todos los eventos requieren los siguientes recursos en común: una sala desocupada en el horario deseado y cuya capacidad permita acoger al total de asistentes; la cantidad requerida de personal de limpieza y seguridad según la asistencia de público; y la cantidad requerida de técnicos de sonido según la sala escogida. Si no hay suficientes trabajadores disponibles para la sala, no se puede realizar el evento. A continuación, se detallan las especificidades según el tipo de evento:

- Proyección Fílmica: admite cualquiera de las 6 salas del recinto. Necesita la cantidad requerida de operadores de proyección según la sala escogida.
- Obra Teatral: solo admite salas que contengan un escenario modular, que puede ser cualquiera entre las salas 4, 5 o 6. Necesita la cantidad requerida de técnicos de iluminación según la sala escogida.
- Concierto Musical: solo admite salas que contengan un escenario modular, que puede ser cualquiera entre las salas 4, 5 o 6. Necesita la cantidad requerida de técnicos de iluminación según la sala escogida.

2.2. Restricciones entre recursos

- Las salas 1, 2 y 3 necesitan al menos 1 técnico de sonido y 1 operador de proyector.

- Las salas 4 y 5 necesitan al menos 2 técnicos de sonido, 2 técnicos de iluminación y 2 operadores de proyector (según el evento que se quiera planificar).
- La sala 6 necesitan al menos 3 técnicos de sonido, 3 técnicos de iluminación y 3 operadores de proyector (según el evento que se quiera planificar).
- Para una asistencia menor de 100 personas, se necesitan al menos 1 persona de limpieza y 1 de seguridad. Para una asistencia entre 100 y 200 personas, se necesitan al menos 2 personas de limpieza y 2 de seguridad. Para una asistencia mayor de 200 personas, se necesitan al menos 3 personas de limpieza y 3 de seguridad.

2.3. Restricciones entre eventos

- Las proyecciones filmicas y las obras teatrales pueden programarse en horarios simultáneos o que coincidan en algún período de tiempo, si y solo si ocurren en salas distintas y hay suficientes trabajadores en el período de tiempo.
- Los conciertos musicales no pueden coincidir con otros eventos (ni siquiera otros conciertos musicales) en ningún período de tiempo.
- El cine-teatro tiene un horario para programar eventos cada día: inicia a las 8:00 y culmina a la hora de fin del último evento programado en dicho día.

3. Tecnologías y Herramientas

- Lenguaje: Python.
- Framework: Streamlit.
- Control de versiones: Git.
- Entorno: VS Code.
- Persistencia de datos: Archivo .json.

****Nota**:** En el archivo *“Requirements.txt”* se especifican todos los paquetes necesarios para ejecutar correctamente la aplicación.

4. Diseño del Sistema

4.1. Arquitectura General

- Archivo *“Inicio.py”*: es la página principal de la aplicación de Streamlit. La bienvenida ;).
- Carpeta *“pages”*: esta carpeta contiene el resto de páginas que conforman la aplicación de Streamlit, integradas con la interfaz gráfica y el llamado a las funciones del backend necesarias. *“Agregar Eventos”* es la página que permite agregar los eventos; está implementada de modo que solo se puedan agregar eventos en un rango de 30 días, desde la fecha actual. *“Lista de Eventos”* es la página que permite ver la lista de todos los eventos planificados, en un rango de 30 días a partir de la fecha actual;

también permite ver detalles de los eventos y eliminar eventos. “*Información de Recursos*” es la página que permite ver la información de todos los recursos del recinto: la cantidad de trabajadores y las salas de cine, con sus capacidades, equipamientos y necesidades.

- Carpeta “*Functions*”: esta carpeta contiene la mayoría de las funciones que constituyen el backend de la aplicación. Específicamente: “*AuxFuncs.py*” incluye dos funciones auxiliares que ocasionalmente resultan útiles en el momento de buscar una fecha u ordenar una lista de fechas; “*Events.Funcs.py*” posee tres funciones claves de la aplicación: agregar un evento, revisar características de un evento y eliminar un evento; “*RevResources.py*” contiene todas las funciones que permiten gestionar los recursos de forma que no existan choques que imposibiliten la planificación de un evento; “*SaveData.py*” incluye dos funciones: una que permite acceder a la información guardada en el archivo .json y otra que permite guardar la información en el archivo .json.
- Carpeta “*Intellisense*”: esta carpeta contiene otra parte importante del backend. Aquí se encuentra el archivo “*Funcs.py*”, el cual contiene funciones que permiten la recomendación de un nuevo horario para agregar un evento, en caso de que en el horario escogido inicialmente no haya sido posible programar el evento.
- Carpeta “*Gallery*”: carpeta poco interesante en cuanto a código, pero contiene imágenes bonitas :) que pueden ser vistas en la página inicial de la aplicación.
- Carpeta “*Data*”: carpeta que contiene el archivo .json, donde se guardan todos los eventos planificados y la información de los recursos del cine-teatro.

4.2. Diagrama del sistema

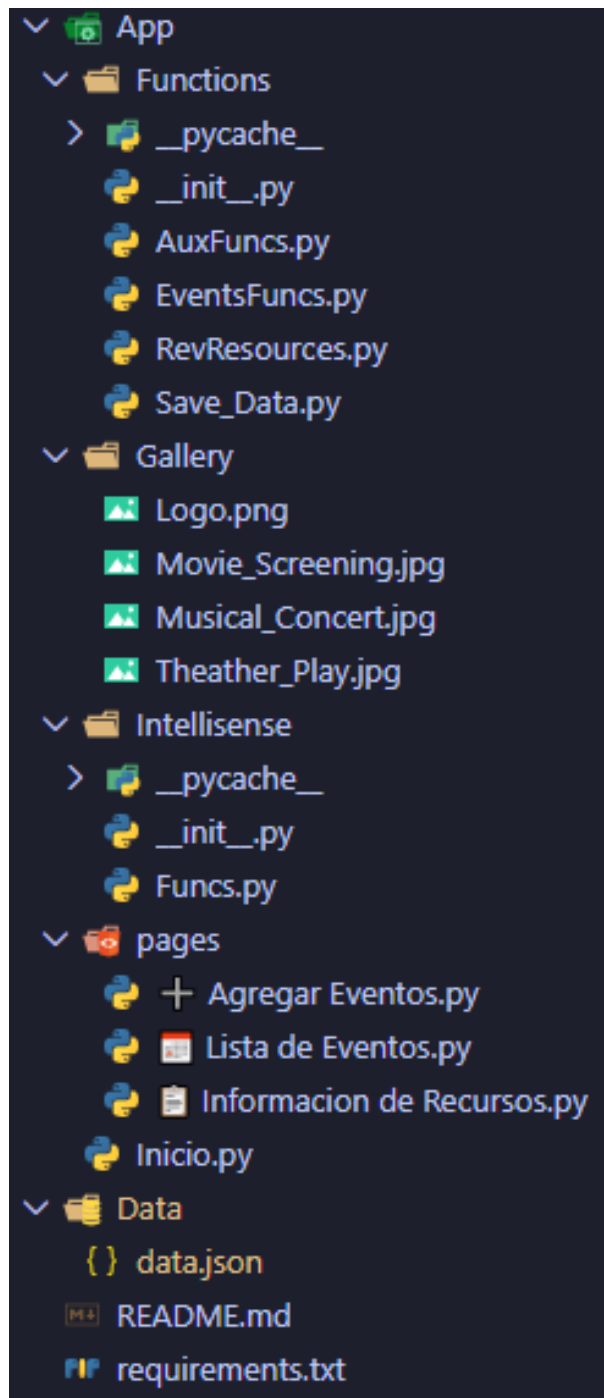


Figura 1: Arquitectura general del sistema

5. Implementación

A continuación se ofrecen detalles sobre cada carpeta y archivo del sistema, con una explicación breve y concisa sobre ciertos fragmentos del código. También se justifican las decisiones tomadas durante la implementación de cada funcionalidad.

5.1. “Inicio.py”

Esta página tiene dos objetivos claros: dar la bienvenida a la aplicación ;) y cargar los datos guardados en el archivo .json.

El fragmento de código 1 muestra cómo se implementó esta segunda funcionalidad en el programa; se tiene en cuenta que si las llaves “Recursos” y “Eventos” no se encuentran guardadas en el “st.session-state” (que permite guardar información entre páginas), entonces no se ha accedido a la información del .json, por tanto se llama a la función “GetData()” para guardar el .json como un diccionario.

Listing 1: Código que carga los datos del .json al inicializar la aplicación

```
1 #Este fragmento de código será visto al inicio de todas las
  páginas, fue una necesidad al descubrir que
2 #al transitar de una página a otra sin esperar que cargue
  completa o
3 #al inicializar la aplicación desde otra página que no fuera la
  inicial
4 #ocurre que no se llegaban a cargar los datos del .json.
5 if "Recursos" not in st.session_state or "Eventos" not in st.
  session_state:
6     appData = Save_Data.GetData()
```

A continuación, se efectúan ciertos cambios iniciales en el diccionario extraído anteriormente, quitando fechas desfasadas de tiempo y eliminando fechas desfasadas que se acumulen, como se muestra en el siguiente fragmento de código.

Listing 2: Dos fragmentos que editan la información antes de guardarla en “st.session-state”

```
1 #Revisa todas las fechas previas a la fecha actual y las ''
  desactiva''
2 today = dt.date.today()
3 for e in appData["Eventos"]:
4     d = dt.date(e["id"][0], e["id"][1], e["id"][2])
5     if d < today:
6         e["In_Time"] = False
7     else: break
8
9 #Cuando se acumulan demasiados días en el .json, se eliminan
  todos los desactivados.
10 if len(appData["Eventos"]) >= 40:
11     for e in appData["Eventos"]:
12         if not e["In_Time"]:
13             appData["Eventos"].remove(e)
14
15 #Se guarda la información
16 Save_Data.SaveData(appData)
```

Después, se guarda la información del .json en el “st.session-state” de forma separada: por un lado los “Recursos” y por el otro los “Eventos”. Esta información puede preservarse al cambiar de página, o sea, no se pierde cuando se reinicia una página de Streamlit.

Listing 3: Guardado de la información en el “st.session-state”


```
1 st.session_state["Recursos"] = appData["Recursos"]
2 st.session_state["Eventos"] = appData["Eventos"]
```

5.2. “pages”

Esta carpeta no fue creada tanto por comodidad de organización (que también), sino más bien por una necesidad de la interfaz visual de la aplicación. Al crear esta carpeta e incluir el resto de páginas, Streamlit las reconoce y las agrupa en un sidebar a la izquierda de la pantalla, desde el cual se puede acceder a cualquier página en cualquier momento.

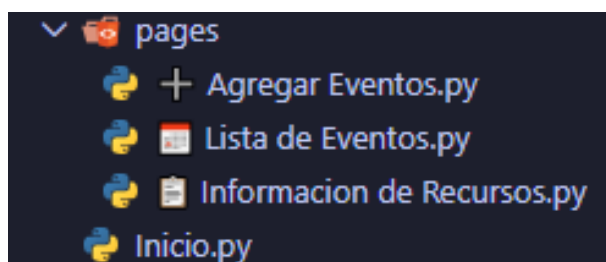


Figura 2: Páginas de la aplicación de Streamlit

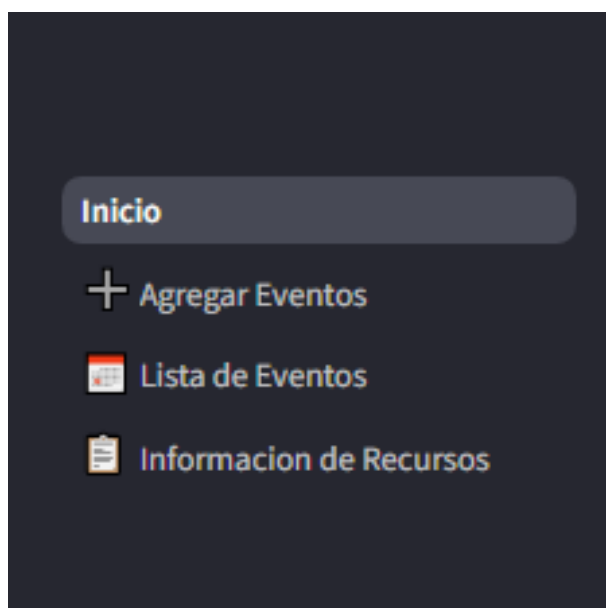


Figura 3: Fragmento de la Sidebar

La primera página “Agregar Eventos.py” posee toda la interfaz que posibilita la planificación de un evento. Ofrece las siguientes opciones: seleccionar el tipo de evento (Proyección Filmica, Obra Teatral, Concierto Musical), establecer una fecha (en un rango máximo de 30 días, a partir de la fecha del día siguiente), una hora (el recinto solo permite iniciar eventos desde las 8:00) y una duración para el evento (hasta 4 horas y 59 minutos), introducir la cantidad de público asistente (máximo de 300 personas), escoger la sala del evento, seleccionar la cantidad de trabajadores para la sala, introducir un identificador del evento (nombre de la película, nombre de la obra teatral, artistas del concierto musical) y ofrecer una descripción del evento (esto es opcional). Según el tipo de evento

escogido, se controla por un condicional cuál fragmento de código ejecutar para ofrecer las opciones de programar el evento. Cuando se escoge una fecha, una hora y una duración, se realiza el llamado a la función “RevResources.Disponibility(evens, fecha, horaInicial, horaFinal)”, la cual crea un diccionario con todos los recursos disponibles en el horario escogido, restando los recursos que ya estén siendo usados por otro evento alrededor del horario escogido. Según la sala y las cantidades de trabajadores escogidos se realizan llamados a las funciones que revisan que no existan colisiones de recursos con otros eventos en el horario escogido. Cuando se detecta algún choque de recursos o eventos, se muestra una advertencia en rojo que indica el choque de recursos que se detectó (por ejemplo: sala 1 ocupada, no hay técnicos de sonido disponibles) o alguna incompatibilidad entre capacidad de la sala y/o cantidad de trabajadores (por ejemplo: si hay asistencia de 200 personas no se puede ocupar la sala 1 porque no tiene esa capacidad); si la advertencia es del primer tipo descrito, se lanza otro mensaje de color verde que indica otro horario donde se puede realizar el evento con las características deseadas. Cuando se coloca toda la información y no existen colisiones de recursos, aparece el botón “Agregar Evento”, que cuando es presionado realiza el llamado a la función “EventsFuncs.AddEvent(evens, selection, fecha, horaInicial, horaFinal, nombre, descripcion, necesidades)”, la cual agrega el evento en el día seleccionando, y guarda la información en el .json; también cambia a la página “Lista de Eventos.py”, para que el usuario pueda visualizar el evento ya planificado.

La segunda página “Lista de Eventos.py” permite ver todos los eventos programados en el período de tiempo desde la fecha actual hasta 30 días después. Se ofrecen dos opciones: ver todos los eventos de todos los días o ver los eventos de un día seleccionado (para escoger la opción se ofrece una barra de búsqueda). Cuando se muestra cada evento, se muestra con el formato de la figura 5. La decisión de colocar los botones de “Ver Detalles” y “Eliminar” junto a los eventos fue puramente estética. Cuando se toca el primer botón, se llama a la función “EventsFuncs.ViewDetails(i, col1)” que permite ver las características del evento. Cuando se toca el segundo botón (aclarar que se debe presionar dos veces para ver su efecto), se llama a la función “EventsFuncs.DeleteEvent(i)”, que se encarga de inhabilitar el evento.

🎬 08:00-09:01 'Proyeccion Filmica: Pelicula 1'	Ver Detalles	Eliminar
🎭 14:00-15:01 'Obra de Teatro: Obra 1'	Ver Detalles	Eliminar
🎵 15:40-16:41 'Concierto Musical: Concierto 1'	Ver Detalles	Eliminar

Figura 4: Formato en que se muestran los eventos

La tercera página “Información de Recursos.py” ofrece toda la información sobre los recursos del cine-teatro. Se divide en dos secciones. La sección “Salas” muestra la capacidad de cada sala, los equipos instalados en la sala y los trabajadores que se necesitan en la sala. La sección “Personal” muestra los grupos de trabajadores del recinto, con sus respectivas cantidades.

5.3. “Functions”

Esta carpeta contiene la mayoría de los archivos que conforman el backend, concretamente los que se encargan de gestionar los recursos, trabajar con los eventos y acceder y guardar la información del archivo .json.

El archivo “SaveData.py” posee dos funciones que se encargan de trabajar en el archivo .json. Por un lado, “GetData()” permite extraer la información del .json y guardarla en un diccionario; presenta una etiqueta llamada “@st.cache-data”, la cual evita que esta función se vuelva a ejecutar cuando se reinicia la página de Streamlit, lo cual significa una mejora del rendimiento de la aplicación. La segunda función es “SaveData(data)” que recibe un parámetro que recoge toda la información que se desea guardar en el .json; la última línea del código de la función permite que se limpie la caché de la aplicación, esto viene enlazado con la etiqueta descrita en el método anterior. Cada vez que se llama a esta última función, se presenta un patrón de código como el del fragmento 5, en el cual se juntan las dos separaciones originales que se hicieron del diccionario del .json en el “st.session-state” en un único diccionario “data” el cual se pasa como parámetro de la función de guardado.

Listing 4: Código de las funciones “GetData()” y “SaveData(data)”

```

1 @st.cache_data
2 def GetData() -> dict:
3     with open("Data\data.json", "r") as Data:
4         appData = json.load(Data)
5     return appData
6
7 def SaveData(data):
8     with open("Data\data.json", "w") as Data:
9         json.dump(data, Data, indent=4, ensure_ascii=True)
10    st.cache_data.clear()

```

Listing 5: Patrón que se da cuando se llama a la función de guardado

```

1 data : dict = {}
2 data["Eventos"] = evens # evens es una variable que contiene a st
   .session_state["Eventos"]
3 data["Recursos"] = res # res es una variable que contiene a st.
   session_state["Recursos"]
4 Save_Data.SaveData(data)

```

El archivo “RevResources.py” contiene todas las funciones que permiten validar un evento, es decir, revisar que no existan colisiones en el uso de recursos. Primeramente, es necesario referirse a la función “Disponibility(events: list, day: date, tm-Inicial: time, tm-Final: time)”, la cual es llamada cuando se escoge una fecha y una hora de inicio y final del evento, para analizar los recursos disponibles en ese momento, restando aquellos que estén destinados a otros eventos que coincidan en algún intervalo de tiempo con el horario escogido. El siguiente diccionario muestra las condiciones iniciales en que todos los recursos están liberados (está dentro del código de la función), la función evalúa por cada evento del día si coincide en algún punto con el horario del evento que se desea planificar, en caso afirmativo resta las cantidades de trabajadores e inhabilita (asigna False) la sala que esté ocupada.

Listing 6: Diccionario que agrupa todos los recursos del recinto

```

1 dispoms = {
2     "personal de limpieza": 8,
3     "tecnicos de sonido": 6,
4     "tecnicos de iluminacion": 6,

```

```

5         "operadores de proyeccion": 6,
6         "personal de seguridad": 8,
7         "salas": [True, True, True, True, True, True]
8     }

```

El resto de funciones se pueden clasificar en tres categorías:

- Validan uso correcto de recursos: analizan que las salas escogidas y las cantidades de trabajadores escogidas coincidan con las dependencias en cuanto a capacidad, asistencia de público y necesidad de cada sala.

Listing 7: Funciones que validan el uso correcto de los recursos

```

1 # Analiza si la sala escogida tiene capacidad suficiente para
  la cantidad de asistentes
2 def Review_Capacity(sala: dict, assistance: int, k: bool) ->
  bool: (...)
3
4 # Analiza si la cantidad de personal seleccionado esta acorde
  a la asistencia del publico (personal de limpieza y de
  seguridad)
5 def Review_PersCapacity(persL: int, persS: int, assistance:
  int, k: bool) -> bool: (...)
6
7 # Analiza si la cantidad de personal seleccionado esta acorde
  a la sala del evento (tecnicos de sonido, tecnicos de
  ilumiancion y operadores de proyeccion)
8 def Review_PersPlace(persS: int, persP: int, persL: int, id:
  int, k: bool) -> bool: (...)
9
10 # Analiza si la sala escogida posee un escenario modular (
    estas son: #4, #5, #6)
11 def Review_Scene(id: int, k: bool) -> bool: (...)

```

- Revisan colisiones de recursos: analizan si la sala escogida está disponible en el horario seleccionado y si la cantidad de trabajadores disponibles permite realizar el evento. Si una de estas validaciones falla, se llama a la función que recomienda un nuevo horario.

Listing 8: Funciones que revisan colisiones de recursos

```

1 # Analiza si la sala escogida se encuentra disponible en el
  horario establecido
2 def Review_Place(id: int, salas: list, k: bool) -> bool:
  (...)
3
4 # Analiza, a partir del personal disponible y el personal
  necesario, si es posible efectuar el evento
5 def Review_Personal(personal_disponible : dict,
  personal_necesario: dict, k: bool) -> bool: (...)
6
7 # Revisa si todas las salas estan disponibles en el momento
8 def Check_Places(salas: list, k: bool) -> bool: (...)

```

```

9
10 # Revisa si el personal esta disponible
11 def Check_Personal(res: dict, typ: str, k: bool) -> bool:
    (...)

```

- Revisan choques de eventos: analizan que en el horario escogido no se estén efectuando conciertos musicales ni cualquier otro tipo de evento; esto es necesario por las realciones descritas sobre los conciertos musicales en la seccion 2. Si una de estas validaciones falla, se llama a la función que recomienda un nuevo horario.

Listing 9: Revisan choques de eventos

```

1 # Revisa si hay un concierto musical en el horario
  seleccionado
2 def Check_MC(events: list, day: date, tm1: time, tm2: time, k
  : bool) -> bool: (...)
3
4 # Revisa si es posible programar un concierto musical en un
  determiando horario (no puede coincidir con otros eventos)
5 def Check_Evs(events: list, day: date, tm1: time, tm2: time,
  k: bool) -> bool:

```

También existe la función “Review-Events(events: list)”, la cual revisa que hayan eventos activos (que no estén eliminados) en un día. Esta función es llamada desde la página “Lista de Eventos.py”, y permite determinar si en un día hay eventos programados que no hayan sido eliminados.

El archivo “EventsFuncs.py” posee tres funciones importantes de la aplicación.

- “AddEvent(events: list, typ: str, day: date, tm-Init: time, tm-End: time, name: str, description: str, recursos: dict)”: recibe como parámetros todos los datos sobre fecha, hora de inicio, hora final, sala, trabajadores, nombre y descripción seleccionados en la página “Agregar Eventos.py” y crea un diccionario que incluye toda esa información, el cual constituye el evento que se guarda en el .json. Esta función busca el día seleccionado en st.session-state[“Eventos”]; si lo encuentra busca la lista de los eventos de ese día y lo agrega, de forma trucada (método insert()) para que quede organizado con el resto de eventos; si no lo encuentra, crea un diccionario como el del fragmento 11 y agrega el evento en la llave que recoge la lista de eventos, la cual esta vacía. Después, realiza el llamado a la función de guardado.

Listing 10: Diccionario que representa un evento

```

1 newEvent = {
2     "activo": True,
3     "nombre": name,
4     "tipo": typ,
5     "descripcion": description,
6     "fecha": day.strftime('%B, %d, %Y'),
7     "hora de inicio": tm_Init.strftime('%H:%M'),
8     "hora de fin": tm_End.strftime('%H:%M')
9 }
10 newEvent.update(recursos) # Los recursos ingresan juntos
    en un diccionario, se usa la funcion update() para
    agregarlos al diccionario anterior

```

Listing 11: Diccionario que representa una jornada en el cine-teatro

```
1 {  
2     "id": (day.year, day.month, day.day), # Dia  
3     "Lista_Eventos": [], # Lista de eventos del dia  
4     "In_Time": True # Indica si el dia esta desfasado de  
5         tiempo o no (respecto a hoy)  
}
```

- “ViewDetails(event: dict, col)” : esta función permite ver los detalles de los eventos. Recorre todas las llaves del diccionario del evento y muestra en la aplicación los detalles del evento.
- “DeleteEvent(event: dict)” : esta función permite eliminar un evento. Lo que hace es muy simple: accede a la llave “activo” le da el valor False, con lo cual “inhabilita” el evento, el cual es ignorado por el resto de funciones del programa. Después, llama a la función de guardado.

El archivo “AuxFuncs.py” tiene dos funciones auxiliares, que permiten realizar dos acciones ocasionalmente útiles: buscar una fecha y ordenar fechas. La función “BS-Date(l: list, d: date)” presenta el algoritmo de búsqueda binaria, aplicado a la búsqueda de una fecha concreta en una lista; es posible aplicarlo porque en Python los tipos date se pueden comparar. La función “Sort-Dates(l: list)” permite dada una lista ordenar todas las fechas, aplicando el algoritmo recursivo Merge-Sort, escogido por su eficiencia.

****Nota****: Cuando se hace referencia a “listas” en el texto anterior, se habla de la lista de diccionarios, donde cada elemento representa una jornada de eventos en el cine-teatro (ver fragmento 11). Lo que hacen las funciones es usar los elementos de la llave “id” para crear un tipo date y poder realizar las comparaciones necesarias.

5.4. “Intellisense”

Esta carpeta contiene la otra parte del backend de la aplicación, la que la hace “inteligente”. El archivo “Funcs.py” contiene tres funciones que permiten recomendar un nuevo horario para organizar un evento, las cuales son llamadas cuando no es posible planificar un evento en el horario deseado originalmente. Existen dos funciones para recomendar un nuevo horario en el día y otra función que busca días con pocos eventos.

- “FindNewHour-Film-Theather(eps: list, d: date, InitH: time, EndH: time, tip: str, id: int, emplsNed: dict)” : esta función es llamada cuando no es posible programar una proyección fílmica o una obra teatral en el horario escogido inicialmente. Esta función realiza dos recorridos: un bucle le permite recorrer los horarios anteriores a la hora inicial escogida, revisando que durante la duración escogida para el evento no existan choques de recursos con otros eventos, para ello llama a las funciones del paquete “Functions” que le permiten realizar esa validación; el otro bucle le permite hacer ese mismo recorrido, pero una hora después de la inicial y hasta las 0:00. Si no encuentra un horario disponible en el día, llama a la función “FindNewDay(day: date)”.
- “FindNewHour-Music(eps: list, d: date, InitH: time, EndH: time)” : esta función es llamada cuando no es posible programar un concierto musical en el horario escogido

inicialmente. Realiza los mismos recorridos que la función anterior, pero solo revisa que no haya ningún evento en cada período de tiempo. Si no encuentra un horario disponible en el día, llama a la función “FindNewDay(day: date)”.

- “FindNewDay(day: date)”: esta función busca en el período de tiempo posterior (30 días) al día actual, aquellos días que poseen pocos eventos (los 3 primeros que encuentra son los que devuelve). Si no es capaz de encontrar ningún día suficientemente vacío, entonces lanza un mensaje que indica esperar al día siguiente (para extender el rango de búsqueda que es de 30 días).

5.5. “Data”

Esta carpeta contiene al archivo .json que posibilita la permanencia de los datos de la aplicación. El archivo se organiza como un enorme diccionario con dos llaves:

- “Eventos”: esta llave tiene asociada como valor una lista con todas las jornadas del cine-teatro, las cuales tienen la estructura presentada en el fragmento 11.

```
1  {
2      "Eventos": [
3          {
4              "id": [
5                  2026,
6                  2,
7                  5
8              ],
9          > "Lista_Eventos": [...
80         ],
81         "In_Time": true
82     }
83 ],
```

Figura 5: Fragmento del .json donde se guardan los eventos

- “Recursos”: esta llave tiene asociado como valor un diccionario que contiene dos llaves: “humanos”, la cual contiene a cada grupo de trabajadores con su respectiva cantidad, y “salas”, la cual contiene varios diccionarios que representan las salas del cine, estos cuentan con una llave “id” que indica el número de la sala, otra llave que muestra los “recursos asociados” o equipos instalados en la sala y una tercera llave que presenta las cantidades de trabajadores necesarias en la sala.

```

84     "Recursos": {
85       "humanos": {
86         "personal de limpieza": 8,
87         "tecnicos de sonido": 6,
88         "tecnicos de iluminacion": 6,
89         "operadores de proyeccion": 6,
90         "personal de seguridad": 8
91       },
92       "salas": [
93         {
94           "id": 1,
95           "capacidad": 150,
96           "recursos_asociados": [
97             "pantalla",
98             "proyector",
99             "sistema de sonido",
100            "sistema de iluminacion"
101          ],
102           "necesita": [
103             "1 tecnico de sonido",
104             "1 operador de proyector",
105             "1 tecnico de iluminacion"
106           ]
107         },

```

Figura 6: Fragmento del .json donde se guardan los recursos

6. Resultados

El resultado obtenido ha dejado sensaciones satisfactorias. A continuación se muestran imágenes de la aplicación en funcionamiento.

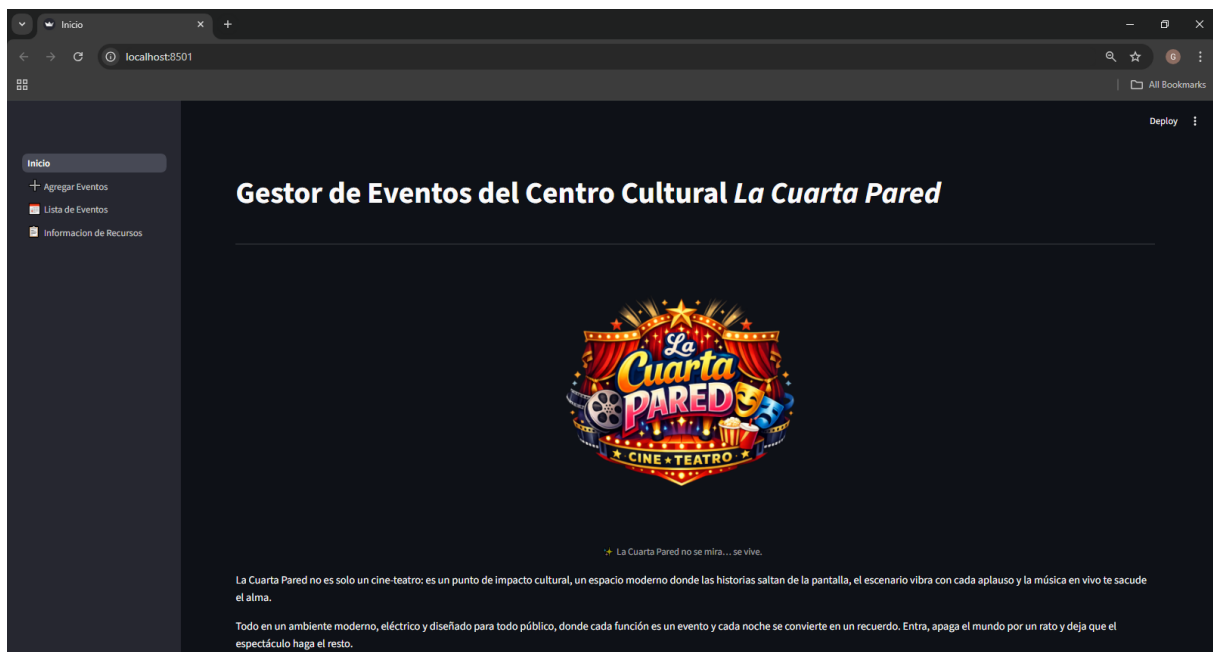


Figura 7: Página de Inicio

Inicio

- Agregar Eventos
- Lista de Eventos
- Información de Recursos

Agregar nuevos eventos

Seleccione el evento que desea agregar:

Proyeccion Filmica

Seleccione la fecha del nuevo evento:

2026/02/08

Ingrese la hora del nuevo evento:

Horas: 10 Minutos: 30

Ingrese la duracion del evento:

Horas: 1 Minutos: 15

Ingrese la cantidad de personas que asistirán al evento:

85

Ingrese la sala donde desea realizar el evento:

1

Figura 8: Página para Agregar Eventos

Inicio

- Agregar Eventos
- Lista de Eventos
- Información de Recursos

Eventos Programados

Seleccione el día del cual desea ver los eventos programados:

Todos los dias

February, 05, 2026

- 08:00-09:01 | 'Proyeccion Filmica: Pelicula 1' [Ver Detalles](#) [Eliminar](#)
- 14:00-15:01 | 'Obra de Teatro: Obra 1' [Ver Detalles](#) [Eliminar](#)
- 15:40-16:41 | 'Concierto Musical: Concierto 1' [Ver Detalles](#) [Eliminar](#)

Figura 9: Página para ver Lista de Eventos



Figura 10: Página para ver Información de Recursos

7. Conclusiones

El presente proyecto ha resultado una excelente experiencia de aprendizaje. He podido familiarizarme con un nuevo framework, así como conocer nuevas bibliotecas del lenguaje Python, que han sido útiles en el desarrollo de la aplicación. Además, adquirí valiosas habilidades en la modularización y organización del código y la aplicación de los contenidos obtenidos en las conferencias para dar solución a todas las situaciones que fueron surgiendo durante el desarrollo de la aplicación.

8. Repositorio

Enlace al repositorio del proyecto:

<https://github.com/GuillermoRR06/Management-System--CC->