ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Diseño de Software

Tarea2-Travel Ease

INTEGRANTES:

Perdomo Ordoñez Paul Isaac

Herrera Nieto Christian Alexander

Muñoz Sanchez Salvador Gabriel

Rosado Alcivar Enrique Gabriel

Grupo:

7

Profesora:

Jurado Mosquera David Alonso

Periodo Académico:

PAO II – 2024

Tabla de contenido

Sección A

1. Builder:

- Para crear instancias de reservas con vuelos, vehículos y servicios adicionales (e.g., seguros).
- Separa la lógica de creación de objetos del resto del sistema.
- Hace que la integración de nuevos tipos sea más sencilla.

2. Observer:

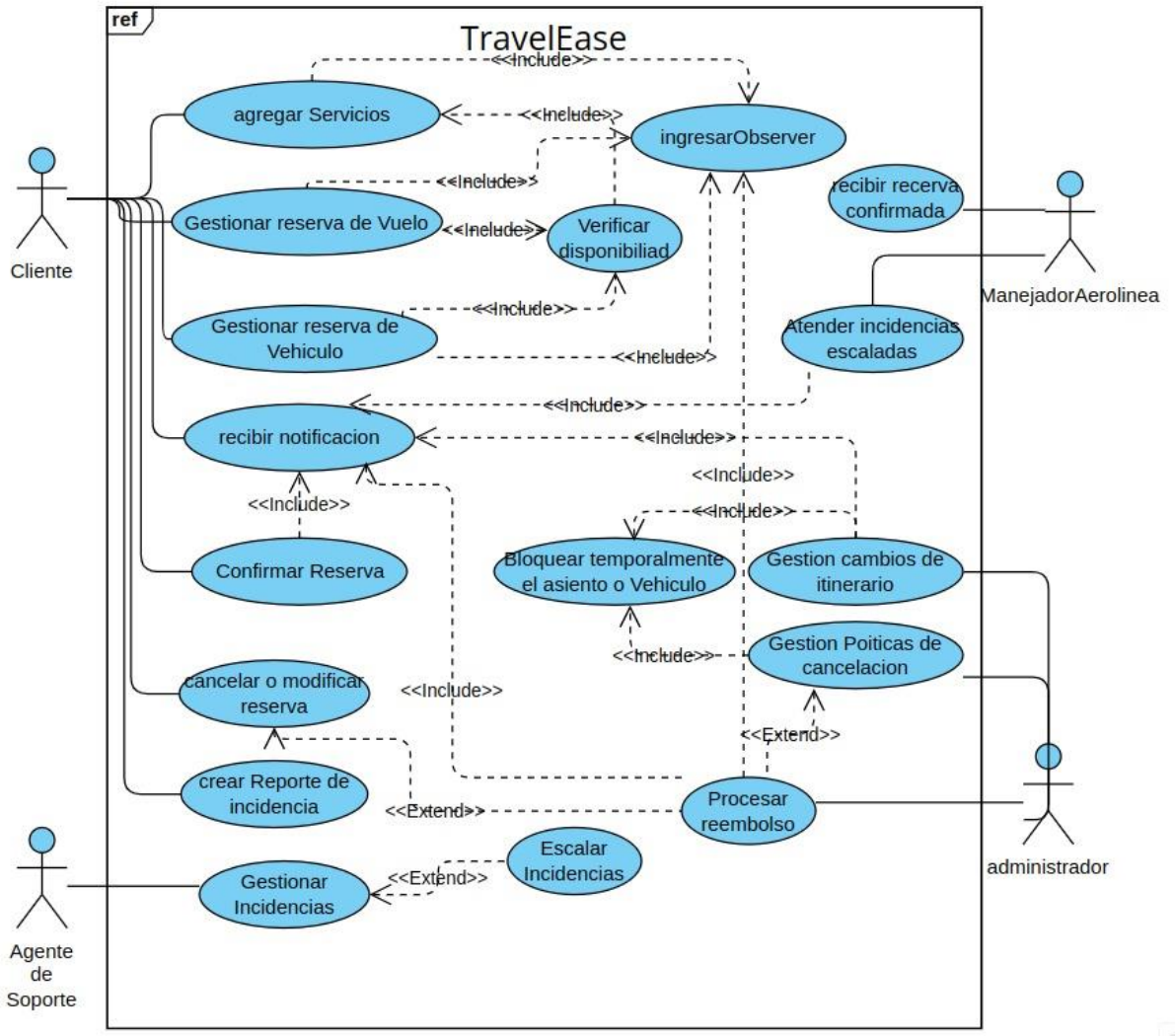
- Para notificaciones a los usuarios sobre el estado de sus reservas.
- Permite la actualización automática de los usuarios cuando hay cambios en vuelos o vehículos.
- Reduce la dependencia entre módulos emisores (vuelos, vehículos) y receptores (usuarios).

3. Chain of responsability:

- Permite manejar estas incidencias mediante una cadena de manejadores que procesan o delegan la responsabilidad según sus capacidades.
- Se evita un diseño rígido en el que el cliente (sistema principal) necesite conocer cómo o quién resolverá la incidencia.
- Nuevos niveles de soporte, como equipos especializados o administradores superiores, pueden añadirse a la cadena sin modificar las clases existentes.

Sección B

1. Diagrama de Caso de Uso



2. Caso de Uso #1

Caso de Uso	
ID	**CDU-A01**
Titulo	Realizar Reserva integrada de vuelo y vehículo
Autor	Paúl Perdomo
Actor principal	Cliente
Actor secundario	Sistema de notificaciones, Gestor de incidencias
Incluye a	Notificar usuario
Extiende de	Agregar servicios adicionales
Descripción	Describe el flujo de acciones que realiza el usuario al reservar un vuelo con un vehículo
Precondición	El usuario inicia sesión en el sistema
Secuencia normal	1. El usuario selecciona el destino, fecha y preferencias de vuelo. 2. El usuario selecciona el tipo de asiento y la aerolínea. 3. El sistema usa el patrón **Builder** para crear una reserva provisional con los datos seleccionados. 4. El sistema verifica la disponibilidad de vuelos y vehículos. 5. El sistema envía notificaciones mediante Observer. 6. El usuario ingresa los datos de pago y confirma la reserva. 7. El sistema confirma la reserva y actualiza a los interesados usando Observer.
Secuencia alternativa	En el paso 4 si no está disponible el asiento o el vehículo se le presentan alternativas al usuario En el paso 5, si el usuario desea puede agregar servicios adicionales En el paso 6 si el método de pago falla, desbloquea los asientos en los vuelos y los vehículos en el destino
Postcondición	El vuelo y el vehículo quedan en estado "confirmado”, y las notificaciones se envían a los usuarios interesados.

3. Caso de uso #2

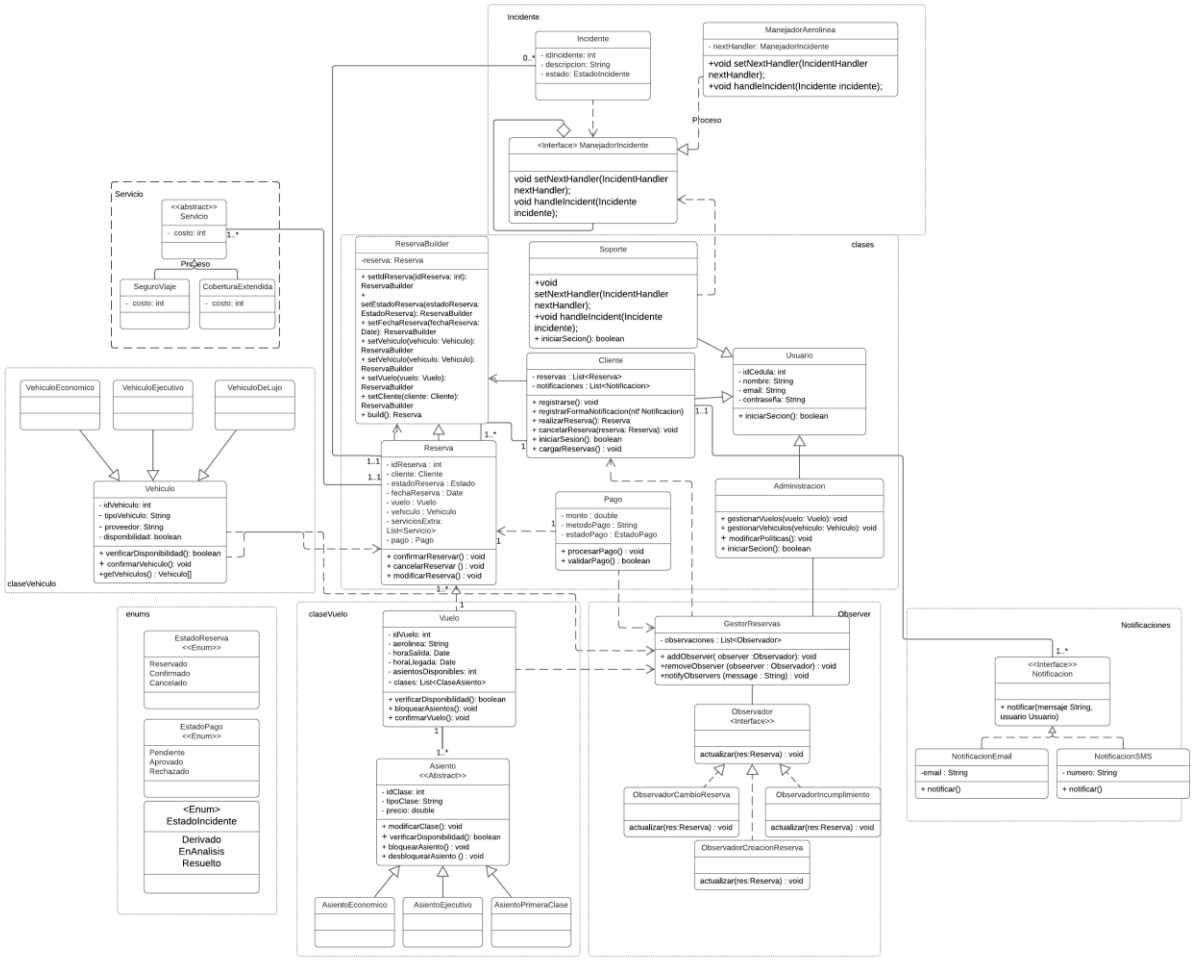
Caso de Uso	
ID	**CDU-A02**
Titulo	Cancelar reserva
Autor	Paúl Perdomo
Actor principal	Cliente
Actor secundario	Administrador, Sistema de Notificaciones
Incluye a	• Notificar usuario • Manejar Incidencias
Extiende de	
Descripción	Este caso de uso describe el proceso de cancelación de una reserva por parte de un cliente. El sistema verifica las políticas de cancelación, actualiza el estado de la reserva y notifica al cliente sobre el resultado. Si la cancelación no es posible, se informa al usuario con un mensaje de error.
Precondición	Debe existir una reserva en estado "reservado" o "confirmado".
Secuencia normal	1. El usuario solicita la cancelación de la reserva. 2. Selecciona una reserva desde el sistema. 3. El sistema cancela la reserva automáticamente. 4. Mediante el patrón *Observer*, el sistema notifica al usuario sobre el estado de la cancelación.
Secuencia alternativa	.
Postcondición	• El vuelo y el vehículo quedan en estado "cancelado”. • El usuario recibe una notificación automática sobre el estado final de la cancelación mediante Observer

4. Caso de uso #3

Caso de Uso	
ID	**CDU-A03**
Titulo	Reportar incidente
Autor	Paúl Perdomo
Actor principal	Cliente
Actor secundario	Agente de soporte
Incluye a	• Notificar usuario • Manejar Escalamiento de Incidencias
Extiende de	
Descripción	Describe el flujo de acciones que realiza el usuario para reportar problemas del sistema en la plataforma. El agente de soporte se encarga de resolver los problemas y, en caso de no resolverse, son escalados al departamento de servicio al cliente de la aerolínea o la empresa de alquiler.
Precondición	• Debe existir una reserva en estado "confirmado" o “reservado”. • El cliente debe identificar un problema o incidente relacionado con la reserva.
Secuencia normal	1. El usuario reporta una incidencia (ej. problema con el horario o el vehículo). 2. El sistema utiliza el patrón Builder para estructurar los detalles del reporte (tipo de incidencia, descripción, datos de la reserva). 3. El reporte se asigna al agente de soporte correspondiente. 4. El agente revisa la incidencia y, si es posible, la resuelve. 5. El sistema usa Observer para notificar al usuario sobre la resolución de su problema.
Secuencia alternativa	En el paso 4 si el agente no puede resolver la incidencia, el sistema activa el Chain of Responsibility, escalando el problema al departamento de servicio al cliente de la aerolínea o la empresa de alquiler.
Postcondición	• Si no fue resuelto, el reporte se escala al departamento correspondiente. • El usuario recibe notificaciones automáticas sobre cada etapa del proceso utilizando el patrón Observer.

Sección C

1. Diagrama de Clase



2. Patrones

- **Builder**:
  La clase ReservaBuilder implementa el patrón Builder para construir objetos de tipo Reserva de forma modular.

- **Observer**:
  El patrón Observer está representado por las clases GestorReservas (sujeto) y las clases ObservadorCambioReserva y ObservadorIncumplimiento (observadores), permitiendo notificar cambios de estado.

- **Chain of Responsibility**:

Implementado por las clases relacionadas con Incidente y ManejadorIncidente, facilitando la delegación del manejo de incidentes a diferentes manejadores.

3. Principios SOLID

- **Responsabilidad Única (SRP)**:
  Cada clase tiene una función específica. Ejemplo: ReservaBuilder solo se encarga de construir reservas.

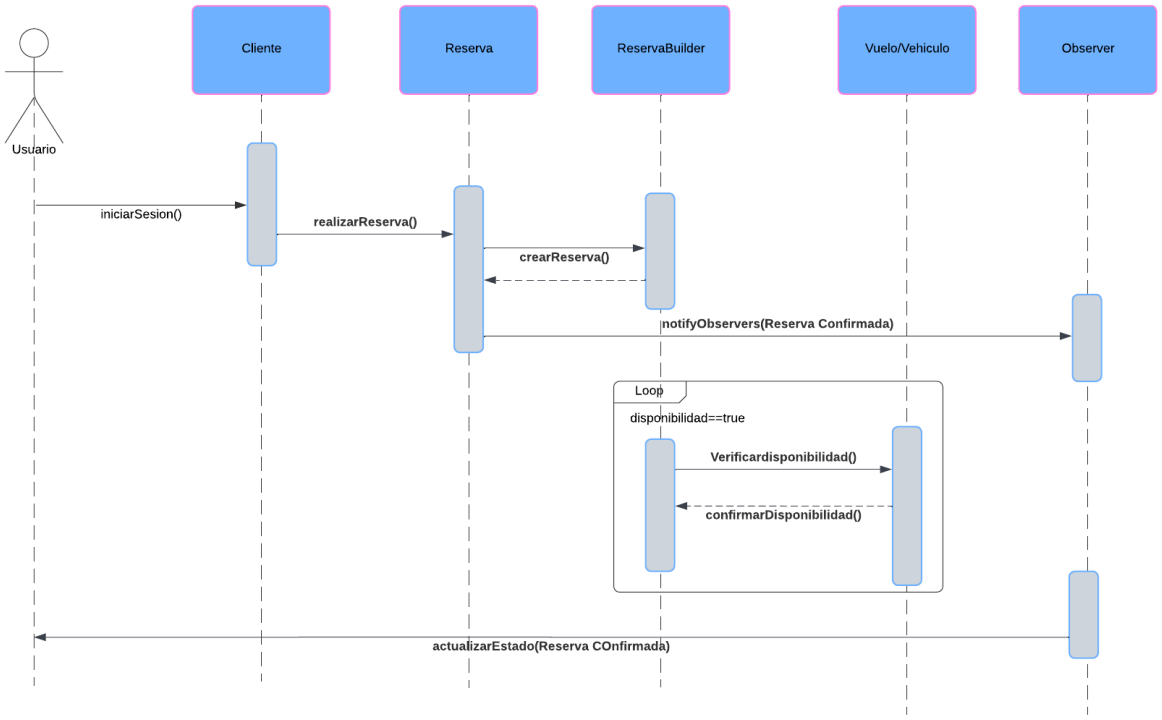
- **Abierto/Cerrado (OCP)**:
  Las clases como Vehículo pueden extenderse (VehículoEconómico, VehículoEjecutivo, etc.) sin modificar su estructura base.

- **Sustitución de Liskov (LSP)**:
  Las subclases de Vehículo son intercambiables donde se utiliza su clase base.

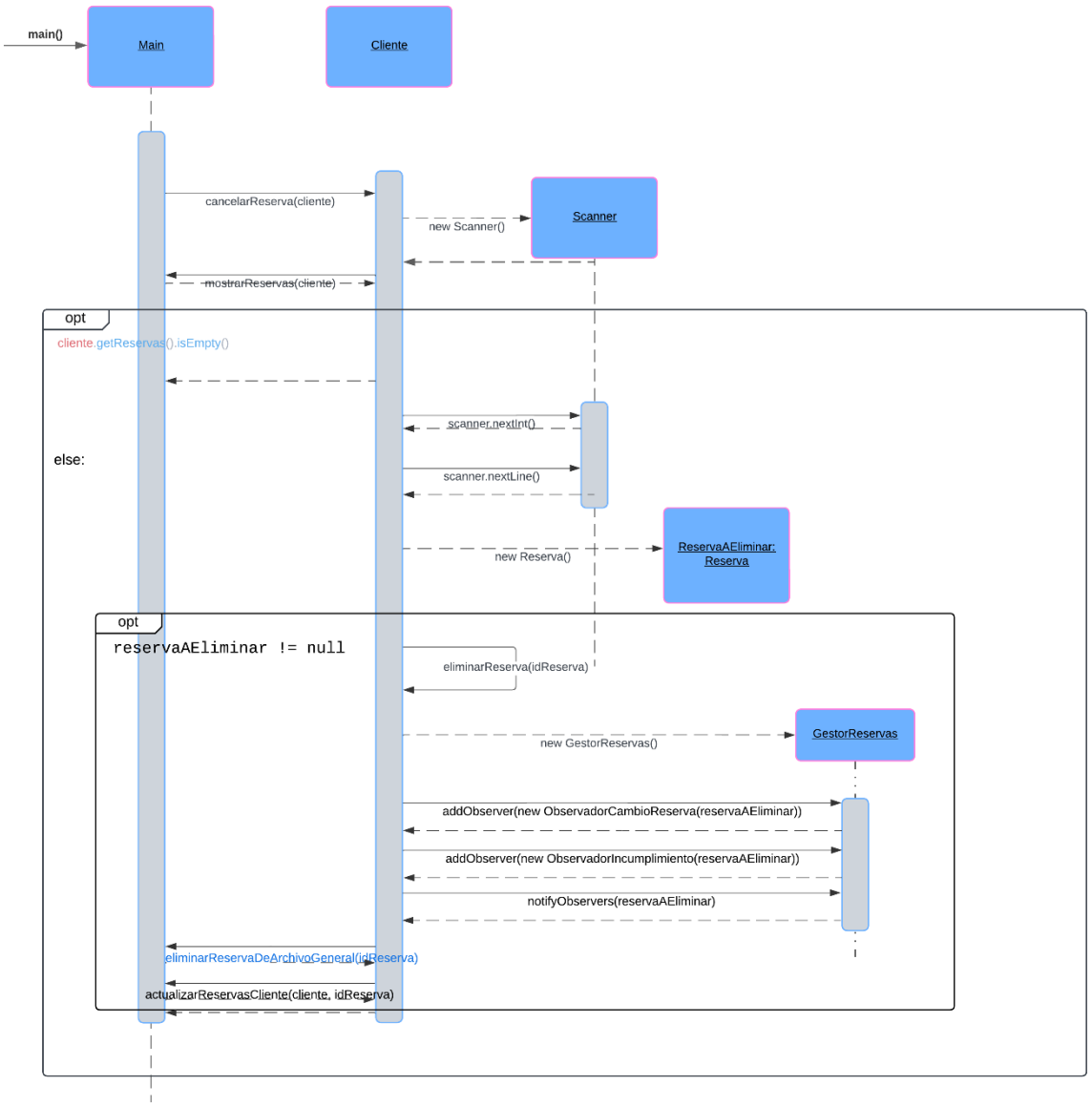
- **Inversión de Dependencias (DIP)**:
  Las clases dependen de abstracciones (Observer, ManejadorIncidente) en lugar de implementaciones concretas.

Sección D

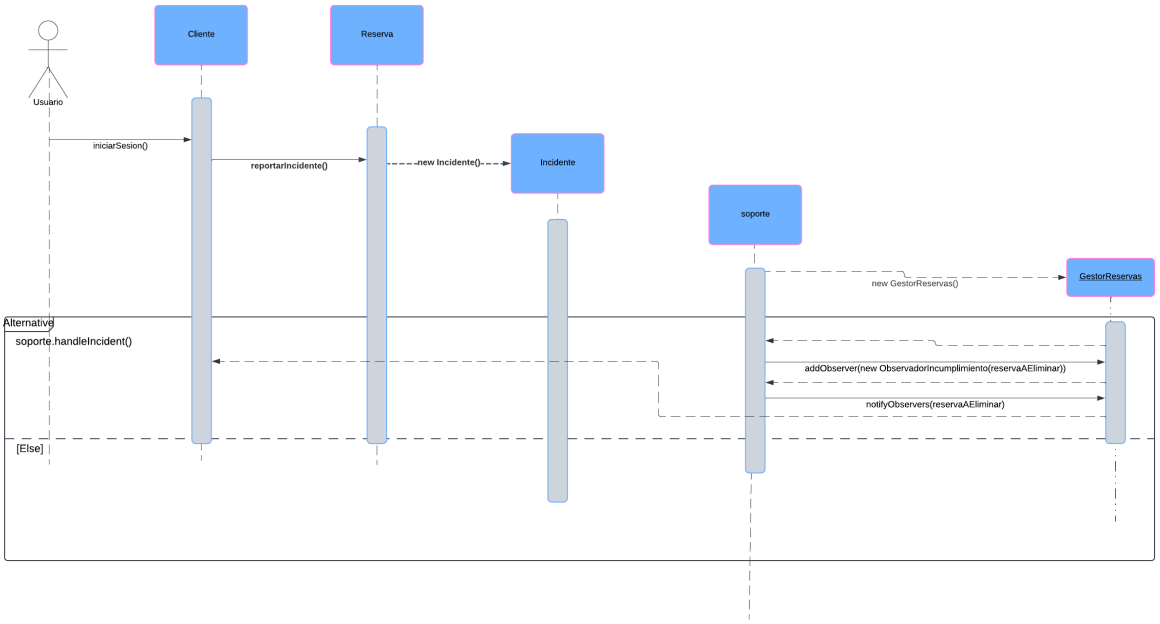
1. Diagrama De Secuencia #1



2. Diagrama De Secuencia #2



3. Diagrama De Secuencia #3



Sección E

Github: https://github.com/Herrera2005/Tarea2Patrones.git

Diagrama de uso: https://online.visual-paradigm.com/w/cpshlpme/diagrams/#diagram:workspace=cpshlpme&proj=2&id=20&type=UseCaseDiagram&width=11&height=8.5&unit=inch

Diagrama de clase: https://lucid.app/lucidchart/2297f08f-6e79-4053-8c04-b162c390b93c/edit?viewport_loc=-1564%2C-826%2C4402%2C2054%2C0_0&invitationId=inv_8c844a89-1e5f-47b2-9fc5-ba0c1aa6bf89

Diagramas de secuencia #1: https://lucid.app/lucidchart/23bf59fd-7f03-4c64-ad0d-a36deda0d48a/edit?viewport_loc=-712%2C-241%2C2215%2C1033%2C0_0&invitationId=inv_cbb1153f-0f52-4220-9c30-f020e77a0cf8

Diagramas de secuencia #2: https://lucid.app/lucidchart/fc6dba49-937d-41b0-a933-9dbdeb75f294/edit?view_items=K1UyUkpB2mjI&invitationId=inv_6b213f8e-b79a-4afc-8279-4748327b55f0

Diagramas de secuencia #3: https://lucid.app/lucidchart/f2eb6405-ab52-4634-8d67-4473dfac53df/edit?view_items=wYUyyqaKfF9h&invitationId=inv_1608339e-91f7-4f74-b21f-5a15ecdb82cf