# Seminario Integrador
# Travelix

Daniel Toledo Daniel Machado Osvaldo Moreno José Antonio Concepción
C-311 - Agencia de Viajes. Equipo 3 Semigrupo 2.

15 de marzo de 2024

# Índice

# 1. Introducción

## 1.1. Alcance del Producto

El alcance del producto abarca la creación de una base de datos y una aplicación para la gestión de varias agencias de viajes. La aplicación permitirá a los usuarios realizar reservas de excursiones y paquetes en determinadas agencias, mientras que la base de datos almacenará información sobre dichas agencias, hoteles, excursiones, turistas y paquetes. La aplicación también podrá responder a preguntas sobre las reservas, los hoteles y las excursiones. Además tendrá un módulo para consultar estadísticas y exportarlas.

## 1.2. Descripción general del producto

**Perspectiva y funciones del producto:**

La aplicación web de la agencia de viajes es una herramienta que permite a los usuarios buscar, comparar y reservar viajes de forma online. La aplicación ofrece varios servicios, incluyendo:

- Reserva de paquetes turísticos.
- Información turística sobre los diferentes destinos.
- Consulta de estadísticas.

En términos de las características de los usuario, la aplicación web de la agencia de viajes está dirigida principalmente a Agencias de viajes que buscan una herramienta para reservar servicios para sus clientes. Estos a su vez se clasifican como turistas individuales, que planifican sus propios viajes y buscan la mejor oferta. En adición se pudiera contemplar su uso por grupos de amigos, compañeros de trabajo o núcleos familiares que planean un viaje juntos.

La aplicación web de la agencia de viajes tiene algunas restricciones. La disponibilidad de los servicios es una de ellas ya que depende de la fecha de viaje, el destino y capacidad. Los precios de los servicios pueden variar según la fecha de viaje, el destino y la disponibilidad, además que estos son dependientes de una agencia y sus ofertas específicas.

## 1.3. Resumen del resto del documento

- En el **Capítulo II**, **Requerimientos Específicos**, se detallan los requisitos funcionales y no funcionales del producto. Esto incluye una lista concisa de las funciones que debe realizar la aplicación según los deseos del cliente, así como los elementos restrictivos que no están directamente relacionados con la funcionalidad, como la usabilidad, la seguridad y el diseño e implementación.
- El **Capítulo III**, **Funcionalidades del Producto**, describe las funcionalidades de la aplicación utilizando un diagrama de casos de uso del sistema. Este capítulo proporciona una visión clara de cómo interactúan los usuarios con la aplicación y qué capacidades ofrece.
- El **Capítulo IV**, **Enfoque Metodológico**, justifica el enfoque utilizado en el desarrollo de la aplicación, teniendo en cuenta las características del problema y del personal involucrado en el proyecto. Esto puede incluir detalles sobre la metodología de desarrollo de software utilizada y cómo se adaptó a las necesidades específicas del proyecto.

- En el **Capítulo V**, **Arquitectura**, se presenta y justifica la propuesta de arquitectura de la aplicación, acompañada de un diagrama que ilustra la estructura general del sistema. Este capítulo aborda las decisiones fundamentales de diseño que sustentan la construcción del software.
- El **Capítulo VI**, **Patrones de visualización y de datos**, argumenta qué patrones de diseño se utilizarán en la interfaz de usuario y en la manipulación de datos, y cómo se aplicarán a la solución desarrollada. Esto puede incluir patrones de diseño de interfaz de usuario y patrones de acceso a datos.
- Finalmente, el **Capítulo VII**, **Modelo de datos**, presenta el modelo que determina la estructura de la base de datos del proyecto. Este capítulo es fundamental para comprender cómo se organiza y almacena la información dentro de la aplicación.

En conjunto, estos capítulos proporcionan una visión integral de la aplicación desarrollada, desde sus requisitos iniciales hasta su arquitectura y modelo de datos, ofreciendo una comprensión completa de su diseño y funcionamiento.

# 2. Requerimientos Específicos

## 2.1. Requerimientos Funcionales

- **Obtención del Listado de Hoteles en Paquetes:** La aplicación deberá proporcionar una funcionalidad que permita a los usuarios obtener de manera eficiente el listado completo de hoteles que están incluidos en los paquetes de viajes ofrecidos por las agencias. Esta información es crucial para los turistas que desean conocer las opciones de alojamiento disponibles en los paquetes turísticos.
- **Cálculo de Reservaciones y Monto Total:** La aplicación debe ser capaz de realizar un seguimiento preciso de todas las reservaciones realizadas en las agencias de viajes. Deberá calcular automáticamente la cantidad total de reservaciones realizadas y el monto económico total asociado. Esto proporcionará a las agencias información valiosa sobre la demanda de sus servicios y permitirá una mejor planificación y gestión financiera.
- **Obtención de Datos de Turistas Frecuentes:** La aplicación deberá identificar y presentar los nombres y direcciones electrónicas de aquellos turistas que han realizado reservaciones individuales en más de una ocasión. Esto facilitará a las agencias de viajes establecer relaciones más personalizadas con estos clientes frecuentes, ofreciendo servicios adaptados a sus preferencias.
- **Obtención de Horarios y Lugares de Salida de Excursiones en Fines de Semana Extendidos:** La aplicación debe proporcionar una funcionalidad que permita obtener la información detallada sobre los lugares y horarios de salida de todas las excursiones ofrecidas durante los fines de semana extendidos (viernes, sábado y domingo). Esta información, ordenada de manera apropiada, será presentada en la interfaz de la agencia para su consulta y visualización.
- **Análisis de Paquetes con Precio Superior al Promedio:** La aplicación debe ser capaz de analizar y calcular la cantidad de paquetes cuyo precio está por encima del precio promedio de todos los paquetes disponibles. Este análisis permitirá a las agencias identificar y destacar paquetes de mayor valor, así como ajustar estrategias de precios.

- **Reservación Apropiada para Turista Individual:** La aplicación deberá proporcionar una funcionalidad eficiente para realizar reservaciones que correspondan de manera apropiada a un turista individual. Esto implica seleccionar la compañía aérea, gestionar la reserva en los hoteles deseados, establecer fechas de llegada y salida, y calcular el precio total de manera precisa.
- **Mostrar e Imprimir Estadísticas:** La aplicación deberá incorporar una funcionalidad que permita a los usuarios visualizar y imprimir estadísticas generadas a partir de consultas específicas. Estas estadísticas representan resúmenes y análisis de datos relevantes, brindando a los usuarios una perspectiva integral del rendimiento y comportamiento de las entidades gestionadas por la aplicación.
- **Aspectos generales del sistema:** El sistema debe ser "tipo web, para que cualquier persona pueda acceder a los servicios que se brindan y los trabajadores también tengan una interacción sencilla".
- **Posibles aspectos futuros:** Dadas las interacciones entre cliente y operador de ventas, es posible que un futuro se quiera registrar el operador que tramita la venta para obtener bonos específicos por cantidad de ventas.

## 2.2. Requerimientos no funcionales

Se consideran elementos restrictivos generados por el cliente, no directamente relacionados con la funcionalidad del producto.

- Eficiencia en el Rendimiento: La aplicación debe mantener un rendimiento eficiente incluso cuando maneje grandes cantidades de datos, garantizando tiempos de respuesta rápidos y una experiencia de usuario fluida.
- Interfaz de Usuario Intuitiva: La interfaz de usuario debe ser intuitiva y fácil de usar para el personal del departamento de marketing y los agentes de venta, sin requerir un extenso entrenamiento.
- Seguridad de Datos: La seguridad de la información, especialmente los datos personales de los turistas, debe ser una prioridad. La aplicación debe implementar medidas robustas de seguridad para proteger la privacidad y confidencialidad de la información del cliente. Debe existir un proceso de autenticación básico para cada usuario. Además se debe crear una jerarquía de permisos para diferenciar administradores de turistas.
- Módulo de Administración: Se incluirá un módulo de administración dedicado a gestionar todas las entidades del modelo de la aplicación. Este módulo facilitará la población inicial y la gestión continua de la base de datos, permitiendo a los administradores realizar operaciones como la creación, modificación y eliminación de registros de hoteles, excursiones, turistas, paquetes, etc.
- Diseño Responsive: La web se diseñará para ser responsive, adaptándose de manera óptima a diferentes tamaños de pantalla y dispositivos. La interfaz de usuario deberá garantizar una experiencia de usuario consistente y amigable, independientemente del dispositivo desde el cual se acceda.

### 2.2.1. Usabilidad

Nuestra aplicación web estará diseñada para ofrecer una experiencia de usuario intuitiva y accesible para todos, independientemente de su nivel de experiencia. Para ello, implementaremos una interfaz sencilla y clara, en la se utilizará un diseño minimalista con una organización lógica de los elementos. Además, se implementará un sistema de navegación consistente y fácil de comprender, para que resulte intuitiva para cualquier usuario.

### 2.2.2. Seguridad

La seguridad en el proyecto es clave para el éxito del mismo, ya que protege la integridad, confidencialidad y disponibilidad de la información, previniendo daños y asegurando la confianza de los usuarios. Implementar medidas de seguridad adecuadas permite minimizar riesgos, garantizar la continuidad del negocio y cumplir con las regulaciones vigentes. Para ello nos basaremos en las siguientes estrategias.

**Confidencialidad**

La confidencialidad es un pilar fundamental de nuestra aplicación. La información de los usuarios, las reservaciones de paquetes están protegidas por un sistema de seguridad integral.

Para acceder a las funciones de reservación, los usuarios deben iniciar sesión con sus credenciales. Esto garantiza que solo los usuarios autenticados puedan realizar transacciones.

Además, solo las agencias autorizadas tienen la capacidad de publicar información sobre ofertas y paquetes. Cada agencia tiene acceso exclusivo a su propia información o datos de interés de sus clientes, lo que evita modificaciones no autorizadas.

**Integridad**

La información manejada por el sistema estará protegida contra la corrupción y estados inconsistentes. Para ello se implementarán la validación de datos dónde se verificará la exactitud y consistencia de los datos antes de ser almacenados en el sistema. Con los controles de acceso, quedará restringido el acceso a la información a usuarios autorizados para evitar modificaciones fuera de lugar. Finalmente mediante copias de seguridad de la información de forma regular podremos evitar la pérdida de datos en caso de fallos o errores. Estas medidas contribuyen a la integridad del sistema.

**Disponibilidad**

Significa que a los usuarios autorizados se le garantizará el acceso a la información y que los dispositivos o mecanismos utilizados para lograr la seguridad no ocultarán o retrasarán a los usuarios para obtener los datos deseados en un momento dado.

Por otra parte no se comparten los datos personales del usuario. Las cuentas son únicas y se encuentran protegidas por una contraseña, que no se almacena directamente en la Base de Datos. En cambio se almacena su hash respaldado por un algoritmo de generación y un keyword a los que solo tiene acceso el programador.

## 2.3. Diseño e implementación

- **React para el frontend:** React es una de las bibliotecas de JavaScript más populares para el desarrollo de interfaces de usuario. Es conocida por su rendimiento, su capacidad para construir componentes reutilizables y su enfoque basado en componentes, que facilita la creación de interfaces de usuario dinámicas e interactivas. Además, cuenta con una amplia comunidad de desarrolladores y una gran cantidad de recursos disponibles, lo que simplifica el proceso de desarrollo y resolución de problemas.

- **FastAPI para el backend:** FastAPI es un framework web rápido (de ahí su nombre) y fácil de usar para crear APIs en Python. Se destaca por su velocidad, gracias a la implementación basada en Starlette y Pydantic, y su facilidad de uso debido a su sintaxis intuitiva y autocompletado. Además, proporciona una documentación automática y precisa de la API, lo que facilita su comprensión y uso por parte de otros desarrolladores.

- **PostgreSQL para gestionar la base de datos:** PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto y altamente confiable. Ofrece una amplia gama de características avanzadas, como soporte para consultas complejas, integridad de datos, transacciones ACID y capacidades de escalabilidad. Además, es conocido por su robustez, estabilidad y comunidad activa de usuarios y desarrolladores.

- **VSCode como IDE:** Visual Studio Code (VSCode) es un entorno de desarrollo integrado liviano y altamente personalizable. Es popular entre los desarrolladores debido a su amplia gama de extensiones disponibles, su integración con Git y su soporte para una variedad de lenguajes de programación. Además, su interfaz de usuario intuitiva y su rendimiento rápido lo convierten en una opción sólida para el desarrollo de software.
- **ORM SQLAlchemy:** SQLAlchemy es una biblioteca de mapeo objeto-relacional (ORM) para Python que facilita la interacción con bases de datos relacionales mediante objetos Python. Proporciona una capa de abstracción sobre la base de datos subyacente, lo que simplifica la escritura y lectura de consultas SQL y facilita la manipulación de datos en la aplicación. Además, es altamente flexible y admite una variedad de bases de datos, incluida PostgreSQL.
- **Arquitectura N-capas:** La arquitectura N-capas es un enfoque común para organizar el código de una aplicación en capas lógicas o niveles de abstracción, como la presentación, la lógica de negocio y el acceso a datos. Proporciona modularidad, flexibilidad y escalabilidad al separar las responsabilidades y facilitar la modificación y mantenimiento del código.
- **Bibliotecas de seguridad de Python y de FastAPI:** La seguridad es una preocupación fundamental en cualquier aplicación web. Utilizar bibliotecas de seguridad específicas para Python y FastAPI permite implementar prácticas recomendadas de seguridad, como autenticación, autorización, encriptación de datos y protección contra ataques comunes, lo que garantiza la protección de la aplicación y los datos de los usuarios.

### 2.4. Requerimientos de Entorno

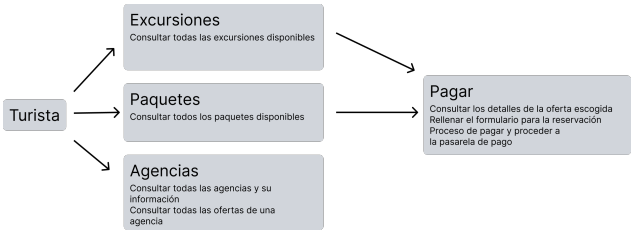
Compatibilidad con Sistemas Operativos: La aplicación debe ser compatible con una variedad de sistemas operativos, incluyendo Windows, macOS y Linux, para garantizar su accesibilidad a una amplia audiencia de usuarios.

- Conexión a Internet: Para la actualización en tiempo real de ofertas y la realización de reservaciones, la aplicación debe requerir acceso a una conexión a Internet estable y segura.
- Escalabilidad: La aplicación debe ser diseñada de manera que sea escalable, capaz de manejar el crecimiento en la cantidad de agencias, turistas y servicios ofrecidos sin comprometer su rendimiento.
- Publicación: Se recomienda registrar un dominio en una compañía registradora y adquirir un plan de alojamiento web o hosting. Específicamente trabajar AWS por su confiabilidad y estabilidad demostrada en el tiempo.
- Tecnologías: El cliente no tiene conocimientos avanzados de informática por lo que le proponemos dadas sus peticiones y amplia capacidad adquisitiva:
  - Alquilar servidores para las bases de datos en la nube.
  - Utilizar React, FastAPI y PostgreSQL como tecnologías.

## 3. Funcionalidades del Producto

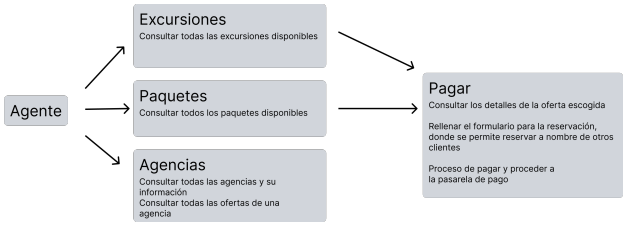
En nuestra aplicación hay cuatro tipos de actores: Turista, Agente, Marketing y Administrador.
A continuación se muestran los distintos casos de uso dependiendo del rol de cada usuario:

## Turista




El turista puede acceder a la información de todas las ofertas (paquetes y excursiones), en esta sección puede seleccionar la oferta que desee, y pasará a otra sección donde va llenar el formulario para la reservación y continuar el proceso de pago. En la sección de agencias puede consultar las ofertas de alguna agencia en específico.

## Agente

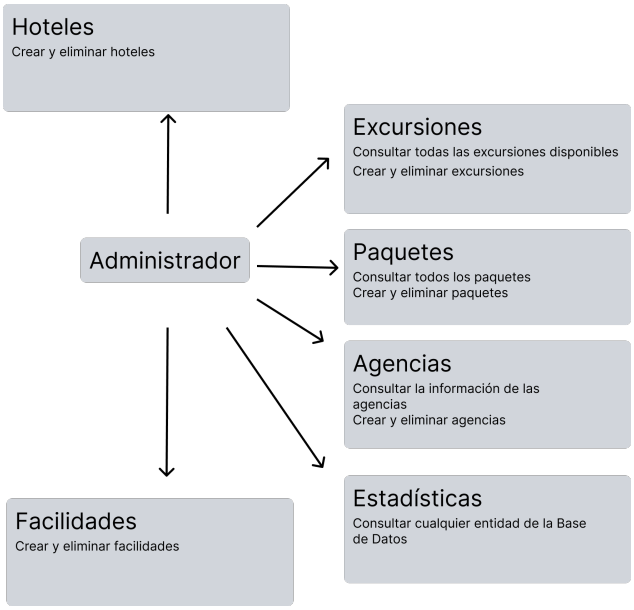



El agente es aquel que tiene la capacidad de realizar reservas en nombre de terceros. Esto significa que si una persona no puede acceder a la aplicación por cualquier motivo, el usuario de agente puede intervenir y facilitar el proceso de reserva para esa persona.

## Marketing


El usuario de marketing tiene varias secciones, en las cuales puede añadir nuevas excursiones y paquetes a su propia agencia, es decir es el encargado de incorporar las ofertas de su agencia a la aplicación. También tiene una sección de estadísticas donde puede consultar las informaciones relacionadas con sus ofertas.

## Administrador

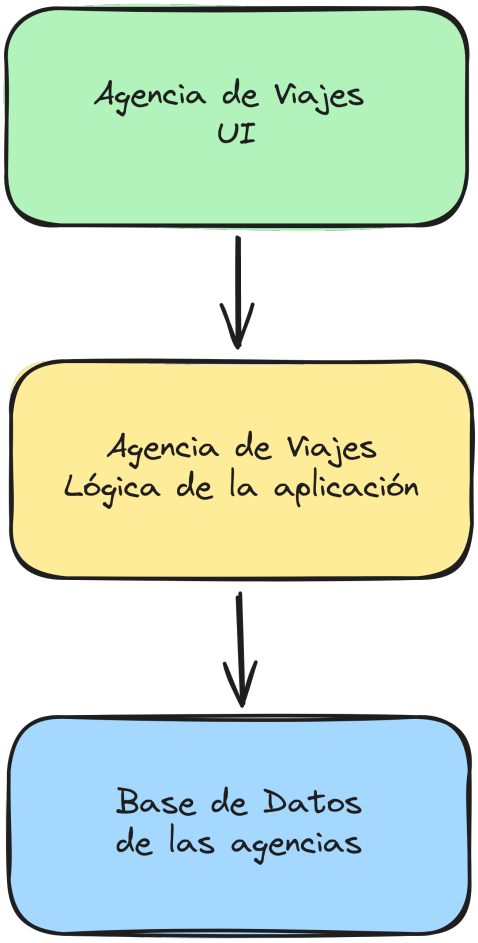


El administrador es el encargado de controlar los datos de la aplicación tiene una sección para cada entidad, donde podrá acceder a sus datos y manejarlos.

# 4. Enfoque Metodológico

La metodología a utilizar será el Desarrollo Esbelto de Software (DES) lo cual implica seguir principios clave como la eliminación del desperdicio, la generación de calidad, la creación de conocimiento, el aplazamiento del compromiso, la entrega rápida, el respeto a las personas y la optimización del proceso en su totalidad. Para ello, en el contexto de la agencia de viajes, se pueden tomar diversas acciones. En primer lugar, se eliminará el desperdicio mediante una cuidadosa evaluación de los requerimientos del cliente y la definición de tareas claras y concisas para evitar características innecesarias y extensos análisis previos. Asimismo, se establecerán ciclos de trabajo cortos para lograr entregas tempranas y continuas de versiones funcionales del software, permitiendo obtener retroalimentación rápida del cliente y facilitando ajustes ágiles. Para garantizar la generación de calidad, se implementará un proceso de testing temprano, con integrantes del equipo dedicados a esta tarea, lo que permitirá detectar errores de manera anticipada y mejorar continuamente la calidad del software. Además, se fomentará el conocimiento de todos los miembros del equipo mediante la colaboración, el intercambio de información y la realización de sesiones periódicas de retroalimentación y aprendizaje. Estas acciones, combinadas con la interacción constante con el cliente y la realización de demostraciones regulares, contribuirán a optimizar el proceso de desarrollo de software y a satisfacer las necesidades del cliente de manera efectiva.

## 5. Arquitectura




Para nuestro problema hemos decidido utilizar la arquitectura N-capas, basado en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división de los problemas a resolver.

En la capa de presentación (Agencia de viajes - UI) se muestra al usuario las diferentes opciones con las que puede interactuar, para ello estaremos utilizando como tecnología React.

En la capa de aplicación (Agencia de viajes - Lógica de aplicación) se estará realizando todo lo relacionado con el procesamiento de datos, como tecnología se estará utilizando FastAPI de python. En esta capa se realizan el procesamiento de información y las solicitudes a la capa de acceso a datos.

En la capa de acceso a datos (Base de datos de las agencias) se realizan las operaciones CRUD sobre los datos, para esto estaremos utilizando PostgreSQL y SQLAlchemy

La arquitectura de N-Capas es un enfoque de diseño de software que separa la lógica de negocio en capas distintas, lo que ofrece varias ventajas para una página web de una agencia de viajes que permite a los usuarios reservar excursiones y paquetes de excursiones. Algunas de las ventajas de utilizar la arquitectura N-Capas para este tipo de página web son:

**Separación de responsabilidades:** La arquitectura N-Capas permite separar claramente las diferentes responsabilidades del sistema, como la presentación de la información, la lógica de negocio y el acceso a datos. Esto facilita la gestión del código y su mantenimiento a lo largo del tiempo.

**Escalabilidad:** Al dividir la aplicación en capas independientes, es más fácil escalar cada una de ellas de forma independiente, lo que permite manejar un mayor número de usuarios y transacciones sin afectar el rendimiento general del sistema.

**Reutilización de código:** La arquitectura de N-Capas fomenta la reutilización del código en diferentes partes de la aplicación. Esto significa que funcionalidades comunes, como la gestión de reservas o el procesamiento de pagos, pueden implementarse una vez y ser utilizadas en varias partes del sistema.

**Mantenimiento más sencillo:** Al tener una separación clara entre las capas de la aplicación, el mantenimiento y la actualización de cada capa se simplifica. Esto permite realizar cambios en una capa sin afectar a las demás, lo que facilita la corrección de errores y la implementación de nuevas funcionalidades.

**Seguridad:** La arquitectura N-Capas facilita la implementación de medidas de seguridad en cada capa de la aplicación. Por ejemplo, se pueden establecer controles de acceso en la capa de presentación para proteger la información sensible de los usuarios.

# 6. Patrones de visualización y de datos

## 6.1. Patrón de visualización MVC

Para la capa de interfaz de usuario, se propone el patrón Modelo-Vista-Controlador (MVC). Este patrón separa la lógica del negocio de la forma en que se visualizan los datos.

El patrón MVC se compone de tres capas independientes:

- **Modelo**: Contiene todas las capas internas de la arquitectura del software, incluyendo el modelo de dominio, la interfaz de repositorio y la capa de lógica del negocio.
- **Vista**: Se encarga de mostrar la información disponible en el modelo. Permite que el modelo sea independiente de la forma en que se presenta la información.
- **Controlador**: Observa las acciones del usuario y decide cómo responder a ellas. En otras palabras, el controlador actúa como un intermediario entre el usuario y el modelo.

El patrón MVC divide la aplicación en tres capas con responsabilidades bien definidas, lo que facilita el desarrollo y mantenimiento del software. A su vez, la separación de capas permite modificar la interfaz de usuario o la lógica del negocio sin afectar a las demás capas, lo que permite que sea flexible. Este patrón además presenta una alta reusabilidad ya que los componentes de cada capa se pueden reutilizar en otras aplicaciones.

Pongamos el ejemplo de un usuario que se va a registrar en la aplicación.

Componente Modal Registrarse:

```
export const SignUp = () => {
  const initialState = {};

  const [state, dispatch] = useFormState<SignUpState, FormData>(
    CreateUserAction,
    initialState
  );
```

```
8
9   const searchParams = useSearchParams();
10  const register = searchParams.get('register');
11  const pathname = usePathname();
12
13  return (
14    register && (
15      <dialog
16        className={clsx(
17          'flex flex-col rounded-[32px]',
18          'bg-white dark:bg-extends-darker-blue-900',
19          'shadow-2xl px-8 pt-6 pb-12 w-[343px] sm:w-[480px] md:w-[520px]'
20        )}
21      >
22        <form
23          className='flex flex-col gap-4 md:gap-8 w-full *:w-full'
24          action={dispatch}
25        >
26        {... Resto del cuerpo del componente}
27        </form>
28      </dialog>
29    )
30  );
31 };
```

En las líneas 4-10 se define que acción se debe ejecutar cuando el usuario interactué con el componente. El componente por su parte solo se encarga de mostrar el formulario que debe llenar el usuario, y un botón que será el disparador de la acción que hayamos definido en el formulario (línea 24). Por otro lado en SignUpState tenemos el estado del fomulario, lo cual puede ser útil para visualizar los errores de validación que hayan ocurrido.

En este caso la acción a ejecutar es CreateUserAction:

```
1
2  export type SignUpState = {
3    message?: string;
4    errors?: {
5      username?: string[];
6      name?: string[];
7      nationality?: string[];
8      password?: string[];
9      confirmPassword?: string[];
10   };
11 };
12
13 const CreateUser = FormSchema.omit({ id: true, date: true });
14
15 export async function CreateUserAction(
16   prevState: SignUpState,
17   formData: FormData
18 ) {
19   const validatedFields = CreateUser.safeParse({
```

```
    username: formData.get('username') as string,
    name: formData.get('name') as string,
    nationality: formData.get('nationality') as string,
    password: formData.get('password') as string,
    confirmPassword: formData.get('confirmPassword') as string,
  });

  if (!validatedFields.success) {
    return {
      errors: validatedFields.error.flatten().fieldErrors,
      message: 'Invalid Fields. Failed to Create User.',
    };
  }

  const { username, name, nationality, password, confirmPassword } =
    validatedFields.data;

  if (password !== confirmPassword) {
    return {
      errors: {},
      message: 'Password does not match',
    };
  }

  // POST /tourist/create, http://127.0.0.1:8000/tourist/create

  const data = {
    username: username,
    name: name,
    nationality: nationality,
    password: password,
  };

  try {
    const resp = await fetch('http://127.0.0.1:8000/tourist/create', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(data),
    });
    if (!resp.ok) {
      return {
        errors: {},
        message: 'Error Response from Server. Failed to Create User.',
      };
    }
  } catch (error) {
    return {
      errors: {},
```

```
70        message: 'Database Connection error.',
71      };
72    }
73    redirect('?registerSuccess=true');
74  }
```

La acción primero valida los datos introducidos (línea 20-26), si fue incorrecta se retornara el estado de error, por otro lado si todo marcha bien, se realiza el intercambio con el modelo de datos (línea 53-72) en el cual añadirá el nuevo usuario creado. Si en algún momento el modelo de datos falla, entonces se retorna el error correspondiente. Una vez terminado con éxito se redirecciona para dar un mensaje de éxito al usuario y cerrar automáticamente el Componente de Registrarse.

## 6.2. Patrones de Acceso a Datos

Los patrones de Acceso a Datos de SQLAlchemy son: **Identity Map**, **Unit of Work**, **Data Mapper**, **Repository** y **Lazy Load**.

### Identity Map

**¿En qué consiste?:**
El patrón Identity Map asegura que una entidad se cargue solo una vez en la memoria del programa, independientemente de cuántas veces se haya solicitado desde diferentes partes del código.

**¿Para qué lo usa el ORM?:**
Evita cargar múltiples instancias de la misma entidad, manteniendo una única referencia para cada entidad en la memoria.

**¿Cómo lo usa el ORM?:**
SQLAlchemy implementa Identity Map automáticamente. Para esto usa el objeto Session, que es la interfaz para el uso del ORM, es decir la carga y persistencia de los datos. Una Session mantiene una referencia a las entidades mapeadas que están presentes en su contexto e implementa un patrón fachada para los patrones de Identity Map y el Unit of Work. El Identity Map mantiene un único mapeo de cada identidad por Session, eliminando los problemas de identidades duplicadas.

Cuando se consulta una entidad con un identificador específico, SQLAlchemy verifica si esa entidad ya está en la memoria. Si es así, devuelve la instancia existente en lugar de cargarla nuevamente desde la base de datos.

**Activar/Desactivar:**
El ORM controla el Identity Map internamente, y no brinda una opción directa para activarlo/desactivarlo manualmente.

### Unit of Work

**¿En qué consiste?:**
El patrón Unit of Work encapsula un conjunto de operaciones de datos, y una lista de acciones que son necesarias para considerar exitosa la transacción, de esta manera se garantiza la consistencia. Es usado en problemas de concurrencia y estabilidad.

**¿Para qué lo usa el ORM?:**
Garantiza que las operaciones de inserción, actualización y eliminación se realicen en conjunto como una transacción única, evitando cambios parciales y asegurando la integridad de la base de datos.

**¿Cómo lo usa el ORM?:**

La clase Session implementa el patrón Unity of Work, para proveer un proceso de persistencia de los cambios de estados de la base de datos. Todas las operaciones de manipulación de datos se realizan en una sesión y se confirman como una transacción única.

**Activar/Desactivar:**

El ORM brinda la capacidad de activar y desactivar explícitamente las transacciones utilizando el método 'commit()' para confirmar los cambios o 'rollback()' para revertir los cambios en una sesión.

## Data Mapper

**¿En qué consiste?:**

El patrón Data Mapper separa la lógica de negocio del código de acceso a la base de datos, asignando una clase independiente para mapear los objetos de dominio a las tablas de la base de datos.

**¿Para qué lo usa el ORM?:**

Facilita el mapeo entre objetos de dominio y la estructura de la base de datos sin que la lógica de la base de datos afecte directamente a los objetos de dominio.

**¿Cómo lo usa el ORM?:**

SQLAlchemy implementa un Data Mapper a través de las clases de modelos de SQLAlchemy. Cada clase de modelo representa una entidad de la base de datos y proporciona un mapeo entre la estructura de la base de datos y los objetos de dominio. A través de mapper() y Table se pueden lograr la utilización del patrón.

**Activar/Desactivar:**

En SQLAlchemy, el uso del Data Mapper está intrínsecamente integrado en la definición de clases de modelos. No hay una opción directa para activar/desactivar esta funcionalidad de forma independiente.

## Lazy Loading

**¿En qué consiste?:**

Carga una colección u objeto relacionado según se necesite.

**¿Para qué lo usa el ORM?:**

Se utiliza cuando accedes a una colección (como una lista de objetos relacionados), SQLAlchemy no carga automáticamente todos los objetos relacionados. En cambio, emite una consulta adicional para cargar los objetos solo cuando los necesitas. Esto es útil para evitar cargar grandes cantidades de datos innecesarios de la base de datos hasta que realmente los requieras.

**¿Cómo lo usa el ORM?:**

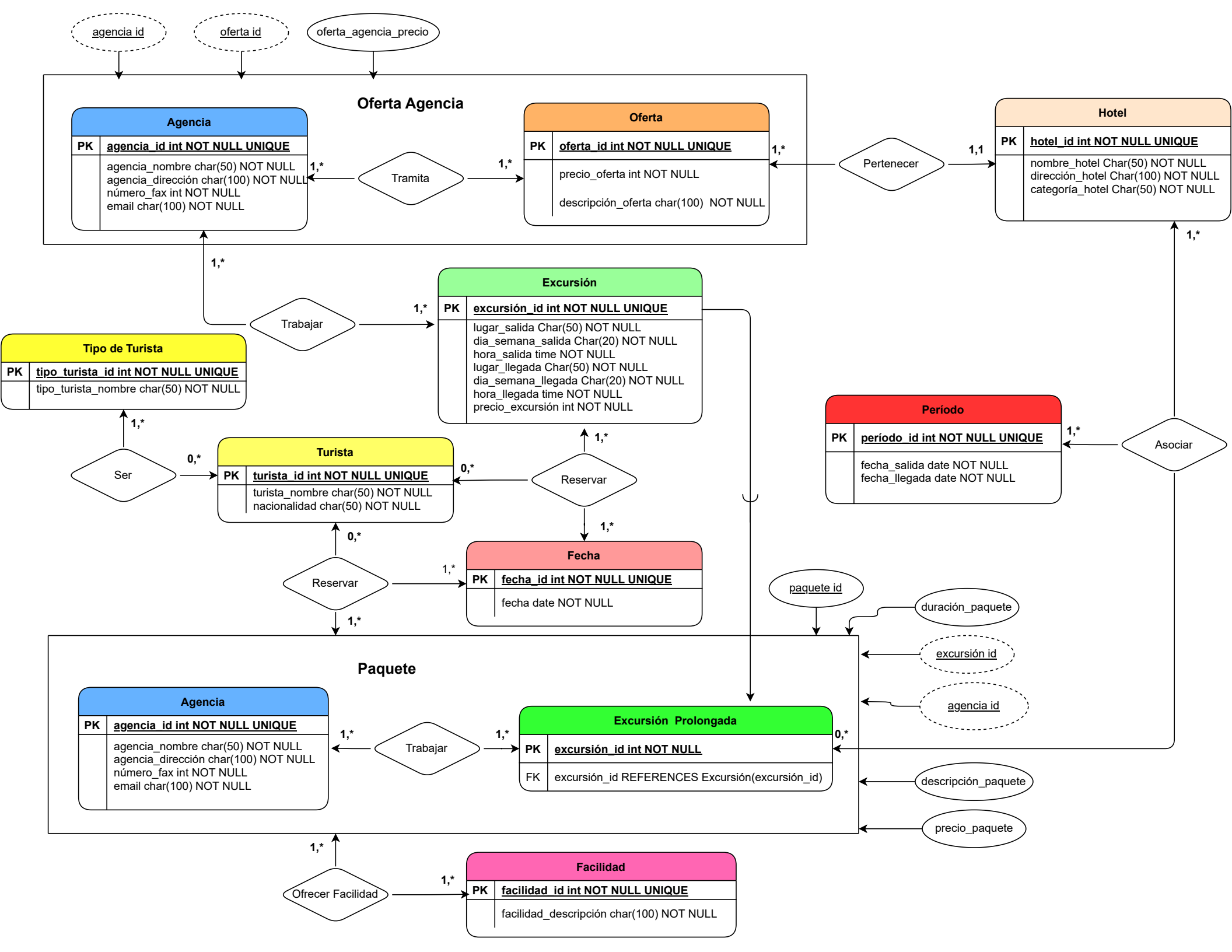
Este comportamiento se puede configurar en el momento de construcción del mapeador utilizando el parámetro relationship.lazy en la función relationship(), así como mediante el uso de opciones de carga ORM con la construcción Select. La carga de relaciones se divide en tres categorías: carga *perezosa*, carga *ansiosa* y *sin* carga. La carga perezosa se refiere a objetos que se devuelven de una consulta sin que los objetos relacionados se carguen inicialmente. Cuando se accede por primera vez a la colección o referencia dada en un objeto particular, se emite una declaración SELECT adicional para cargar la colección solicitada. Además se pueden usar otras formas de carga como:

- lazy loading - disponible mediante lazy='select' o utilizando la opción lazyload()
- select IN loading - disponible mediante lazy='selectin' o utilizando la opción selectinload()
- joined loading - disponible mediante lazy='joined' o utilizando la opción joinedload()

- raise loading - disponible mediante lazy='raise', o utilizando la opción raiseload()
- subquery loading - disponible mediante lazy='subquery' o utilizando la opción subqueryload()

# 7. Modelo de datos

Para mejor visualización se adjuntó en la siguiente página.


agencia id
oferta id
oferta_agencia_precio
Oferta Agencia
Agencia
PK
agencia_id int NOT NULL UNIQUE
agencia_nombre char(50) NOT NULL
agencia_dirección char(100) NOT NULL
número_fax int NOT NULL
email char(100) NOT NULL
1,*
Tramita
1,*
Oferta
PK
oferta_id int NOT NULL UNIQUE
precio_oferta int NOT NULL
descripción_oferta char(100) NOT NULL
1,*
Pertenecer
1,1
Hotel
PK
hotel_id int NOT NULL UNIQUE
nombre_hotel Char(50) NOT NULL
dirección_hotel Char(100) NOT NULL
categoría_hotel Char(50) NOT NULL
1,*
1,*
Trabajar
1,*
Excursión
PK
excursión_id int NOT NULL UNIQUE
lugar_salida Char(50) NOT NULL
dia_semana_salida Char(20) NOT NULL
hora_salida time NOT NULL
lugar_llegada Char(50) NOT NULL
dia_semana_llegada Char(20) NOT NULL
hora_llegada time NOT NULL
precio_excursión int NOT NULL
Tipo de Turista
PK
tipo_turista_id int NOT NULL UNIQUE
tipo_turista_nombre char(50) NOT NULL
1,*
Ser
0,*
Turista
PK
turista_id int NOT NULL UNIQUE
turista_nombre char(50) NOT NULL
nacionalidad char(50) NOT NULL
0,*
Reservar
1,*
Período
PK
período_id int NOT NULL UNIQUE
fecha_salida date NOT NULL
fecha_llegada date NOT NULL
1,*
Asociar
0,*
Reservar
1,*
Fecha
PK
fecha_id int NOT NULL UNIQUE
fecha date NOT NULL
1,*
paquete id
duración_paquete
excursión id
agencia id
descripción_paquete
precio_paquete
Paquete
Agencia
PK
agencia_id int NOT NULL UNIQUE
agencia_nombre char(50) NOT NULL
agencia_dirección char(100) NOT NULL
número_fax int NOT NULL
email char(100) NOT NULL
1,*
Trabajar
1,*
Excursión Prolongada
PK
excursión_id int NOT NULL
FK
excursión_id REFERENCES Excursión(excursión_id)
0,*
1,*
Ofrecer Facilidad
1,*
Facilidad
PK
facilidad_id int NOT NULL UNIQUE
facilidad_descripción char(100) NOT NULL