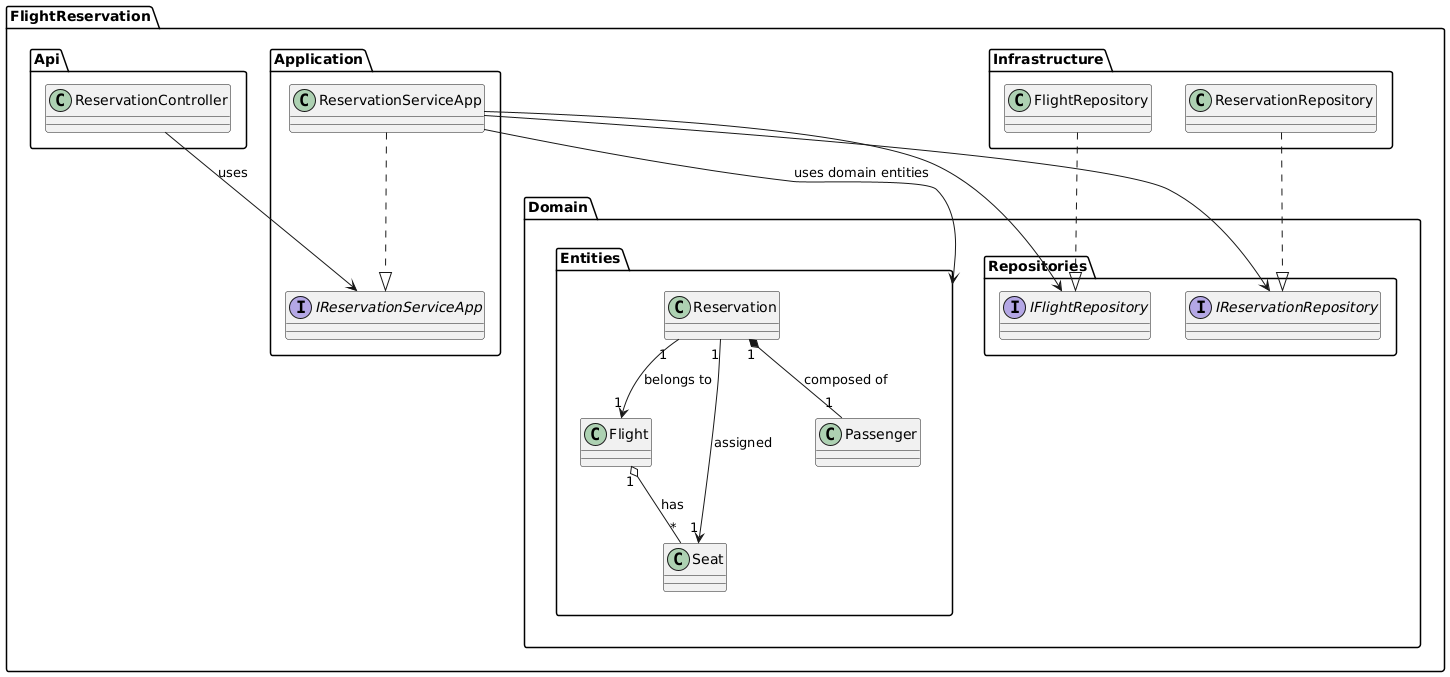
**Arquitectura del Ejemplo Base Reservar Cita**

**Clean Architecture**



**Propósito**

Esta arquitectura busca:

* Separar claramente **responsabilidades** y **niveles de abstracción**.
* Hacer el sistema **escalable**, **testeable**, **mantenible** y **orientado a dominio**.
* Enseñar a los estudiantes conceptos **profesionales**, aplicables en proyectos reales.

**Capas de la Arquitectura**

**1. FlightReservation.Api (Capa de Presentación / Entrada)**

* Expone los **endpoints HTTP** a través de controladores (ReservationController).
* No contiene lógica de negocio ni acceso a datos.
* Inyecta dependencias y orquesta la ejecución de casos de uso.

**2. FlightReservation.Application (Capa de Aplicación / Casos de Uso)**

* Implementa la **lógica de orquestación de casos de uso**.
* Usa servicios como ReservationServiceApp, definidos por interfaces (IReservationServiceApp).
* Esta capa:
  + No conoce infraestructura.
  + Depende solo de interfaces de repositorios del dominio.

**Responsabilidad:**

* Orquestar: obtener vuelo, buscar silla, crear pasajero, registrar reserva.
* Sin lógica de negocio profunda ni validaciones de dominio complejas.

**3. FlightReservation.Domain (Capa de Dominio)**

**Subpaquete: Entities**

* Contiene el **modelo del negocio**:
  + Flight, Seat, Reservation, Passenger.
* Las entidades incluyen **invarianzas** y lógica relevante (e.g., Seat.Reserve()).

**Subpaquete: Repositories**

* Define **interfaces de repositorios** (IFlightRepository, IReservationRepository).
* Actúan como **puertos** que Application puede invocar sin conocer detalles de almacenamiento.

**4. FlightReservation.Infrastructure (Capa de Infraestructura / Adaptadores)**

* Implementa los **repositorios reales** con Entity Framework Core y SQLite.
  + FlightRepository implementa IFlightRepository
  + ReservationRepository implementa IReservationRepository
* Contiene el DbContext: FlightReservationDbContext.
* También incluye SeedData para poblar datos iniciales.

**Esta capa depende de:**

* Domain (entidades)
* Application (contratos que implementa)

**Inversión de Dependencias**

Este diseño aplica el **principio de inversión de dependencias (DIP)**:

* Application depende de **interfaces**, no de clases concretas.
* Infrastructure implementa esas interfaces.
* Api inyecta todo en tiempo de ejecución (por ejemplo, en Program.cs).

| **Capa** | **Responsabilidad** | **Puede depender de** |
| --- | --- | --- |
| **Domain** | Entidades, lógica de negocio | Independiente de todo |
| **Application** | Casos de uso, orquestación | Domain |
| **Infrastructure** | Persistencia, tecnologías externas | Application, Domain |
| **Api** | Controladores HTTP, entrada de usuario | Application, Domain *(opcional: Infrastructure para inyección)* |