

Class Diagram

```

classDiagram
    class ClientGateway {
        - Clients: List<Client>
        + create(name: String, age: int, clientID: String): Client
        + findClient(clientID): Client
        - ClientCatalogue()
    }
    class Client {
        + name: String
        + age: Int
        + clientID: String
    }
    class Ticket {
        + ticketID: int
        + reservation: Reservation
    }
    class Reservation {
        + reservationID
    }
    class Console {
        - catalogueConnection: ConnectionCatalogue
        - catalogueRoute: RouteCatalogue
        - catalogueClient: ClientCatalogue
        - tripList
        + searchForConnections(arrivalTime: LocalTime, departureTime: LocalTime, arrivalCity: String, departureCity: String, daysOfOperation: String, trainType: String, ticketFirstClass: float, ticketSecondClass: float): List<Connections>
        + sort(connection.totalCost(), connection.totalDuration()): List<Connection>
        + pickedConnection(): Connection
        + book(name: List<String>, age: List<int>, clientID: List<String>, connection: Connection): String
        + getTicket(): Ticket
    }
    class TripGateway {
        - trips: List<Trip>
        + searchTrip(name: String, age: int, clientID: String)
        + create(): Trip
    }
    class Trip {
        + tripID: String
        + reservations: List<Reservation>
        + Connection connection
    }
    class ConnectionCatalogue {
        - Connections: List<Connection>
        + create(routes: List<Routes>): Connection
        - ConnectionCatalogue()
        - isValidLayover(ArrayList<Routes> routes)
    }
    class RouteGateway {
        - routes: List<Route>
        + create(): Route
        - RouteCatalogue()
    }
    class Route {
        - routeID: String
        + departureTime: LocalTime
        + arrivalTime: LocalTime
        + arrivalCity: String
        + departureCity: String
        + trainType: String
        + firstClassTicket: float
        + secondClassTicket: float
        + daysOfOperation: ArrayList<String>
        + duration(): LocalTime
        + Route(routeID, departureTime, arrivalTime, arrivalCity, departureCity, trainType, firstClassTicket, secondClassTicket, daysofOperation)
    }
    class Connection {
        - connection : List<Route>
        + totalDuration(): LocalTime
        + totalCost(): float
        + Connection(List<Route>)
    }

    ClientGateway --> Client : searches
    ClientGateway --> Ticket
    Client --> Ticket
    Ticket --> Reservation : aggregation
    Reservation --> Trip : aggregation
    Console --> ClientGateway : searches
    Console --> TripGateway
    Console --> ConnectionCatalogue : uses
    TripGateway --> Trip : aggregation
    Trip --> ConnectionCatalogue
    ConnectionCatalogue --> RouteGateway : searches through
    RouteGateway --> Route : aggregation (1 to *)
    ConnectionCatalogue --> Connection : aggregation (1 to *)
    Connection --> Route : aggregation (1 to 3)
  
```

The diagram illustrates the structure of a travel system. Key components and relationships include:

- ClientGateway** manages **Client** objects and interacts with **Ticket**.
- Client** provides personal information and is associated with a **Ticket**.
- Ticket** is associated with a **Reservation**, which in turn is associated with a **Trip**.
- Console** is the central interface, interacting with **ClientGateway**, **TripGateway**, **ConnectionCatalogue**, and **RouteGateway**.
- TripGateway** manages **Trip** objects, which are associated with **ConnectionCatalogue**.
- ConnectionCatalogue** manages **Connection** objects, which are associated with **RouteGateway**.
- RouteGateway** manages **Route** objects, which are associated with **Connection**.
- Connection** is associated with **Route** objects.

Additional notes from the diagram:

- private constructor due to being a Singleton** (for ClientGateway, TripGateway, ConnectionCatalogue, RouteGateway).
- The system would acknowledge the successful booking. returns a String** (for Console.book).
- RouteCatalogue is an aggregate of Route** (for RouteGateway).
- Client calls system.searchForConnections(...) System then accesses routeCatalogue and get routes objects, it then evaluates possible combinations and for each valid sequence it has the connection catalogue create a connection object.** (for Console.searchForConnections).
- pickedConnection would initiate booking and user input prompt** (for Console.pickedConnection).
- Connections are computed dynamically based on the user's search criteria.** (for Console.sort).
- For a given connection, a client may only have a reservation under their name.** (for Trip.reservations).
- private constructor due to being a Singleton** (for Connection).

Client calls
system.searchForConnections(...)
System then accesses
routeCatalogue and get routes
objects, it then evaluates possible
combinations and for each valid
sequence it **has the connection
catalogue** create a connection
object.

private constructor due
to being a Singleton