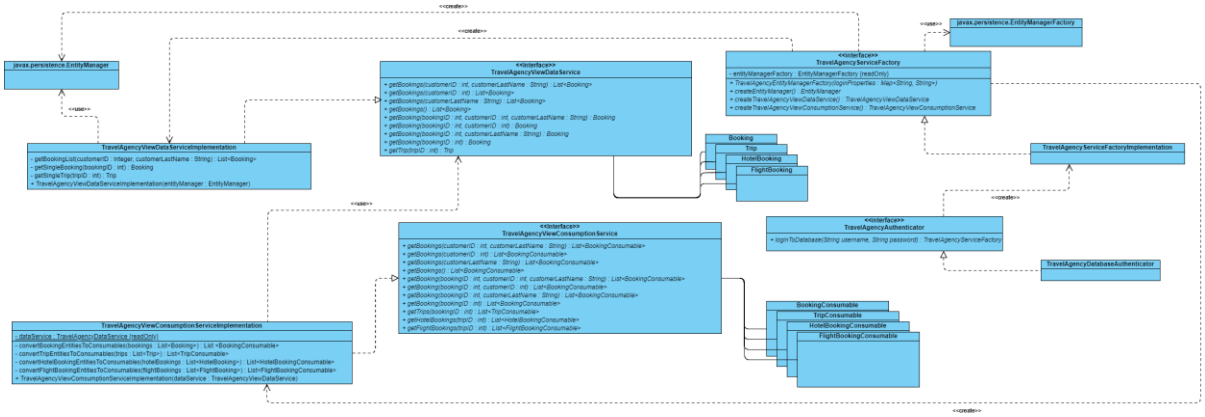
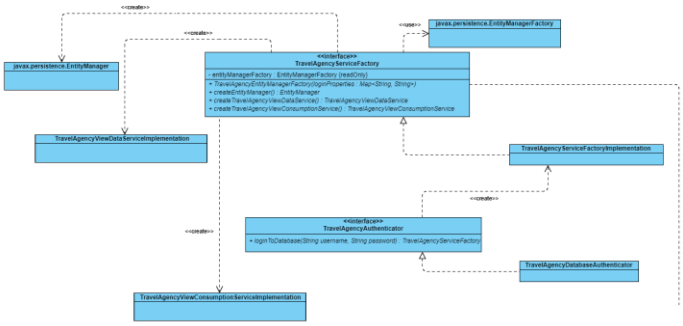
Table of Contents

1 Service Architecture – Class Diagram

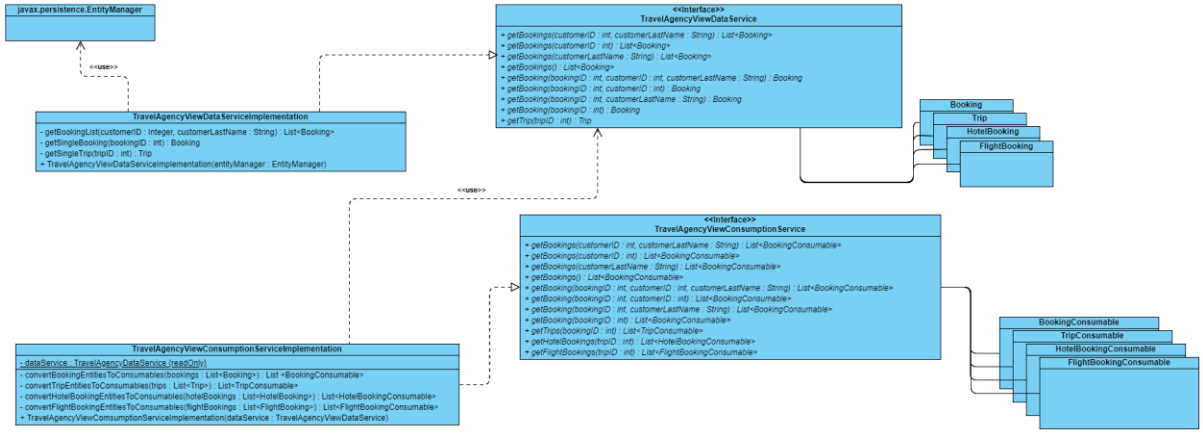
This UML-Class-Diagram models the architecture of the service implemented in this module. As this module only implements a service which returns entities from the database (in the following called *view service*), it will only contain elements for such service.
However, since the entire architecture is very modular, additional services may easily be added in the future.
The service architecture consists of a login architecture (*TravelAgencyServiceAuthenticator* & *TravelAgencyServiceFactory*) and a service architecture (*TravelAgencyViewDataService* & *TravelAgencyViewConsumptionService*).
The login architecture is designed to verify the users credentials and if correct establish a connection to the database using the JPA framework (javax.persistence) with the entities from the database scheme (see more at *travel-agency-service/docs/database/Database – UML Diagrams.pdf*).
The service architecture is designed to use the established database connection to load entities from the database filtered by parameters provided by the user. In this case, the data service has the responsibility to retrieve the entities specified from the database while the consumption service has the responsibility to convert those jpa-entities into objects useful for the user interface.



Login architecture extract:

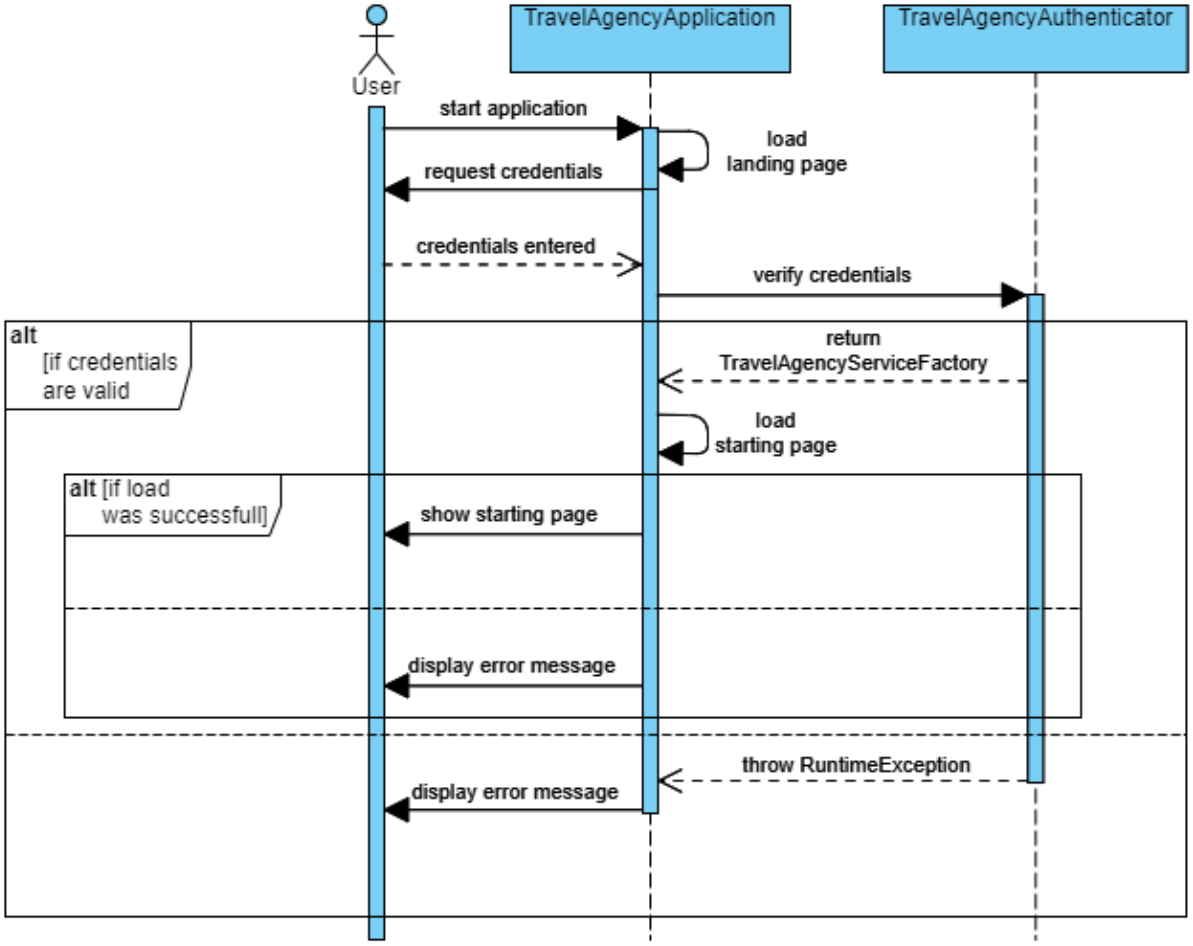


Service architecture extract:


javax.persistence.EntityManager
<<use>>
TravelAgencyViewDataServiceImplementation
- getBookingList(customerID : Integer, customerLastName : String) : List<Booking>
- getSingleBooking(bookingID : int) : Booking
- getSingleTrip(tripID : int) : Trip
+ TravelAgencyViewDataServiceImplementation(entityManager : EntityManager)
<<Interface>>
TravelAgencyViewDataService
+ getBookings(customerID : int, customerLastName : String) : List<Booking>
+ getBookings(customerID : int) : List<Booking>
+ getBookings(customerLastName : String) : List<Booking>
+ getBookings() : List<Booking>
+ getBooking(bookingID : int, customerID : int, customerLastName : String) : Booking
+ getBooking(bookingID : int, customerID : int) : Booking
+ getBooking(bookingID : int, customerLastName : String) : Booking
+ getBooking(bookingID : int) : Booking
+ getTrip(tripID : int) : Trip
Booking
Trip
HotelBooking
FlightBooking
<<use>>
<<interface>>
TravelAgencyViewConsumptionService
+ getBookings(customerID : int, customerLastName : String) : List<BookingConsumable>
+ getBookings(customerID : int) : List<BookingConsumable>
+ getBookings(customerLastName : String) : List<BookingConsumable>
+ getBookings() : List<BookingConsumable>
+ getBooking(bookingID : int, customerID : int, customerLastName : String) : List<BookingConsumable>
+ getBooking(bookingID : int, customerID : int) : List<BookingConsumable>
+ getBooking(bookingID : int, customerLastName : String) : List<BookingConsumable>
+ getBooking(bookingID : int) : List<BookingConsumable>
+ getTrips(bookingID : int) : List<TripConsumable>
+ getHotelBookings(tripID : int) : List<HotelBookingConsumable>
+ getFlightBookings(tripID : int) : List<FlightBookingConsumable>
TravelAgencyViewConsumptionServiceImplementation
- dataService : TravelAgencyDataService {readOnly}
- convertBookingEntitiesToConsumables(bookings : List<Booking>) : List <BookingConsumable>
- convertTripEntitiesToConsumables(trips : List<Trip>) : List<TripConsumable>
- convertHotelBookingEntitiesToConsumables(hotelBookings : List<HotelBooking>) : List<HotelBookingConsumable>
- convertFlightBookingEntitiesToConsumables(flightBookings : List<FlightBooking>) : List<FlightBookingConsumable>
+ TravelAgencyViewComsumptionServiceImplementation(dataService : TravelAgencyViewDataService)
BookingConsumable
TripConsumable
HotelBookingConsumable
FlightBookingConsumable

2 Login Process – Sequence Diagram

This UML-Sequence-Diagram shows the sequential order of interaction between the objects in question from the start of the application until the database connection is established, or in case of invalid credentials or other errors, a corresponding message is returned.
For reasons of simplicity, the front-end application is reduced to the *TravelAgencyServiceApplication* instance and the different *Controller* objects are left out. In the case of their interaction, the *Application* will be representative for them.
Note that after the display of an error message, the entire process may start over from *request credentials*.



3 Service Process – Sequence Diagram

This UML-Sequence-Diagram shows the sequential order of interaction between the objects in question from when the database connection is established to the display of flights and hotels associated with a booking.
For reasons of simplicity, the front-end application is reduced to the *TravelAgencyServiceApplication* instance and the different *Controller* objects are left out. In the case of their interaction, the *Application* will be representative for them.
Note that after the display of an error message, the entire process may start over from *request credentials*.