PROJECT LOG AND IDEAS:

**22/03/23: (14:18 PM )** Currently working on choosing the idea to work with for the database since I can’t work with anything relating to soccer.   
Initial Ideas:  
1. Online Shopping System  
2. Travel Booking system  
3. Game Shop  
  
Decided to with Travel Booking system, seems like something I haven’t done for any other module and would like to work on something new.  
  
**22/03/23: (14:55 PM )** Want to setup the entities for the **TRAVEL BOOKING SYSTEM** and the fields they will entail.  
For a travel booking system, these are the entities I will work on implementing

1. User: This entity will contain information about users who use the system for booking travel services. Fields can include an **integer ID**, a **string field for the username**, and **string fields for the user's name, email address, and phone number.**
2. Trip: This entity will contain information about the trips available for booking. Fields can include an **integer ID**, a **string field for the trip name**, a **string field for the trip description**, a **floating field for the trip cost,** and an **integer field for the maximum number of people** allowed on the trip.
3. Booking: This entity will contain information about the bookings made by users. Fields can include **an integer ID**, an **integer field for the user ID**, an integer **field for the trip ID**, a **string field for the booking status** (such as "pending", "confirmed", or "cancelled"), and a **floating field for the booking cost**.
4. Payment: This entity will contain information about the payments made by users for their bookings. Fields can include an **integer ID**, an **integer field for the booking ID**, a **string field for the payment method,** and a **floating field for the payment amount**.
5. Location: This entity will contain information about the locations that users can travel to. Fields can include **an integer ID**, a **string field for the location name**, a **string field for the location description**, and a **floating field for the location cost**.
6. Review: This entity will contain information about the reviews submitted by users for the trips they have taken. Fields can **include an integer ID**, an **integer field for the user ID**, an **integer field for the trip ID**, a **string field for the review text**, and an **integer field for the review rating.**

**22/03/23: (15:05 PM )** Database Modelling initial setup might be a bit complex so possible adjustments can be made in the future  
  
In this database model, each entity is represented by a table with its corresponding fields. The relationships between the entities are indicated by the connecting lines.

The **User** table has a one-to-many relationship with the **Booking** table, as a user can make multiple bookings. The **Booking** table also has a one-to-many relationship with the **Payment** table, as a booking can have multiple payments associated with it.

The **Trip** table has a many-to-many relationship with the **Location** table, as a trip can have multiple locations and a location can be part of multiple trips. This relationship is represented by the **TripLocation** junction table, which contains foreign keys to both the **Trip** and **Location** tables.

The **Review** table has a one-to-many relationship with both the **User** and **Trip** tables, as a **user** can submit multiple **reviews** for different **trips** and a trip can have multiple reviews submitted by different users.

**22/03/23: (16:37 PM )** Decided to make changes and amendments to the booking system to reduce the complications of the system.  
  
The new travel system will be implemented like this:  
1. **Customer table** - This table would contain information about registered customers such as their customer ID, name, email, phone number and address.  
  
2. **Booking table** - This table would contain information about each booking made by a customer. It would include the booking ID, flight id, customer ID, travel date, travel time and seats.

3. **Flight table** - This table would contain information about each available flight. It would include the flight ID, airport ID, departure location, arrival location, airline name, duration(mins) and cost.  
  
4. **Airport table** - This table would contain information about each airport. It would include the airport ID, name, location.  
  
5. **Payment table** - This table would contain information about each payment made by a user. It would include the payment ID, booking ID, amount paid(which will be flight cost multiplied with the number of seats in the booking), payment date, and payment method.

**22/03/23: (17:23 PM )** Database Model and Diagram done and attached in github.

**22/03/23: (17:23 PM – 18:08PM)** Database populated.  
  
**22/03/23: (18:23PM)** Application Folder Structure Created in Intellij  
  
**22/03/23: (18:25PM)** Using the DAO\_product example I made up my project folder structure.  
  
**22/03/23: (18:32PM)** Made an initial base for the Customer DTO.   
  
**22/03/23: (18:38PM)** Made an initial base for the Airport DTO.  
  
**22/03/23: (18:42PM)** Made an initial base for the Flight DTO.  
  
**22/03/23: (18:47PM)** Made an initial base for the Booking DTO.  
  
**22/03/23: (18:52PM)** Made an initial base for the Payment DTO.  
  
**22/03/23: (19:00 PM )** Added JDBC driver to the Driver  
  
**22/03/23: (19:12 PM )** Added JDBC connection and DaoException.  
  
**22/03/23: (19:17 PM )** Added CustomerDaoInterface with Features 1-4.  
  
**22/03/23: (20:45 PM )** Implemented findAllCustomers() method.  
  
**22/03/23: (21:08 PM )** Implemented testFindAllCustomers() - Junit Test and Successful.  
  
**22/03/23: (21:45 PM )** Implemented findCustomerById(int customerId)

method.  
  
**22/03/23: (22:21 PM )** Implemented testFindCustomerById()

- Junit Test and Successful.

**22/03/23: (22:32 PM )** Implemented deleteCustomerById(int customerId) method.  
  
**22/03/23: (22:46 PM )** Implemented testDeleteCustomerById ()

- Junit Test and Successful.

**22/03/23: (22:48 PM )** Implemented insertCustomer(Customer customer)

method.  
  
**22/03/23: (23:06 PM )** Implemented testInsertCustomer()

- Junit Test and Successful. While working on this test, it was failed a couple times then I had to change how I did the auto increment of my id for all DTOs and it was successful.  
  
**22/03/23: (23:16 PM )** Added AirportDaoInterface with Features 1-4.

**22/03/23: (23:21 PM )** Implemented findAllAirports() method.  
  
**22/03/23: (23:23 PM )** Implemented testFindAllAirports() - Junit Test and Successful.

**22/03/23: (23:26 PM )** Implemented findAirportById(int airportId)

method.  
  
**22/03/23: (23:28 PM )** Implemented testFindAirportById() - Junit Test and Successful.  
  
 **THURSDAY: 23/03/23**

**23/03/23: (23:42 PM )** Implemented deleteAirportById(int airportId)

method.  
  
**23/03/23: (23:46 PM )** Implemented testDeleteAirportById() - Junit Test and Successful.  
  
**23/03/23: (23:49 PM )** Implemented *insertAirport(Airport airport)*

method.  
  
**23/03/23: (23:56 PM )** Implemented testInsertAirport() - Junit Test and Successful.  
  
 **FRIDAY: 24/03/23**  
  
**24/03/23: (00:10 PM )** Added FlightDaoInterface with Features 1-4.  
  
**24/03/23: (00:23 PM )** Implemented findAllFlights() method.

**24/03/23: (00:28 PM )** Implemented testFindAllFlights() - Junit Test and Successful.  
  
 **SATURDAY: 25/03/23**  
  
**25/03/23: (17:02 PM )** Implemented findFlightById(int flightId)

method.  
  
**25/03/23: (17:06 PM )** Implemented testFindFlightById()- Junit Test and successful.  
  
**25/03/23: (17:13 PM )** Implemented deleteFlightById(int flightId)

method.  
  
**25/03/23: (17:19 PM )** Implemented testDeleteFlightById()- Junit Test and successful.

**25/03/23: (17:22 PM )** Implemented insertFlight(Flight flight) method.  
  
**25/03/23: (17:29 PM )** Implemented testInsertFlight()- Junit Test and successful.  
  
**25/03/23: (17:51 PM )** Added BookingDaoInterface with Features 1-4.

**25/03/23: (17:58 PM )** Implemented findAllBookings() method.  
  
**25/03/23: (18:00 PM )** Implemented testFindAllBookings() - Junit Test and Successful  
  
**25/03/23: (18:05 PM )** Implemented findBookingById(int bookingId) method.  
  
**25/03/23: (18:16 PM )** Implemented testFindBookingById()- Junit Test and successful.  
  
**25/03/23: (19:42 PM )** Implemented deleteBookingById(int bookingId) method.  
  
**25/03/23: (19:58 PM )** Implemented testDeleteBookingById()- Junit Test and successful.  
  
**25/03/23: (20:03 PM )** Implemented insertBooking(Booking booking) method.  
  
**25/03/23: (20:24 PM )** Implemented testInsertBooking()- Junit Test and successful.

**25/03/23: (20:30 PM )** Added PaymentDaoInterface with Features 1-4.  
  
**25/03/23: (20:35 PM )** Implemented findAllPayments() method.  
  
**25/03/23: (20:55 PM )** Implemented testFindAllPayment() - Junit Test and Successful  
  
**25/03/23: (21:35 PM )** Implemented findPaymentById(int paymentId) method.  
  
**25/03/23: (21:37 PM )** Implemented testFindPaymentById() - Junit Test and Successful  
  
**25/03/23: (21:42 PM )** Implemented deletePaymentById(int paymentId) method.

**25/03/23: (21:45 PM )** Implemented testDeletePaymentId() - Junit Test and Successful  
  
**25/03/23: (22:10 PM )** Implemented insertPayment(Payment payment) method.  
  
**25/03/23: (22:13 PM )** Implemented testInsertPayment()- Junit Test and successful.  
  
  
 **SUNDAY: 26/03/23  
26/03/23: (12:00 AM )** Created a new database setup to add numbers as the primary keys instead of just ids.

**26/03/23: (12:40 AM )** Modified All parts with Customers to work with new database  
  
**26/03/23: (03:01 AM )** Modified All parts with Airports to work with new database  
  
**26/03/23: (03:24 AM )** Modified All parts with Flights to work with new database  
  
**26/03/23: (03:56 AM )** Modified All parts with Bookings to work with new database  
  
**26/03/23: (04:10 AM )** Modified All parts with Payments to work with new database  
  
**26/03/23: (04:55 AM )** Since I have finished working on all the features for this week. Time to make up my menu system. So I made up my menu to be able to handle wrong input and probably never break and run on a loop. Then added in the initial menu and submenu structure. I thought of the errors that might rise up due to foreign key constraints so in that case I will finish up the easy display all methods.  
  
**26/03/23: (05:35 AM )** Just added the findByNumber() methods was quite straightforward. Note that already in the MySqlDao for the respective entities I already had lowercase implemented there to make sure that it prevents errors from arising.  
  
**26/03/23: (13:42 PM )** Since I am working on all 5 of my entities I have to check for foreign key constraints before I can delete some entities. Just like Customer which is a parent table for Bookings, I can’t drop a Customer that still has bookings. So I will be making an extra method that when the user wants to drop a customer that already has bookings, the booking will be displayed.  
  
**26/03/23: (13:46 PM )** The deleteAirport is also similar, foreign key constraints on the flighst table, so would have to make findAllFlightsByAirportNumber method.   
  
**26/03/23: (14:03 PM )** The deleteFlight is also similar, foreign key constraints on the Booking table, so would have to make findAllBookingsByFlightNumber method.  
  
**26/03/23: (14:10 PM )** The deleteBooking is also similar, foreign key constraints on the Payment table, so would have to make findAllPay,entsByBookingNumber method.  
  
**26/03/23: (16:08 PM )** Working on insertCustomers, since we are working with a database the right thing to do is to prevent invalid data from passing through. So in that case I will add data validation for all inputs before passing in the data. Provided validation for all necessary files with the readField() method and also validations for fields like email and telephone.  
  
**26/03/23: (16:12 PM )** Working on insertAiport, almost had the same approach like the insertCustomers, I just added to the already made readField to validate and check the airportNumber, airportName and airportLocation.  
  
**26/03/23: (16:33 PM )** For the insertFlight it was a bit different, added the flightNumber check in the readField, created a separate readAirport that checks if the airportNumber exists and if so then it can be added this is done to prevent error that can be generated from the foreign key, because an aiportNumber that doesn’t exist in the airport table can’t be added. Added the departureLocation, arrivalLocation and airlineName into the readFied and created checks for the duration and the flightCost.  
  
**26/03/23: (16:59 PM )** For the insertBooking I added the bookingNumber in the readFiled to validate that it is not more than 10 and also check if it already exists in the database, then validated the flightNumber and customerNumber to make sure they already exist in the database. Made my normal validations for the traveldate, travelTime to prevent any error from getting into the database upon insertion. Then did a simple check to make the seat\_number between 1A and 100F.

**26/03/23: (17:32 PM )** For the insertPayment I added the paymentNumber to the readField and did the check for the bookingNumber to make sure that the bookingNumber already exists and did a normal doublr validation for the amount paid and a recent date validation.  
  
  
  
 **THURSDAY: 30/03/23**

**30/03/23: (23:50 PM )** Following the feedback received, I created a utility class called Helper that I will use to carry out all my validations and checks. And add more Junit tests for the extra interfaces added. Also added the departure time and arrival time to the Flight table.  
  
**30/03/23: (23:55 PM )** Thinking of how to implement the filter. Took me a while but I came up with an idea similar to that of normal websites, because filters are based off a field. For instance a shoe website, colors are made into a set and then you can filter based off those colors. My initial plan now is to filter customers based off address(City), filter airports based off location, filter flights based off time and airline name, filter booking with the date and filter payments based off methods. And also be able to sort ascending, descending or by default.