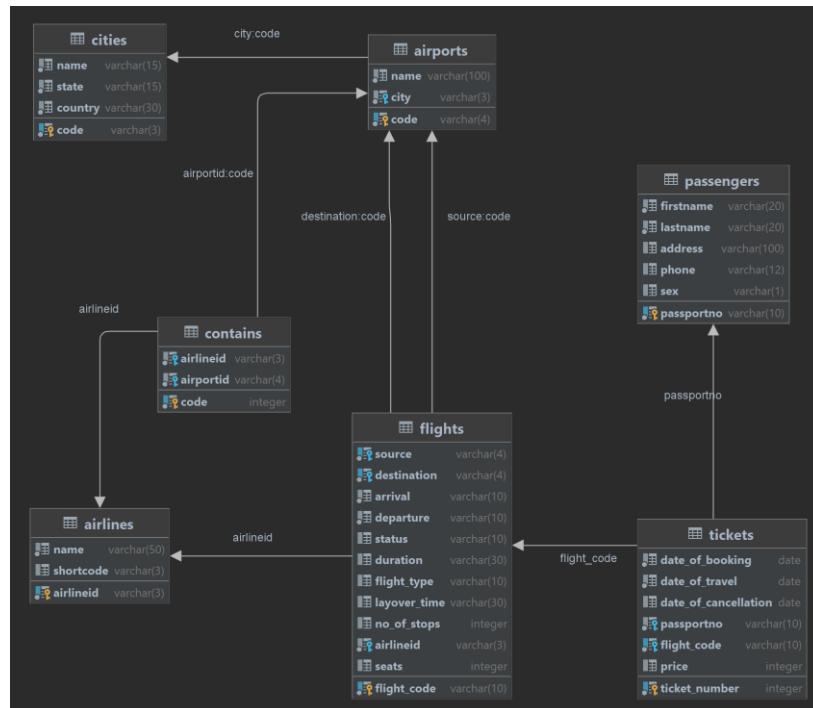


**Catch-up activity:**

**Flights**

The student must write both a JAVA application (using JavaFX as GUI) and a Spring Boot application to manage a flights database. The script to create the PostgreSQL database can be downloaded from AULES. The entity-relationship corresponding to this database is the following one:



Some things you should keep in mind when designing your application:

- The `flights.seats` column shows the total seats available in a flight. You shouldn't update this value.
- To know how many seats are available in a flight, you must subtract the number of tickets bought for the flight from the number of seats.
- A ticket can be cancelled. In that case, it shouldn't be taken into account for other operations.

On start, the application must permit the user to buy a ticket for a flight. The data necessary to buy a ticket can be seen in the following screenshot:

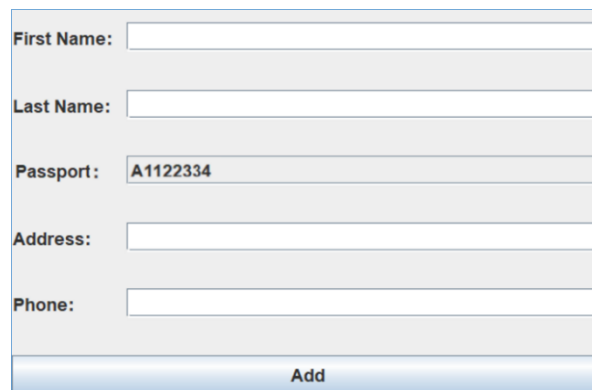
As it can be seen in the screenshot, origin and destination airport must be chosen with a combo box. Additionally, you can see that both the destination and flight combos are empty at start. That is because, in order to fill the destinations, we need to know first the origin (we only choose destinations available from the current origin with a flight). Also, to fill in the flights combo we need to know first both the origin and destination.

Finally, we should make some checking prior to adding the new ticket:

- The passenger cannot have another ticket bought for the same date.
- There are seats available for that flight.

Both checkings will be made throughout a stored function/procedure.

The last step will be to check if the specified passport exists in the database. If it does not exist, we should register the new passenger before registering the flight ticket. The fields that must be collected are the following:



The screenshot shows a web form for adding a new passenger. It contains five text input fields: 'First Name:', 'Last Name:', 'Passport:', 'Address:', and 'Phone:'. The 'Passport:' field is pre-filled with the value 'A1122334'. Below the input fields is a blue button labeled 'Add'.

Keep in mind that the passport field has been provided in the previous screen and shouldn't be editable. You must also check all the fields that require a format.

The student should deliver:

1. A REST API with all the necessary endpoints to permit the application retrieve the data needed from the database. If you have doubts about an endpoint, ask in the forums.
2. A Thymeleaf webApp with the specified functionality.
3. A JavaFX application with the same functionality. You can retrieve the data from a local database server (as we have made in the first period) or from the REST API (+1 point).
4. The REST API airport controller must include documentation and testing.
5. The JavaFX airport data manager must include documentation and testing.

If you have any questions/doubts about the requirements, please don't hesitate to ask in the forums.