# Exercises: Best Practices and Architecture

This document defines the **exercise assignments** for the ["Databases Advanced – Entity Framework" course @ Software University.](https://softuni.bg/trainings/1442/databases-advanced-entity-framework-october-2016)

## Photo Share System

You are given a project skeleton of a **Photo Sharing System**. Take a look at it to get more familiar with that project. The database is modeled by code first approach and the models are fine (in other words there is nothing to modify in PhotoShare.Models project).

But the other project PhotShare.Client is poorly written. Your task is to **improve the architecture of the project** and made the **code to be testable** (e.g. decouple data layer from presentation layer, implement **Repository** and/or **Unit of Work Pattern** etc.). Seed some more data in the database.

Then **implement the missing commands** by the hints given in each command class and **fix any bugs** in the already implemented code.

Finally **write unit tests** to check if commands AddTag, AddTagTo, RegisterUser, MakeFriends and UploadProfilePicture works as expected.

## \*Extend Photo Share System

Extend the **Photo Share System** by implementing two more commands:

* **Login <username> <password> -** log a user into the system and keep a reference to it until the Logout command is called
* **Logout –** log out a user of the system

**Modify all commands accordingly**. For example, the user can modify his profile information or add new friends only when he/she is logged in and so on.

## Bus Tickets System

Your task is to design a database for **Bus Tickets System** using the **code first** approach. The database should keep information about:

* **bus companies** - name, nationality, rating
* **tickets** - price, seat, customer, trip
* **customers** - first name, last name, date of birth, gender, home town
* **trips** - departure time, arrival time, status, origin bus station, destination bus station, bus company
* **bus stations** - name, town
* **towns** - name, country
* **reviews** - content, grade, bus station, customer, date and time of publishing
* **bank accounts** - account number, balance, customer

Each ticket is bought from a customer for certain trip. A customer has only one home town. Every trip has origin and destination bus station and it is organized by only one bus company. A bus station is located in only one town and one town can have many bus stations located in it. Reviews are written for a certain bus company and a bus company can have many reviews. One customer can write many reviews but a single review can be written only by one customer. Bank account can be owned by one customer and each customer can own only one bank account.

Gender can be only male, female or not specified. The status of the trip can be departed, arrived, delayed or cancelled. Grade of a review can any be a floating-point number in range [1, 10].

When database is up and running **seed** it with some **sample records in each table**.

Now let’s **make a command line application** that would **use that database and provide the following functionalities**:

* **print information for trips for a given bus station –** Print a list of arrivals and departures buses for given bus station id in the format provided below
* **buy ticket –** Insert new ticket and reduce the balance from customers’ bank account. Consider managing all cases such as the customer does not have enough money in his/her bank account.
* **publish review –** insert new review from given user into the database
* **print reviews –** print a list of reviews for a given bus company in the format provided below

|  |  |
| --- | --- |
| **Command** | **Successful Output** |
| print-info {Bus Station ID} | {Bus Station Name}, {Town}  Arrivals:  From {origin station} | Arrive at: {Arrival Time} | Status: {status}  Departures:  To {destination station} | Depart at: {Departure Time} | Status {status} |
| buy-ticket {customer ID} {Trip ID} {Price} {Seat} | Customer {Full Name} bought ticket for trip {Trip ID} for {Price} on seat {Seat} |
| publish-review {Customer ID} {Grade} {Bus Company Name} {Content} | Customer {Full Name} published review for company {Company Name} |
| print-reviews {Bus Company ID} | {Id} {Grade} {Date and Time}  {Customer Full Name}  {Content} |

The application should **receive and execute unlimited number of commands** until the **“exit”** command is received.

Use all of the **best practices** you know to design and write your application. Finally **write unit tests** to make **sure all these functionalities** work correctly.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| print-info 1  exit | Sofia Central Station, Sofia  Arrivals:  From: Burgas | Arrive at: 14:30 | Status: Departed  From: Svishtov | Arrive at: 07:30 | Status: Arrived  From: V.Tarnovo | Arrive at: 14:30 | Status: Departed  Departures:  To: Varna | Depart at: 14:40 | Status: Delayed  To: Plovdiv | Depart at: 15:30 | Status: Cancelled |
| buy-ticket 2 323 14.20 A4  buy-ticket 3 333 -12.00 B8  buy-ticket 9 874 6.90 A1  exit | Customer John Doe bought ticket for trip 323 for $14.20 on seat A4  Invalid price  Insufficient amount of money for customer Harry Potter with bank account number BGR33133235 |
| publish-review 2 10 BusTrip2000 Excellent trip! Look forward to travel again.  publish-review 3 2 BusCompany2001 Awful and dirty bus. Cannot recommend it to anyone.  exit | John Doe’s review was succesfully published  Chuck Norris’ review was successfully published |
| publish-review 2 8.5 BusTrip2000 Would recommend it but the driver needs to stop smoking while driving.  print-reviews 1  exit | John Doe’s review was succesfully published  1 10 2016/11/15 10:46:18  John Doe  Excellent trip! Look forward to travel again.  2 8.5 2016/11/18 12:20:00  John Doe  Would recommend it but the driver needs to stop smoking while driving. |

### Bonus task

Implement one additional command **change-trip-status**. That command would change the status of a given trip

|  |  |
| --- | --- |
| **Command** | **Successful Output** |
| change-trip-status {Trip Id} {New Status} | Trip from {Origin Bus Station Town} to {Destination Bus Station Town} on {Trip Departure Date and Time}  Status changed from {Old Status} to {New Status} |

Add **new table to the database** to keep information about **arrived trips** (actual arrival time, origin bus station, destination bus station, passengers count).

In case a **trip status is changed to Arrived, automatically add new record** in the **arrived trips table** with the required information.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| change-trip-status 2 Departed  change-trip-status 3 Cancelled  change-trip-status 134 Arrived  exit | Trip from Burgas to Sofia on 2016/11/15 10:45:00  Status changed from Cancelled to Departed  Trip from Sofia to Varna on 2016/11/15 10:00:00  Status changed from Delayed to Cancelled  Trip from Plovdiv to Varna on 2016/11/14 15:30:00  Status changed from Departed to Arrived  On 2016/11/14 22:32:43 - 34 passengers arrived at Varna from Plovdiv |

### Hint for Bonus Task

Use events ☺