## Implementation

The model  
For the model of the data base we used “*code first*” approach. There are tables for all primary type. Data base implements inheritance through the abstract class Client and its descendants: “Human” and “Firm”. This is done by using “*table per type*”, where the ID of the “Client” class is foreign key for the descendants. The Contract class implements the *IArchivable* interface, which gives opportunity for expired contracts being archived. There is an enumeration for holding information about loyalty of the clients. The Contract class has static collection of promotions, using struct Promotion. Each promotion have start date, end date and discount. Before saving of each contract “*GetPromotion*()” method checks is there active promotions, and if there are, records in the database the promotional price.

The custom exception  
 There is “*NoEntriesFound*” exception that catch an exception when the collection of all records from the table is empty.

Design patterns  
We used two design patterns for working with the database: “Repository” and “Unit of Work”, developing with them an abstract layer between the business logic and database. In this case this helps to remove the dependency of changing the type of data store, and improves the testing options. This interaction is implemented in *Entity Framework* on a specific form, but raising the level of abstraction is easy to make *ready for use* classes for other technologies. Unit of work class coordinating the work of all repositories in order to create single database context. The repository class is generic to avoid duplication of code. This class implements basic logic for working with database operations- CRUD. On the other hand with “*UnitOfWork*” we ensures that all repositories use the same database context.