ŽILINSKA UNIVERZITA

FAKULTA RIADENIA A INFORMATIKY

**Patrik Dolovski**

LIBRARY FOR AUTO-GENERATING ACTIONS WITH DATABASE ENTITIES

Project for course DESIGN PATTERNs

Žilina, 2018.

Table of contents

[1. Motivation 1](#_Toc532680812)

[2. Environment 1](#_Toc532680813)

[3. Concept 1](#_Toc532680814)

[4. Use-case 4](#_Toc532680815)

[5. Proof of concept 5](#_Toc532680816)

[6. Conclusion 8](#_Toc532680817)

1. Motivation

Everything started when I pitched the idea of fixing one of my old projects which I have done for Database Design course back on my university. It was a simple CRUD application working with Postgres database. The motivation behind this project is to create a library for C#, which could generate operations of our pick for any database entity. Due to the flexibility of the library it’s possible to use it with multiple database management systems. The library will build presentation of our choosing(Windows form, terminal, etc.).

1. Environment

The project is implemented using Microsoft Visual Studio 2017 on Windows 10 platform, using C# programming language.

.

1. Concept

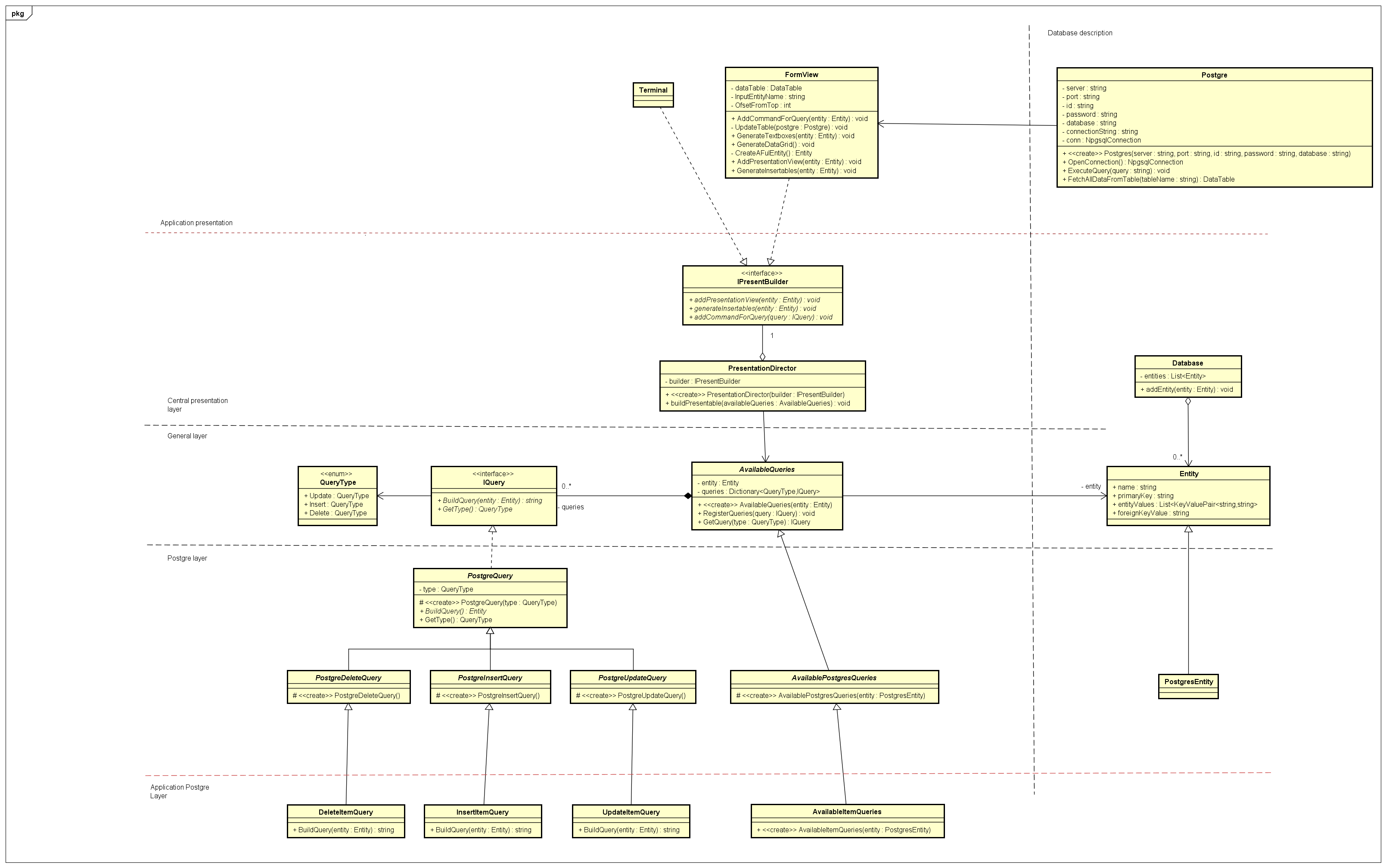


Figure 1 - Class diagram

The concept our library, as it was said before, is to create a library which will offer flexibility in presentation of different entities from different database management systems. Thus, our library consists of multiple layers which are designed using different design patterns.

Figure 1. shows class diagram with layers of our library including two extra application layers which are a part of the practical example of the library application.

Starting with the general layer and Postgres, we are introducing a Command design pattern which consists of an abstract class AvailableQueries, interface IQueries, abstract class PostgreQuery which serves as an adapter for getType() method and abstract classes PostgreInsertQuery, PostgreUpdateQuery and PostgreDeleteQuery serve as adapters for certain QueryTypes. AvailableQueries is capable registering queries for given entities and IQuery serves as an interface which when implemented offers different kinds of queries.

Moving to central presentation layer, PresentationDirector and IPresentBuilder are parts of the Builder design pattern with PresentationDirector executing the building itself and IPresentBuilder knowing the correct steps. In our case builder offers flexibility in creating multiple ways of presentation like Windows Forms or terminal.

Since our application communicates with database, in database description we defined entities and database. Entities can be inherited if the structure is not enough for other database management systems. The idea of the database is to have an object where every entity is stored and it could come handy for further changes in the library. In this class diagram part of database description is class Postgres. The idea is that user defines the connection to the database and the DBMS.

The two application layers(Application Postgre layer and Application presentation) that are not part of the library are depictions of one practical usage of the library. The application will build Windows Forms and generate controls, based on user input entities and database operations.

Figure 2. shows an example sequence diagram for generating an Insert command using Postgre DBMS and showing it to the user.

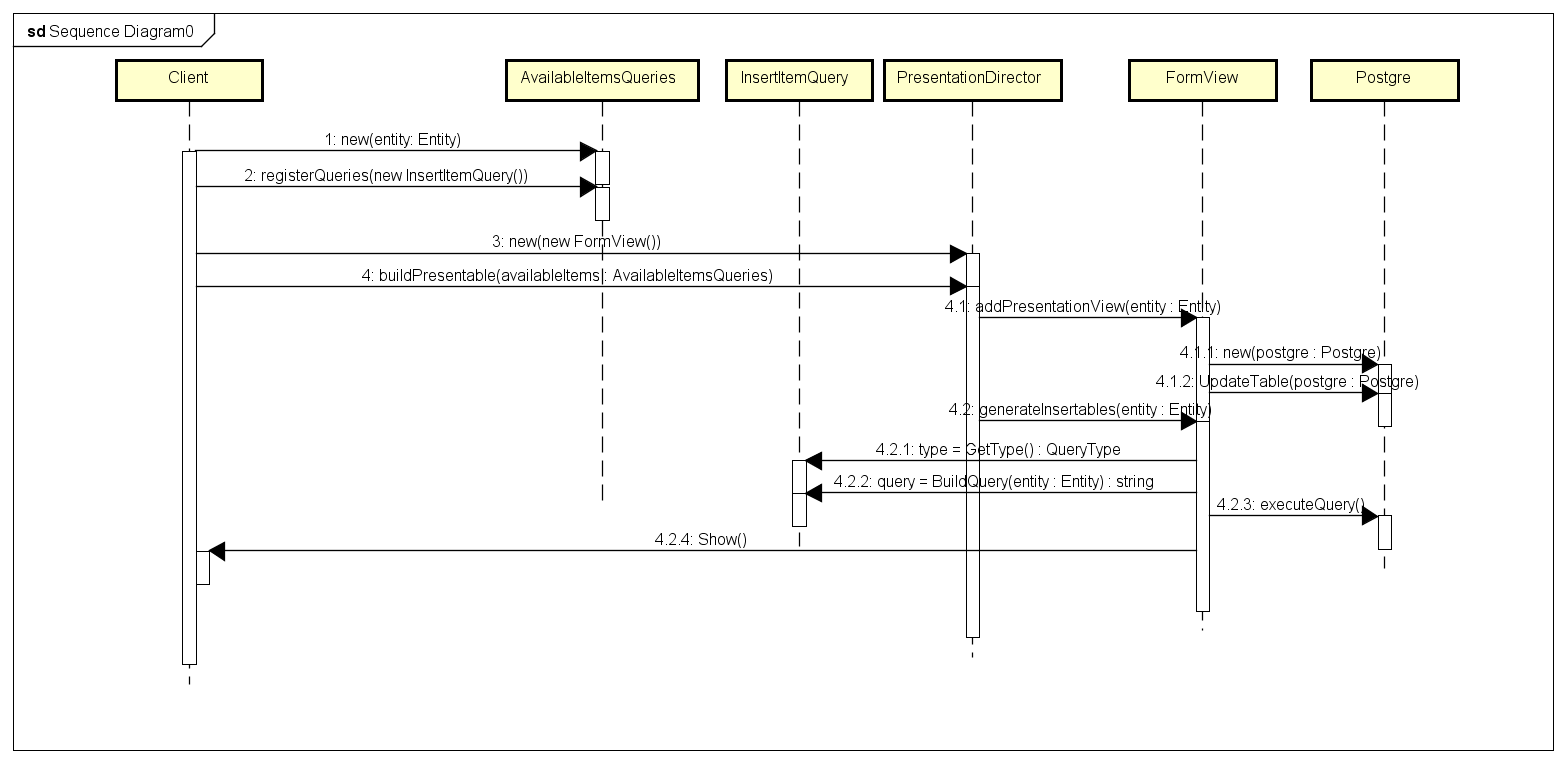


Figure 2 - Sequence diagram Insert example

1. Use-case

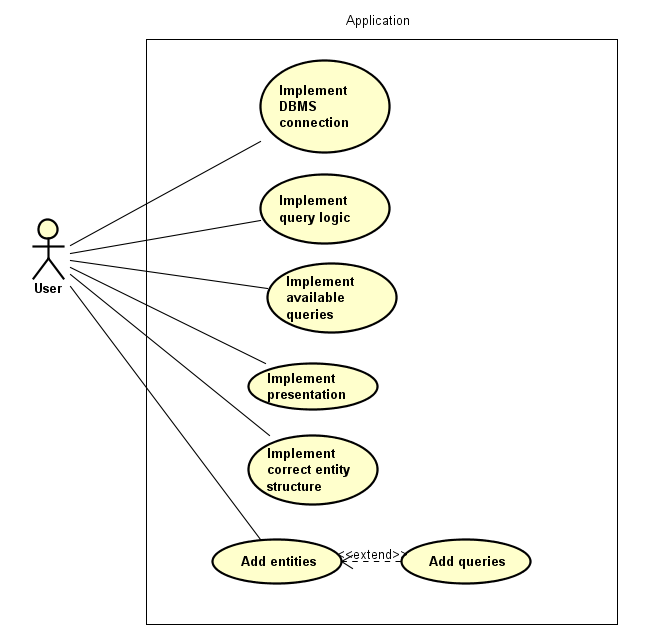


Figure 3 - Use-case diagram

Figure 3. shows a use-case diagram for the correct usage of library. For correct implementation user should implement connection with database management system, query logic for the same DBMS, implement available queries, implement some sort of presentation, implement changes to entity structure if needed and in the end add the entities with their queries.

1. Proof of concept

To prove that concept of the library is working we are going to try showing it on a simple application which will generate a Windows form based on the given entity and database operations. Application will dynamically generate controls like DataGridView to show the data, labels, textboxes and buttons which are going to do certain operations.

* 1. Implementing queries

While we are working with Postgre database, for each of the queries (Update, Insert, Delete) we will make a class which implements its correct adapted abstract class (PostgreUpdateQuery, PostgreInsertQuery, PostgreDeleteQuery). An example of Delete query can be seen on Figure 4.

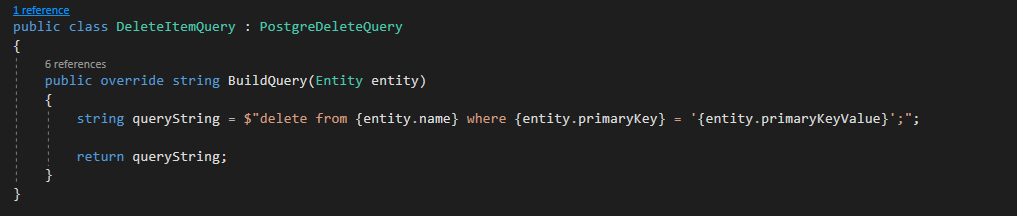


Figure 4 – DeleteItemQuery

* 1. Implementing available queries

Since we are using the same structure of an Entity/PostgreEntity like the one in library, there is no need of changing something. We are just going to create a class which implements adapted abstract class AvailablePostgresQueries.

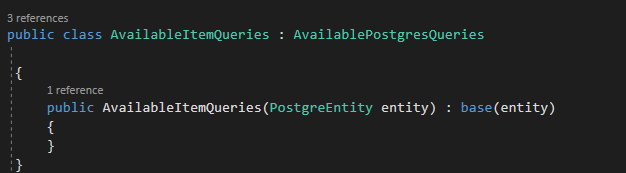


Figure 5 - AvailableItemQueries

* 1. Postgre class

Our application has a connection to the database,thus we made the class for database communication. Class can open a connection to the database, execute a certain query and fetch all data from a specific entity.

* 1. Presentation form

As it was said before, our presentation will be a Windows form. Empty Windows form class will implement *IPresentBuilder* interface. Overridden method addPresentationView will dynamically generate a *DataGridView* control. Overridden method *generateInsertables()* will dynamically generate labels and textboxes. Last overridden method *addCommandForQuery()* will generate buttons with lambda function for executing database operations.

* 1. Main form

For testing purposes, main form will have only 1 button, when pressed, will add our entity, available queries for the entity and then execute presentation build. An example can be seen on Figure 6.

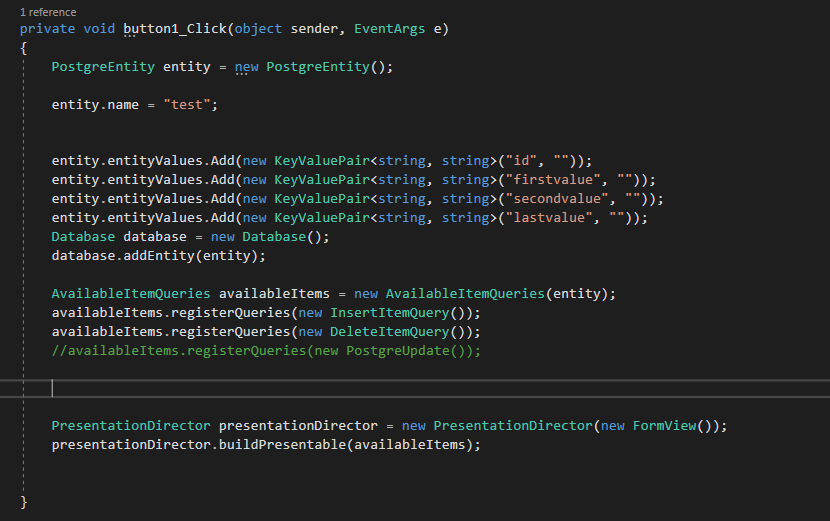


Figure 6 - Main form button click

* 1. Running the test application

When the application starts, we are greeted with a screen which has only 1 button. As can be seen on Figure 7.

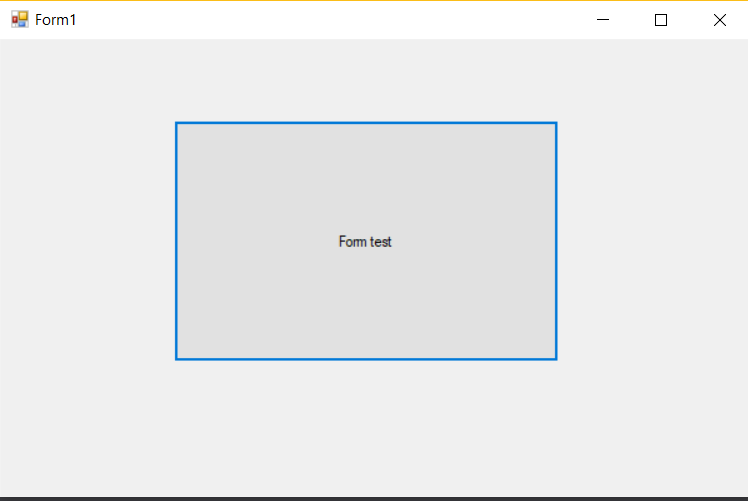


Figure 7 - Main application screen

After clicking on the button, the application will generate a new form which will present values from the database and it will generate controls which are related to the given entity.

Since in the code on Figure 6. we only picked Insert and Delete controls, the application will generate form as it is shown on Figure 8.

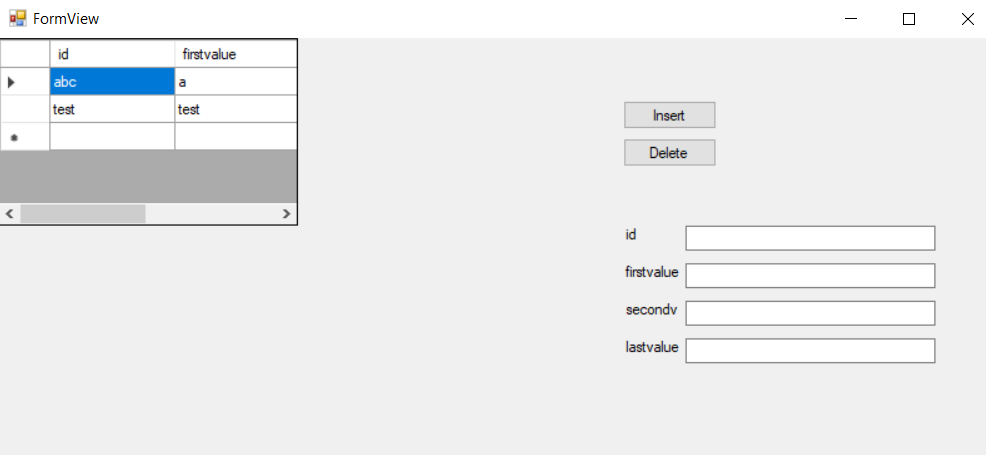


Figure 8 - Dynamically generated form

1. Conclusion

After taking this course and making this project it is safe to say design patterns expanded my horizons. In the making of this project, I slowly realized that in the past I was focusing too much on the “bottom” layer of applications, hence a lot of applications resulted in being inflexible. Putting it simply, they were only good for one type of the problem. Introduction of design patterns shed some light on how flexible applications are made.

Using this library, the aforementioned Database Design application could be rewritten with the fact that changing a DBMS or the presentation would be a case of changing only a couple of classes.