**EUROPEAN UNIVERSITY OF LEFKE**

FACULTY OF ENGINEERING

Graduation Project 2

# Real Estate Automation

**Recep Pazarlı**

**180443**

**When we look at real estate companies today, we see that they usually keep the data they have manually either in excel or similar applications. This method is both slow and time consuming. The main idea and purpose of my project is to collect this system under a single roof as organized as possible and to prepare an application that is easy, usable and equipped with new technologies. One of the goals of this project is to leave a mark in the real estate world and to ensure that it is an application that can be developed against all kinds of innovations.**

**Supervisor**

Asst. Prof. Dr. Ferhun Yorgancıoğlu

03.01.2024

# Table Of Contents

# 1.Introduction

## 1.1 Problem definition

When we look at today's real estate offices, we can see that there is a complexity and disorganization. For example, while keeping the plots, fields, houses and shops in their hands as data, they keep records on paper or excel-like applications. I have observed many times that this registration process is both tiring and time consuming. Thanks to this project, these problems will be minimized to the lowest level, saving time for the employee and the boss in every subject, it will be a more useful system and efficiency will increase to the highest levels

Problems in the current system:

- Recording the immovables in the book leads to disorganization and complexity.
- It is very difficult to keep track of land, fields and houses, and in any case, it is very likely that you will get lost in the files.
- Tracking other expenditures and similar transactions is also difficult and time consuming.
- It is time-consuming to find some incoming customers or the customer's information is lost.

## 1.2 Goals

The aim of this project is to digitize data to reduce complexity and increase concentration. This will make data more organized and secure. At the same time, work tracking will become easier, productivity and speed will increase. Employees will be able to add information to the system more easily and the boss will be able to view it. Thanks to this project, transactions on the accounting side will also be easy and income and expense tracking will be very fast. In general, the project will be prepared in a way equipped with new technologies and new additions, subtractions and updates can be made in a developable way.

- When it is necessary to access old data (Plots, Fields, Houses and Shops), it provides easy and fast access without getting lost in the documents.
- It allows the boss to access the data easily and quickly.
- Provides quick access to customer information when requested.
- Optionally, it presents the data to the user as a report.

# 2. Literature Survey

Existing systems also used manual processes for tasks such as Property management and customer support. This can lead to problems such as paper records or irregularly piling up invoices to keep track of real estate. This project, on the other hand, facilitates these processes in the shortest time possible with its efficient, fast, safe and easy use. When we look at the applications used today, although there are advanced applications in general, we aim to collect the deficiencies in all of them and reveal a unity with this application. In today's applications, it can take time for the user to understand the application, and this causes these applications not to be used. In this project, usability is at a high level and an easy and simple use is aimed for the user. The user will happily use an application prepared in this way, which will create trust in terms of the application. Keeping up with the developing technologies of the application creates an attraction for the user. It also includes the wishes of the user, which develops with the developing technologies, and special innovations can be brought to each user. In this way, a program prepared according to the user's request will see a lot of demand in the real estate sector. As someone who has been in the real estate sector for more than 10 years, I have observed these shortcomings well, and considering these shortcomings, I am developing a useful application for this sector.

Features 1:

Usability plays a very important role in marketing the app. An application that is simple and easy to use for customers will be effective for users.

Features 2 :

Adding, removing and updating the features that may vary from user to user according to the user's wishes. (For example: integration of a newly released system into the program.)

Features 3 :

In the system, it is a desired feature for this user to allow the user to follow up their income and expenses, and they will be able to both track their data and handle accounting transactions easily.

Features 4 :

In the project, a design that has been carefully chosen for the user in terms of design and that gives pleasure to the users without tiring the eyes has been realized.

Features 5: It provides more convenience to the user in terms of cost and appeals to users from all walks of life. Optionally, there is the option of renting or purchasing.

# 3. Background Information

You must write which language, tools you use, and why. If you have any hardware you should write them. You can find an example Below do not forget to remove examples !

## 3.1 Required software

1. **C# Form Application (.NET Framework):** I chose this language because I wanted to design a good and quality project and because it is suitable for Windows application format.

2. **Visual Studio:** Editor for coding.

3. **SQL server:** To hold the database.

4. **DevExpress:** A tool app to design tools in my project more modern.

## 3.2 Other software

5. **Adobe Illustrator:** Design icons, logo, etc. for.

6. **Adobe XD:** For the design poster.

7. **Git:** Used for repository.

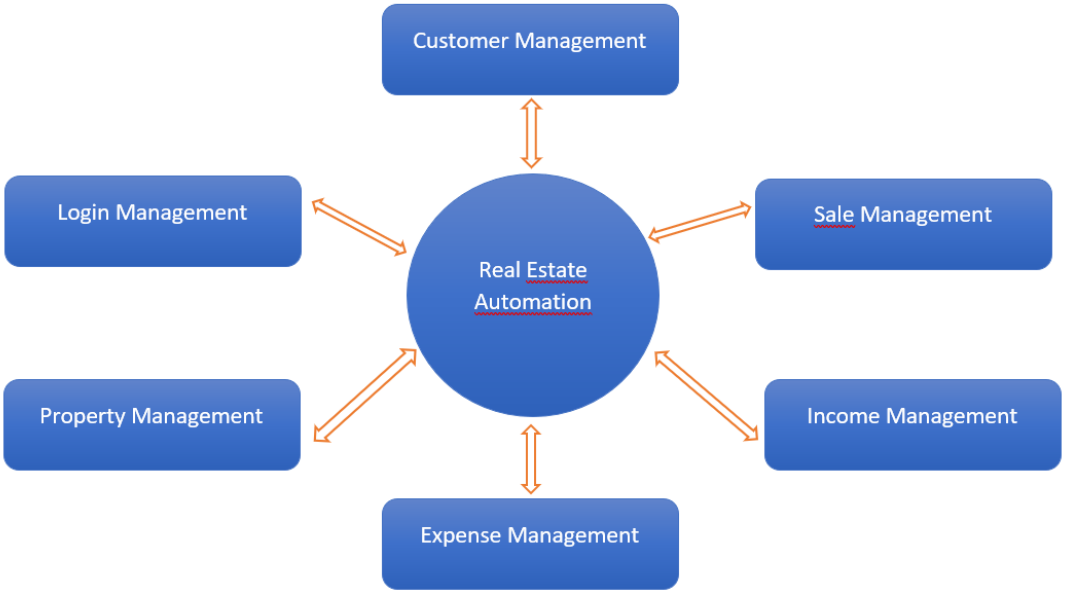
# 4. Design Documents

## 4.1 Use Case Diagram

## 4.2 ER Diagram

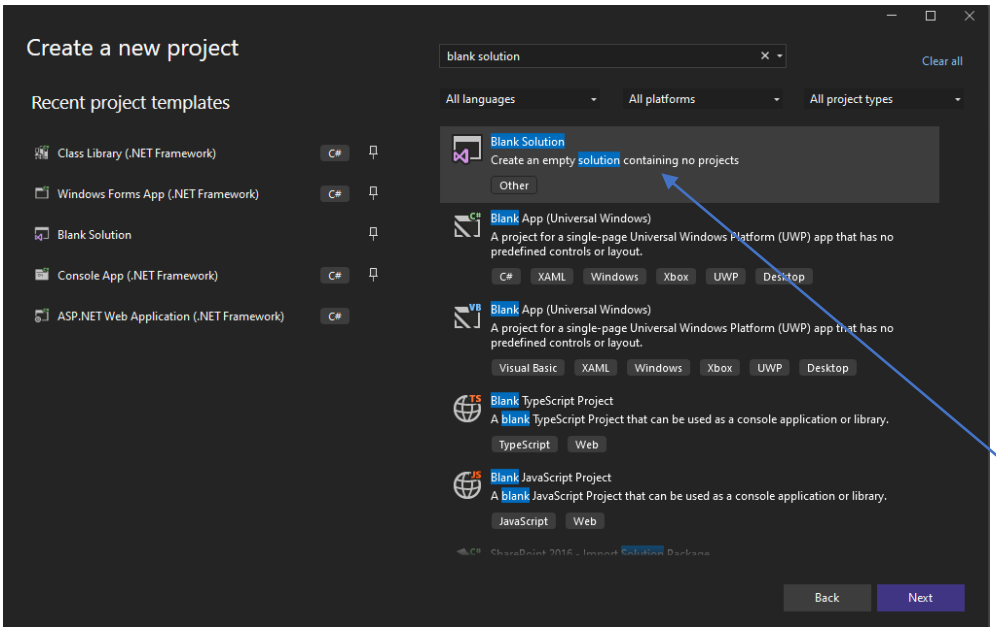

,

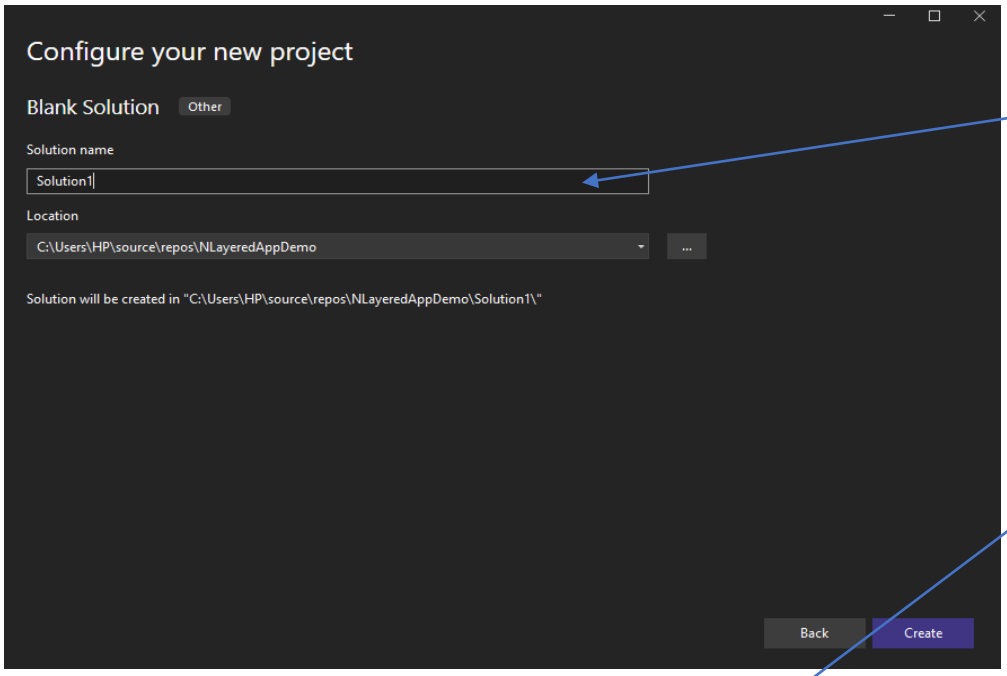
## 4.3 Context Diagram

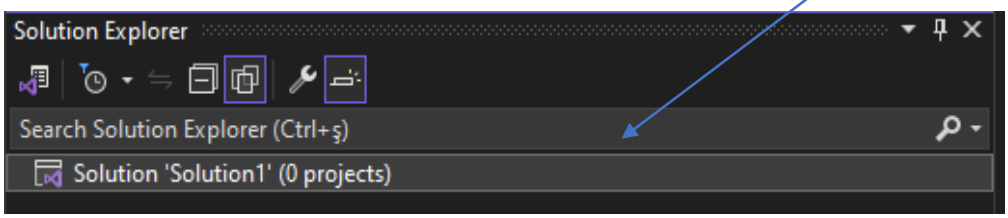


# 5. Methodology

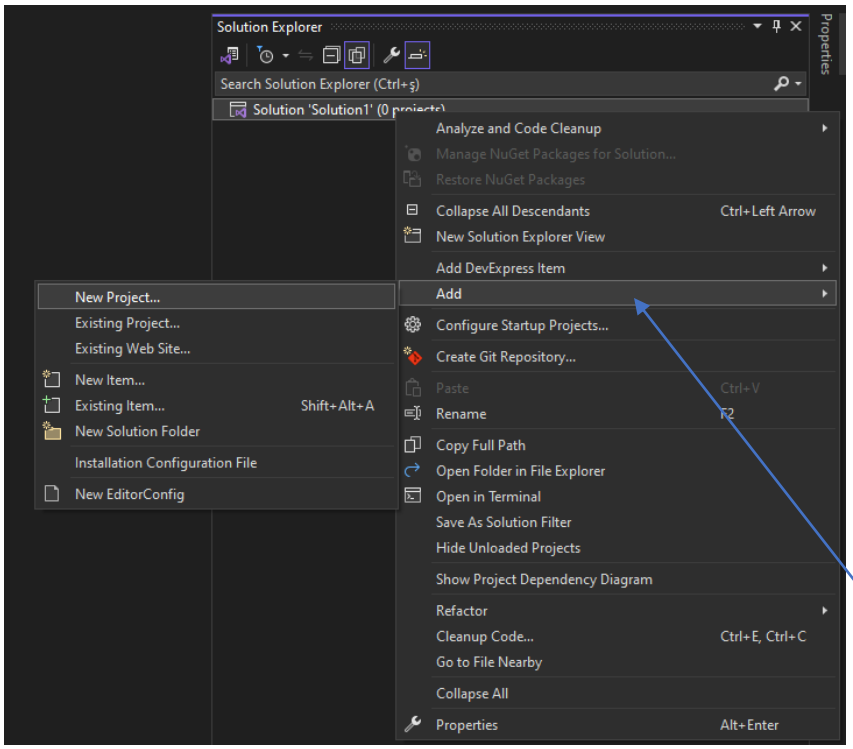
## 5.1 Creation of the project

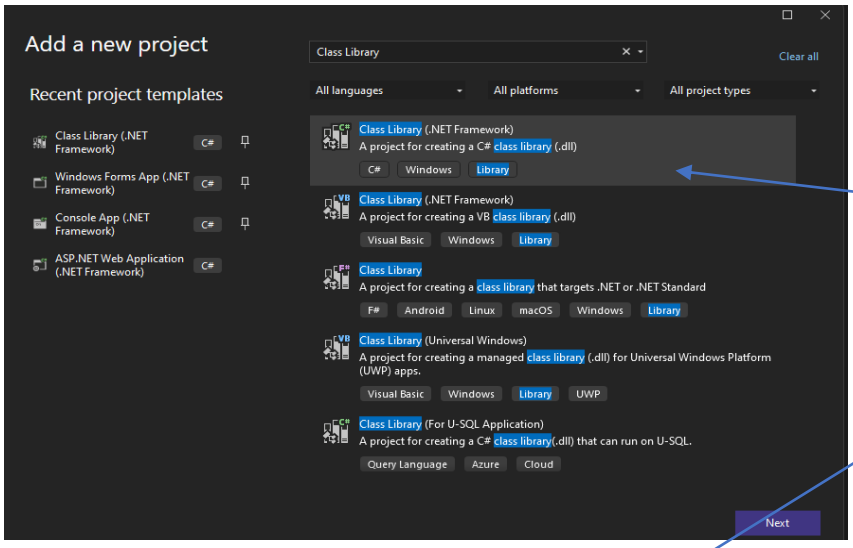


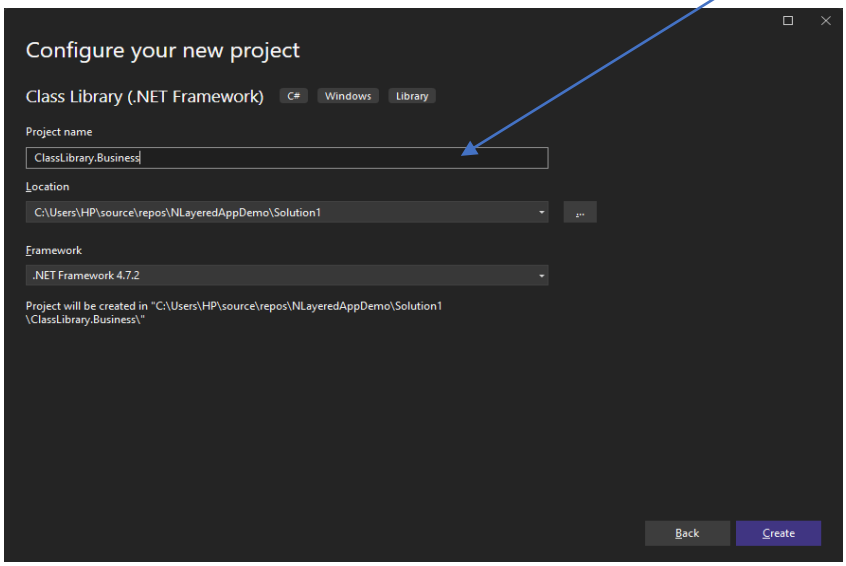



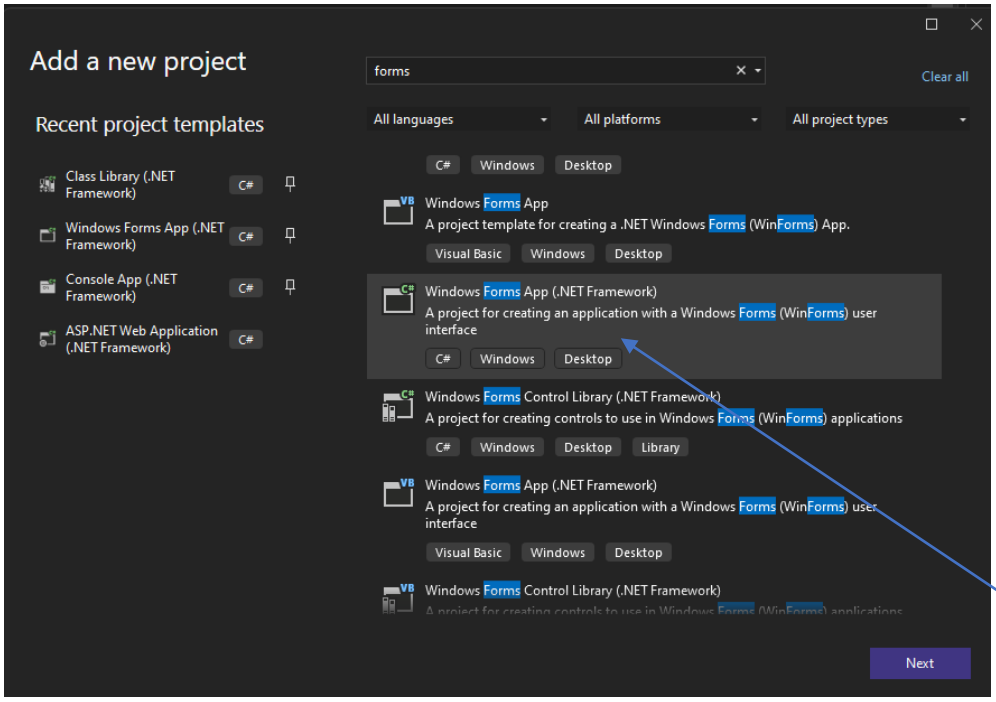

First, I create a "Blank Solution" and start the program by giving the name of my project. The reason why I start it as a Blank solution is that I will add a "Class Library" in it because I use "Layered Architecture" in my project, so I will start with an empty project and prepare the layered construction into it.



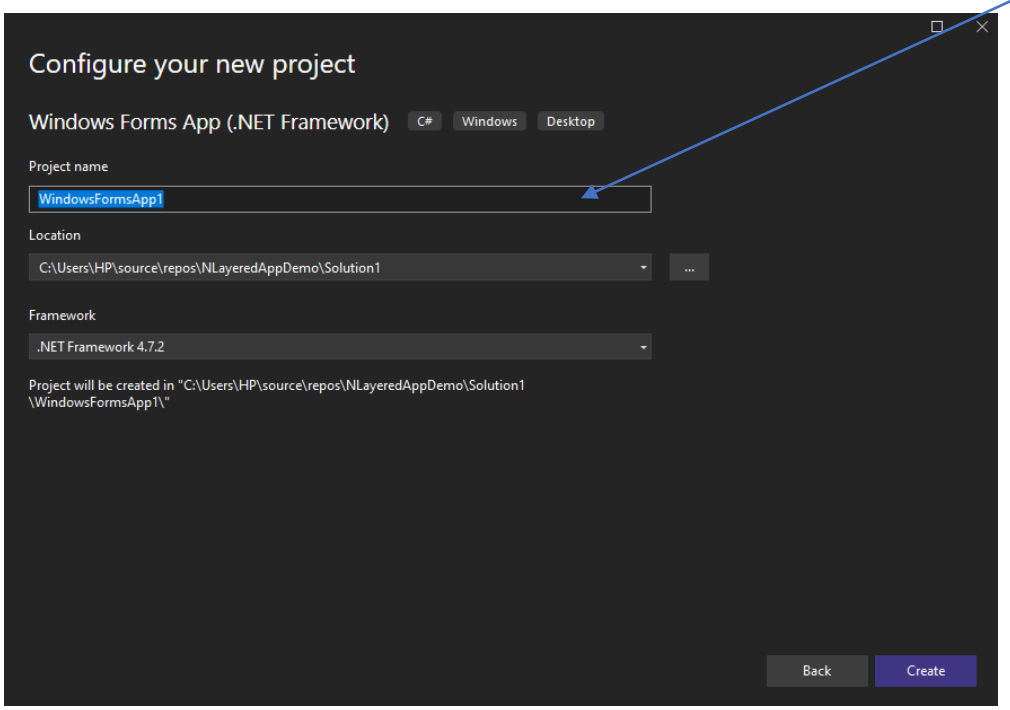



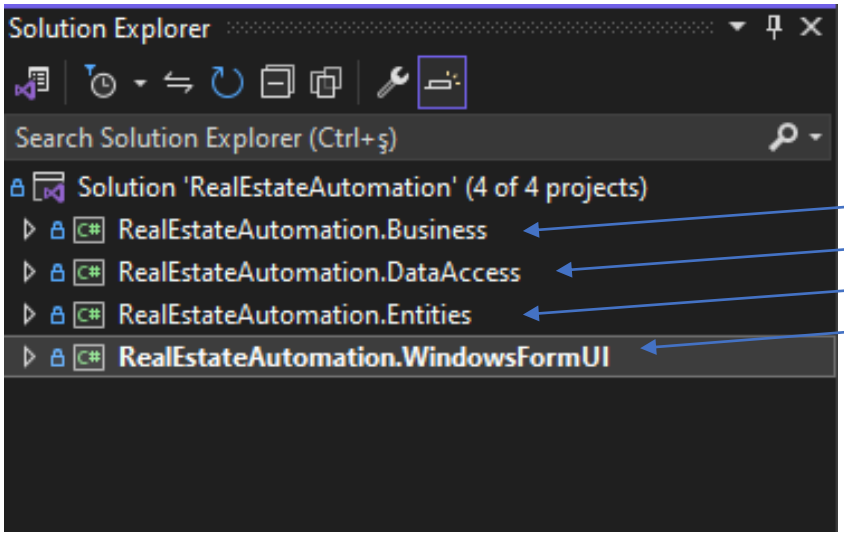




I created the layers of my project by defining them in this way.




After defining all the "Class Library", I create a new "Windows Form App (.NET Framework)" project and name it.





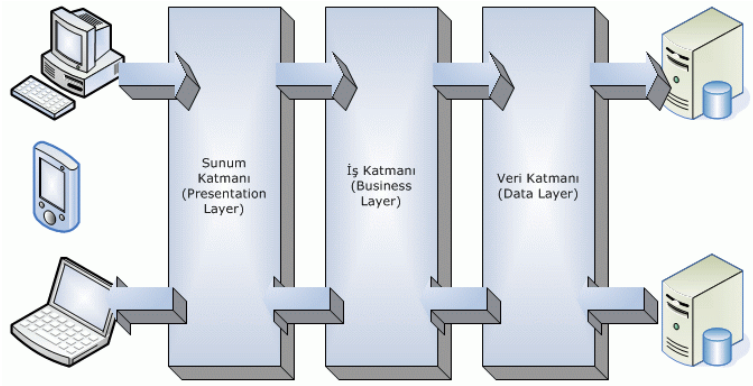

Finally, I started my project in this way and created the necessary "Class Library" and "Windows Form Apps" in them. The final image is as follows

- ✓ **Entities Layer :** In this layer I defined my real Entities and created the necessary classes.

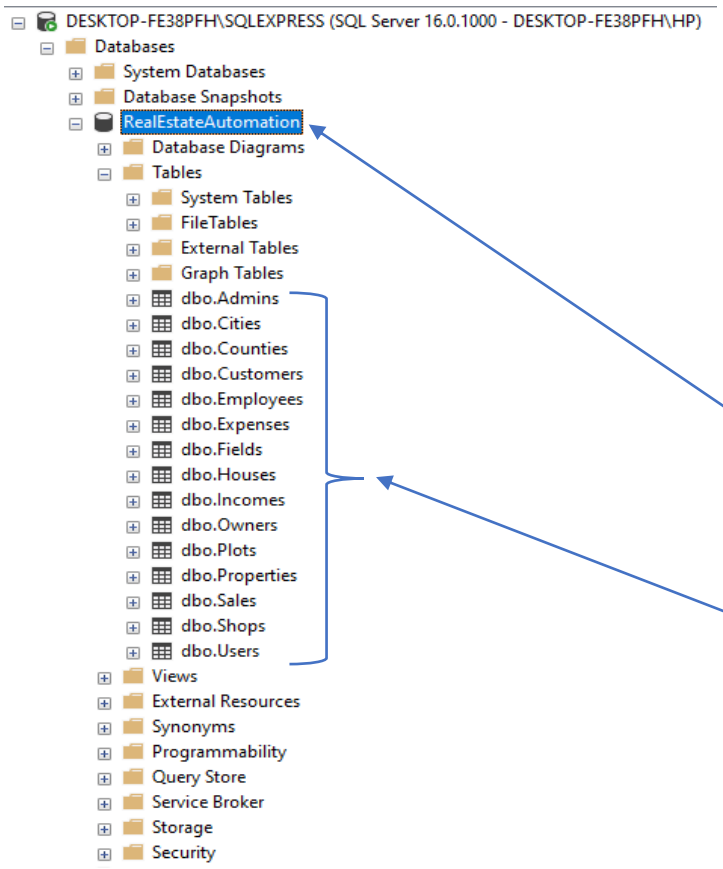
- ✓ **Data Access Layer :** In this layer, I prepared the database operations. I used it for the Create, Read, Update and Delete operations required for the data.

- ✓ **Presentation Layer (Windows Form User Interface) :** This is the layer where communication with the user takes place. I prepared it to show the data to the user and to receive the data entered by the user. I transferred the incoming data to Data Access with the Business layer and performed the operations.

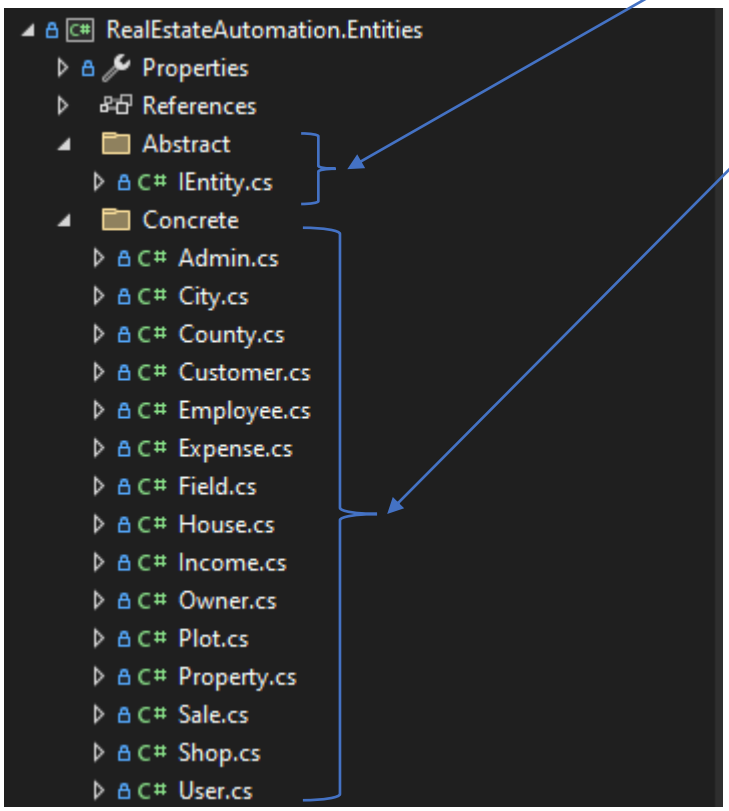
- ✓ **Business Layer** : In this layer, I have prepared processes for the service, manager and validation of my project. I use the business layer as a connection between the Data Access layer and the Presentation layer. I receive data from the Data Access layer, process it in the business layer and transfer it to the Presentation layer by making the necessary checks. I also receive data from the Presentation layer, make the necessary checks and transfer it to the Data Access layer. In this way, I provide the necessary validation processes and data access controls for my project.



## 5.2 Entities Layer




I created "Abstract" and "Concrete" files. I created "Interface" definitions of abstract objects in my Abstract file and "Class" definitions of concrete objects in my Concrete file.



### 5.2.1Abstract && Concrete

1. IEntity

```
17 references
public interface IEntity
{
    99+ references
    int Id { get; set; }
}
```

I created an interface called "IEntity". My goal in doing this is to implement all my Entities in this interface and give them a signature. I will use it when writing generic constraints for the generic structures I will make later. I only defined the id field for it. I could have done without this, my goal is to give what is common to all of them, so I used this, as I said, I could have written nothing at all, my goal is to use it as a signature.

2. Admin

```
15 references
public class Admin : IEntity
{
    4 references
    public int Id { get; set; }
    3 references
    public int UserId { get; set; }
    6 references
    public string NationalityId { get; set; }
    4 references
    public string FirstName { get; set; }
    4 references
    public string LastName { get; set; }
    4 references
    public string Phone { get; set; }
    3 references
    public string Email { get; set; }
    3 references
    public string City { get; set; }
    3 references
    public string County { get; set; }
    3 references
    public string Address { get; set; }
    4 references
    public bool DeleteFlag { get; set; }

}
```

As I mentioned above, I implemented it from the IEntity interface. I created the admin object by defining the special properties of the admin field. Although the Id field does not appear here, I defined it as a primary key. I used UserId to control the information of the admin in the User table. I added the general features in this way and completed my admin object.

3. City

```
5 references
public class City : IEntity
{
    26 references
    public int Id { get; set; }
    24 references
    public string CityName { get; set; }
}
```

I defined my City and County fields in this way. I pulled the data in these classes by adding them with ready-made sql codes from the internet in the database. I prepared these fields for the user to pull data from lookupedit where City and County are required and save them in other tables.

4. County

```
5 references
public class County :IEntity
{
    26 references
    public int Id { get; set; }
    24 references
    public string CountyName { get; set; }
    1 reference
    public int  CityId { get; set; }
}
```

For the data of my City and County fields that I mentioned above, I added with a sql code as in the example below. Although there was a mistake in their nomenclature at first, I made the correction afterwards.

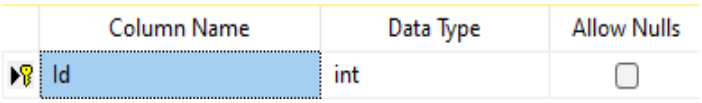
```
CREATE TABLE Cities
(
  Id INT PRIMARY KEY IDENTITY(1, 1),
  CityName NVARCHAR(255) NULL
);

--Iller tablosuna veri ekleme

INSERT INTO Cities
(
  CityName
)
VALUES
('ADANA'),
('ADIYAMAN'),
('AFYON'),
```

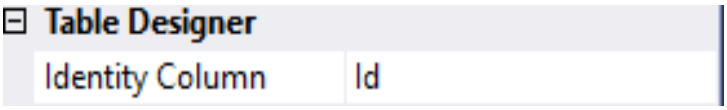
```
CREATE TABLE ilceler
(
    id INT PRIMARY KEY IDENTITY(1, 1),
    ilceadi [NVARCHAR](255) NOT NULL,
    sehirid INT NOT NULL
);

INSERT INTO ilceler(ilceadi, sehirid) VALUES
('ALADAĞ',1),
('CEYHAN',1),
('ÇUKUROVA',1),
('FEKE',1),
('İMAMOĞLU',1),
```

5. Customer

```
public class Customer : IEntity
{

    public int Id { get; set; }

    public string NationalityId { get; set; }

    public string FirstName { get; set; }

    public string LastName { get; set; }
    - references
    public string Phone { get; set; }
    - references
    public string County { get; set; }
    - references
    public string City { get; set; }
    - references
    public string Address { get; set; }
    3 references
    public string Description { get; set; }
    4 references
    public bool DeleteFlag { get; set; }
}
```

Although it does not appear here in the Customer field, I defined my Id field as the primary key. I defined the customer's own special features. In this way, I completed my customer field.




In all my tables, I defined my id fields as primary key and selected all id fields as automatic identity columns. In this way, I took under control that they occur automatically in the background when adding data and that they are unique.

6. Employee

```
public class Employee : IEntity
{

    public int Id { get; set; }
    3 references
    public int UserId { get; set; }
    6 references
    public string NationalityId { get; set; }
    4 references
    public string FirstName { get; set; }
    4 references
    public string LastName { get; set; }
    4 references
    public string Phone { get; set; }
    3 references
    public string Email { get; set; }
    3 references
    public string City { get; set; }
    3 references
    public string County { get; set; }
    3 references
    public string Address { get; set; }
    4 references
    public bool DeleteFlag { get; set; }

}
```

I defined special properties in my Employee class. I used UserId as a foreign key from the User table. In this way, I establish a relationship between the User table and the Employee table.

7. Expense

```
13 references
public class Expense : IEntity
{
    4 references
    public int Id { get; set; }
    5 references
    public DateTime? ExpenseDate { get; set; }
    6 references
    public decimal ExpensePrice { get; set; }
    4 references
    public string Description { get; set; }
    5 references
    public bool DeleteFlag { get; set; }
}
```

In this class, I defined the properties of the user required for expense tracking. The properties of my class are as in the figure on the side

8. Field

```
25 references
public class Field : IEntity
{
    19 references
    public int Id { get; set; }
    20 references
    public int PropertyId { get; set; }
    20 references
    public int OwnerId { get; set; }
    18 references
    public decimal Area { get; set; }
    17 references
    public string Pafta { get; set; }
    17 references
    public int City { get; set; }
    17 references
    public int County { get; set; }
    16 references
    public string Address { get; set; }
    18 references
    public decimal Price { get; set; }
    16 references
    public string Description { get; set; }
    14 references
    public bool Sold { get; set; }
    17 references
    public bool DeleteFlag { get; set; }


}
```

This is my Field class. I defined the necessary special field properties. In the PropertyId property, I defined a property id field for all fields and established a relationship between them.

9. House

```
22 references
public class House : IEntity
{
    18 references
    public int Id { get; set; }
    19 references
    public int PropertyId { get; set; }
    18 references
    public int OwnerId { get; set; }
    18 references
    public decimal Area { get; set; }
    17 references
    public string HouseType { get; set; }
    17 references
    public int City { get; set; }
    17 references
    public int County { get; set; }
    16 references
    public string Address { get; set; }
    18 references
    public decimal Price { get; set; }
    16 references
    public string Description { get; set; }
    14 references
    public bool Sold { get; set; }
    17 references
    public bool DeleteFlag { get; set; }
}
```

This is my House class. I defined the necessary special House properties. In the PropertyId property, I defined a property id field for all Houses and established a relationship between them.

10. Income

```
public class Income : IEntity
{

    public int Id { get; set; }

    public DateTime? IncomeDate  { get; set; }

    public decimal IncomePrice { get; set; }

    public string Description { get; set; }

    public bool DeleteFlag { get; set; }
}
```

In this class, I defined the user properties required for income tracking. The properties of my class are as shown in the figure below

11. Owner

```
15 references
public class Owner : IEntity
{
    28 references
    public int Id { get; set; }
    6 references
    public string NationalityId { get; set; }
    24 references
    public string FirstName { get; set; }
    4 references
    public string LastName { get; set; }
    4 references
    public string Phone { get; set; }
    4 references
    public bool DeleteFlag { get; set; }
}
```

In this class, I defined the user properties required for income tracking. The properties of my class are as shown in the figure below

12. Plot

```
public class Plot : IEntity
{

    public int Id { get; set; }

    public int PropertyId { get; set; }

    public int OwnerId { get; set; }

    public decimal Area { get; set; }

    public string Ada { get; set; }

    public string Pafta { get; set; }

    public int City { get; set; }
    - references
    public int County { get; set; }
    16 references
    public string Address { get; set; }
    18 references
    public decimal Price { get; set; }
    16 references
    public string Description { get; set; }
    14 references
    public bool Sold { get; set; }
    17 references
    public bool DeleteFlag { get; set; }

}
```

This is my Plot class. I defined the necessary special Plot properties. In the PropertyId property, I defined a property id field for all Plots and established a relationship between them.

13. Property

```
– references
public class Property : IEntity
{
    – references
    public int Id { get; set; }
    12 references
    public int ReferenceId { get; set; }
    20 references
    public string PropertyType { get; set; }
    13 references
    public bool DeleteFlag { get; set; }
}
```

My Property class is as shown in the next figure. The ReferenceId field takes from the ids of my Field, House, Plot and Shop fields, so I make associations between them. I keep which type with my Property type field.

14. Sale

```
22 references
public class Sale : IEntity
{
    14 references
    public int Id { get; set; }
    20 references
    public int OwnerId { get; set; }
    21 references
    public int CustomerId { get; set; }
    16 references
    public string SalePropertyType { get; set; }
    21 references
    public int SalePropertyId { get; set; }
    16 references
    public DateTime SaleDate { get; set; }
    18 references
    public decimal SalePrice { get; set; }
    17 references
    public bool DeleteFlag { get; set; }
}
```

This is my sale class and I have defined the special properties required for sale. I associate with OwnerId, CustomerId and PropertyId and I get all the necessary properties in my sales table. My class is as in the table on the right.

15. Shop

```
20 references
public class Shop : IEntity
{
    19 references
    public int Id { get; set; }
    19 references
    public int PropertyId { get; set; }
    17 references
    public int OwnerId { get; set; }
    18 references
    public decimal Area { get; set; }
    17 references
    public int City { get; set; }
    17 references
    public int County { get; set; }
    16 references
    public string Address { get; set; }
    18 references
    public decimal Price { get; set; }
    16 references
    public string Description { get; set; }
    14 references
    public bool Sold { get; set; }
    17 references
    public bool DeleteFlag { get; set; }
}
```

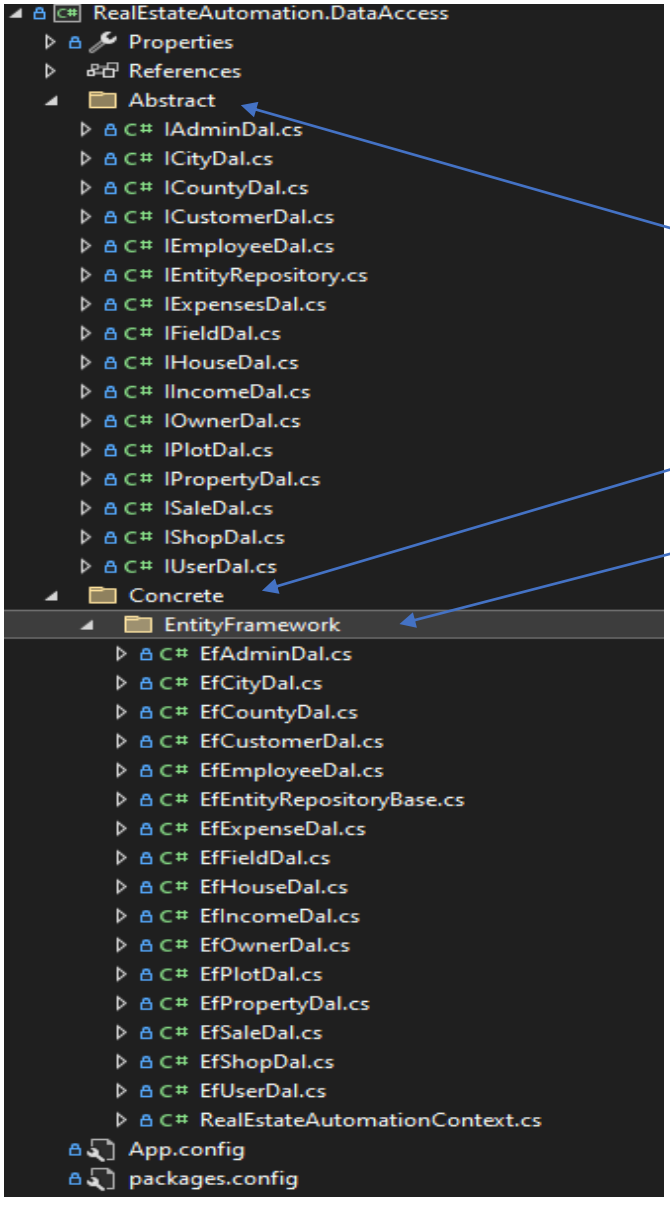
This is my Shop class. I defined the necessary special Shop properties. In the PropertyId property, I defined a property id field for all Shops and established a relationship between them.

16. User

```
public class User : IEntity
{

    public int Id { get; set; }
    13 references
    public string UserName { get; set; }
    14 references
    public string UserPassword { get; set; }
    9 references
    public int UserAuthorization { get; set; }
    6 references
    public bool DeleteFlag { get; set; }


}
```

This class is as shown in the next figure. I have prepared the special properties required for the user. I have prepared these properties to keep information about the entries and exits to the programme.

## 5.3 Data Access Layer




The Data Acces layer is as shown in the figure on the right. I made some additions for the necessary database operations in it. In the Abstract part, I made abstract (Interface) definitions. In the Concrete part, I made concrete (Class) definitions. In my Concrete file, I made it under the Entity Framework file name. Because, the operations in this file are the operations I prepared for data usage from SQL. If new databases are added in this way, I can easily add a folder to my system and prepare operations for it.

### 5.3.1Abstract && Concrete

1. IEntityRepository

This interface is a basic asset repository construction. My aim here is to prepare the Repo structure in an interface field and implement it in my repo class. There are my necessary methods that I will need in it. As I mentioned before, I prepared a generic structure and checked this structure. I wanted to avoid the same process repetition. I tried to write code in accordance with SOLID principles, although not exactly everywhere.

```
16 references
public interface IEntityRepository<T> where T : class, IEntity, new()
{
    18 references
    List<T> GetAll(Expression<Func<T, bool>> filter = null);
    1 reference
    T GetById(Expression<Func<T, bool>> filter);
    14 references
    void Add(T entity);
    16 references
    void Update(T entity);
    3 references
    void Delete(T entity);

    7 references
    T GetLastAddedEntity(Expression<Func<T, bool>> filter = null);
}
```

In this way, I built all my functions in generic structure according to the structure given while implementing the interface

I used some filters while defining my functions. I can use filters according to my needs. I can use filters with or without filters if I want.

2. Ef Entity Repository Base

In this class, I implemented and prepared the Repository Base class operations by implementing IEntityRepository. I designed this structure in a generic structure. I will use these structures in all my Data Acces operations. Since they are all the same operations, I made such a use. I made the necessary restrictions in the generic structure and prepared my operations accordingly.

```
15 references
public class EfEntityRepositoryBase<TEntity, TContext> : IEntityRepository<TEntity>
    where TEntity : class, IEntity, new()
    where TContext : DbContext, new()
{
```

My GettAll method is in a public structure as shown below and is open to access from anywhere. If desired, filtering can be done with the Expression in the parameter. In the inner part, I continue by defining the Context construction that I took as generic above with using. The reason I use Using is that the garbage collector deletes the garbage collector directly from the ram when I'm done with this part. In this way, I made the programme spend less from ram. If I have a filter, I have completed this method by making it return according to it.

```
18 references
public List<TEntity> GetAll(Expression<Func<TEntity, bool>> filter = null)
{
    using (TContext context = new TContext())
    {
        return filter == null ? context.Set<TEntity>().ToList() :
            context.Set<TEntity>().Where(filter).ToList();
    }
}
```

My GetById method has all operations like GetAll and this method returns data according to a value.

```
1 reference
public TEntity GetById(Expression<Func<TEntity, bool>> filter)
{
    using (TContext context = new TContext())
    {
        return context.Set<TEntity>().SingleOrDefault(filter);
    }
}
```

```
14 references
public void Add(TEntity entity)
{
    using (TContext context = new TContext())
    {
        var addedEntity = context.Entry(entity);
        addedEntity.State = EntityState.Added;
        context.SaveChanges();
    }
}
```

By logging in my add method according to the entity given when the class is inherited. I set the status of those coming from that entity as added and add it thanks to the entity framework and finally I save the change in my context.

```
16 references
public void Update(TEntity entity)
{
    using (TContext context = new TContext())
    {
        var updatedEntity = context.Entry(entity);
        updatedEntity.State = EntityState.Modified;
        context.SaveChanges();
    }
}
```

My update method is the same as Add and I take the necessary data entered in it and mark its status as update and in this way I update the data and save the changes.

```
3 references
public void Delete(TEntity entity)
{
    using (TContext context = new TContext())
    {
        var deletedEntity = context.Entry(entity);
        deletedEntity.State = EntityState.Deleted;
        context.SaveChanges();
    }
}
```

In my Delete method, like the others, I delete the data according to the entity entered here and save the changes. Although I don't use this method much, I used this method to protect against data leaks in some places and prevented data complexity.

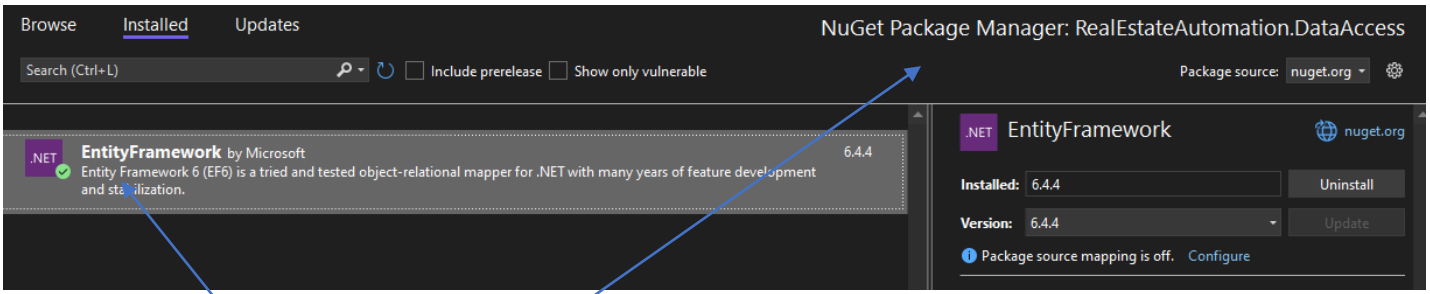
My GetLastAddedEntity method is a method that works for me when I make two additions in some tables, I defined it here in my architecture to use it if I need it again. It returns me the last added entity according to the value given to the table and allows me to capture its id.

```
7 references
public TEntity GetLastAddedEntity(Expression<Func<TEntity, bool>> filter = null)
{
    using (TContext context = new TContext())
    {
        return context.Set<TEntity>().OrderByDescending(x :TEntity => x.Id).FirstOrDefault();
    }
}
```

3. Real Estate Automation Context

```
65 references
public class RealEstateAutomationContext : DbContext
{
    0 references
    public DbSet<Admin> Admins { get; set; }
    24 references
    public DbSet<City> Cities { get; set; }
    24 references
    public DbSet<County> Counties { get; set; }
    4 references
    public DbSet<Customer> Customers { get; set; }
    0 references
    public DbSet<Employee> Employees { get; set; }
    1 reference
    public DbSet<Expense> Expenses { get; set; }
    6 references
    public DbSet<Field> Fields { get; set; }
    6 references
    public DbSet<House> Houses { get; set; }
    1 reference
    public DbSet<Income> Incomes { get; set; }
    24 references
    public DbSet<Owner> Owners { get; set; }
    6 references
    public DbSet<Plot> Plots { get; set; }
    24 references
    public DbSet<Property> Properties { get; set; }
    4 references
    public DbSet<Sale> Sales { get; set; }
    6 references
    public DbSet<Shop> Shops { get; set; }
    0 references
    public DbSet<User> Users { get; set; }

}
```

This is the context construction I wanted generically above. Here, thanks to the Entity framework, I define a context for myself and make this data available. I inherit from the DbContext structure and use DbSet. In my DbSet structure, I write the name of my entity and tell what its name will be in the database and in this way I communicate between them. I made the naming according to the EntityFramework rules and prepared it, and the entities I made as singular in my project are registered in my database as plural. Entity framework automatically searches in the background in this way and I made it work faster by following the rule.




Before I do all the things I mentioned above, I need to integrate the Entity framework into my project. After this process, I can easily prepare all my data operations. I completed the integration process by right clicking on my DataAcces layer and searching for the entity framework through the manage nuget package and adding it to my project from there.

4. IAdminDal && EfAdminDal

```
4 references
public interface IAdminDal : IEntityRepository<Admin>
{

}
```

I defined an interface and class for each Entity object like this. I implemented the interfaces IEntityRepository and added the features in it to my own interface. The reason for this implementation is that I want to use the common features there. Instead of writing it everywhere again, I designed my layer in this way. If I want to add special methods only to Entities, I can add them to these fields and prepare the transactions. In my class structure, I inherited from EfRepositoryBase, told which structures I would work on, and took my common features to myself. Thanks to these signatures, I avoided rewriting common features and repeating code each time.

```
1 reference
public class EfAdminDal : EfEntityRepositoryBase<Admin, RealEstateAutomationContext>, IAdminDal
{

}
```

For naming, I named the interfaces using "I" to distinguish them. The reason I say "branch" is to distinguish whether it is a class or interface from the Data Access Layer. I applied these naming settings to all of them and did the operations I mentioned above for all my other entity objects, you can find the images of all of them below.

5. ICityDal && EfCityDal

```
4 references
public interface ICityDal : IEntityRepository<City>
{

}
```

```
1 reference
public class EfCityDal : EfEntityRepositoryBase<City, RealEstateAutomationContext>, ICityDal
{

}
```

6. ICountyDal && EfCountyDal

```
4 references
public interface ICountyDal : IEntityRepository<County>
{

}
```

```
1 reference
public class EfCountyDal : EfEntityRepositoryBase<County, RealEstateAutomationContext>, ICountyDal
{

}
```

7. ICustomerDal && EfCustomerDal

```
public interface ICustomerDal : IEntityRepository<Customer>
{
}
```

```
1 reference
public class EfCustomerDal : EfEntityRepositoryBase<Customer, RealEstateAutomationContext>, ICustomerDal
{

}
```

8. IEmployeeDal && EfEmployeDal

```
4 references
public interface IEmployeeDal : IEntityRepository<Employee>
{

}
```

```
1 reference
public class EfEmployeeDal : EfEntityRepositoryBase<Employee, RealEstateAutomationContext>, IEmployeeDal
{

}
```

9. IExpenseDal && EfExpenseDal

```
4 references
public interface IExpensesDal :IEntityRepository<Expense>
{

}
```

```
1 reference
public class EfExpenseDal : EfEntityRepositoryBase<Expense, RealEstateAutomationContext>, IExpensesDal
{

}
```

10. IFieldDal && EfFieldDal

```
4 references
public interface IFieldDal : IEntityRepository<Field>
{

}
```

```
1 reference
public class EfFieldDal : EfEntityRepositoryBase<Field, RealEstateAutomationContext>, IFieldDal
{

}
```

11. IHouseDal && EfHouseDal

```
4 references
public interface IHouseDal : IEntityRepository<House>
{

}
```

```
1 reference
public class EfHouseDal : EfEntityRepositoryBase<House, RealEstateAutomationContext>, IHouseDal
{

}
```

12. IIncomeDal && EfIncomeDal

```
4 references
public interface IIncomeDal : IEntityRepository<Income>
{

}
```

```
1 reference
public class EfIncomeDal : EfEntityRepositoryBase<Income, RealEstateAutomationContext>, IIncomeDal
{

}
```

13. IOwnerDal && EfOwnerDal

```
4 references
public interface IOwnerDal : IEntityRepository<Owner>
{

}
```

```
1 reference
public class EfOwnerDal : EfEntityRepositoryBase<Owner, RealEstateAutomationContext>, IOwnerDal
{

}
```

14. IPlotDal && EfPlotDal

```
4 references
public interface IPlotDal : IEntityRepository<Plot>
{

}
```

```
1 reference
public class EfPlotDal : EfEntityRepositoryBase<Plot, RealEstateAutomationContext>, IPlotDal
{

}
```

15. IPropertyDal && EfPropertyDal

```
4 references
public interface IPropertyDal : IEntityRepository<Property>
{

}
```

```
1 reference
public class EfPropertyDal : EfEntityRepositoryBase<Property, RealEstateAutomationContext>, IPropertyDal
{

}
```

16. ISaleDal && EfSaleDal

```
4 references
public interface ISaleDal : IEntityRepository<Sale>
{

}
```

```
1 reference
public class EfSaleDal : EfEntityRepositoryBase<Sale, RealEstateAutomationContext>, ISaleDal
{

}
```

17. IShopDal && EfShopDal

```
4 references
public interface IShopDal : IEntityRepository<Shop>
{

}
```

```
1 reference
public class EfShopDal : EfEntityRepositoryBase<Shop, RealEstateAutomationContext>, IShopDal
{

}
```

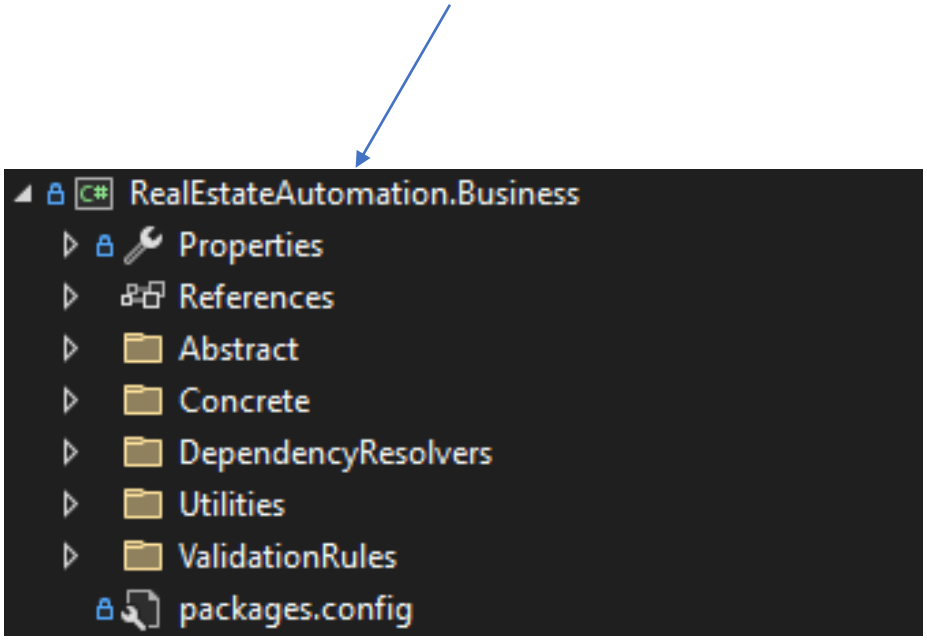
18. IUserDal && EfUserDal

```
4 references
public interface IUserDal : IEntityRepository<User>
{

}
```

```
1 reference
public class EfUserDal : EfEntityRepositoryBase<User, RealEstateAutomationContext>, IUserDal
{

}
```

## 5.4 Business Layer

The contents of my business layer are as in the image. I preferred to use it by making folders according to my needs for convenience. I made Interface and Class definitions for each entity in my Abstract and Concrete files. I performed dependency checks on the modules by integrating Ninject into my system in my DependencyResolvers folder. I integrated FluentValidation into my project for verification operations and used the last 2 folders for verification operations. I wrote a generic method for verification in my Utilities folder. Finally, I prepared validation procedures for my Entities in my ValidationRules folder. I will give detailed explanations below.



### 5.4.1 Abstract && Concrete

1. IAdminService && AdminManager

```
4 references
public interface IAdminService
{
    2 references
    List<Admin> GetAll();
    2 references
    void Add(Admin admin);
    2 references
    void Update(Admin admin);
    2 references
    void Update2(Admin admin);
}
```

I use it to connect with my presentation layer in my admin service interface. Here, I prepared the necessary operations in my presentation layer and defined and controlled these operations in the manager section.

```
2 references
public class AdminManager : IAdminService
{
    private readonly IAdminDal _adminDal;

    0 references
    public AdminManager(IAdminDal adminDal)
    {
        _adminDal = adminDal;
    }
```

This is my AdminManager class. I implemented IAdminService and prepared the internal processes of the necessary functions in my presentation layer. I created the content of my function by calling IAdminDal and assigning it to the class constructor.

```
2 references
public List<Admin> GetAll()
{
    try
    {
        return _adminDal.GetAll(filter: x :Admin => x.DeleteFlag == false);
    }
    catch (Exception)
    {
        MessageBox.Show(text: "There was an error loading the information. Please try again.", caption: "Information",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        throw;
    }
}
```

The content of my GetAll method is as shown in the figure. I prepared a try catch to catch the error in case of an error and checked the errors in case of an error. I call the data from my admindal class and assign parameters to it. It will return data with deleteflag false in the parameter. I do not delete the data from the database in order to never lose the data. DeleteFlag is true, when I delete data, I bring the undeleted data to my system thanks to this filter.

```
2 references
public void Add(Admin admin)
{

    ValidationTool.Validate(new AdminValidator(), admin);
    _adminDal.Add(admin);

}
```

In my add method, there is an add method for admin. I use the ValidationTool.validate in it, check the incoming content, and then add it. If there is an error, I catch it in the background. I will look more into the validation tool below.

This is my update method, here I update the incoming data after making the necessary validate checks. If I encounter an error, I capture it according to the type and code of the error and show it to the user via message box instead of revealing information to the user.

```
2 references
public void Update(Admin admin)
{

    try
    {
        ValidationTool.Validate(new AdminValidator(), admin);
        _adminDal.Update(admin);
    }

    catch (SqlException e)
    {
        switch (e.Number)
        {
            case 2627: // Unique key

                MessageBox.Show(text: "This record already exists please check your details", caption: "Information",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
                break;

            default:
                MessageBox.Show(text: "An unexpected database error occurred, please try again.", caption: "Information",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
                break;
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show(
            text: ex.InnerException == null ? ex.Message : "This record already exists please check your details",
            caption: "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

Depending on the need, I had to do some functions without using try catch, so I completed them directly this way. The reason for this was that while I was using the method on the presentation screen, I had to take some actions depending on the error situation and I caught the errors on the presentation screen. These methods are available in some areas and I will explain them in more detail where I use them.

```
2 references
public void Update2(Admin admin)
{
    ValidationTool.Validate(new AdminValidator(), admin);
    _adminDal.Update(admin);
}
```

I made a detailed explanation above for the operations about the service and managers of the Admin structure I explained above. Thanks to this service structure, I provide my controls and control the operations. I prepared the Service structures by using the structures common to each entity and added the ones with different features. Since they are all similar, you can examine the Service structures from the images below in order not to make repeated explanations.

2. ICityDal && CityManager

```
26 references
public interface ICityService
{
    12 references
    List<City> GetAll();
}
```

3. ICountyService && CountyManager

```
26 references
public interface ICountyService
{
    12 references
    List<County> GetAll(int key);
}
```

That's how I built the service for City and County. In the Manager section, I made an interface implementation and wrote the contents of the functions. Since the functions are similar, I made this explanation in this way, the detailed explanation is available in the admin section.

4. ICustomerService && CustomerManager

```
8 references
public interface ICustomerService
{
    4 references
    List<Customer> GetAll();
    2 references
    void Add(Customer customer);
    3 references
    void Update(Customer customer);
}
```

This is how I build customer service. I implemented it in the Manager section and prepared the transactions.

5. IEmployeeService && EmployeeManager

```
4 references
public interface IEmployeeService
{
    2 references
    List<Employee> GetAll();
    2 references
    void Add(Employee employee);
    2 references
    void Update(Employee employee);

    2 references
    void Update2(Employee employee);
}
```

My employee service construction is like this. I implemented it in the Manager section and prepared the transactions.

6. IExpenseService && ExpenseManager

```
6 references
public interface IExpenseService
{
    2 references
    List<Expense> GetAll();
    2 references
    void Add(Expense expense);
    3 references
    void Update(Expense expense);
}
```

Expense service construction is like this. I implemented it in the Manager section and prepared the transactions.

7. IFieldService && IExpense Manager

```
10 references
public interface IFieldService
{
    4 references
    List<Field> GetAll();
    2 references
    void Add(Field field);
    7 references
    void Update(Field field);
    1 reference
    List<Field> GetById(int id);
    2 references
    Field GetLastAddedField();
}
```

```
2 references
public Field GetLastAddedField()
{
    return _fieldDal.GetLastAddedEntity();
}
```

In this way, I prepared the functions required for me on the presentation screen and prepared my transactions. Unlike other fields, there is a different function in this service structure. The content of this function returns me the last added entity as in the figure on the side, and I capture and use it in the background.

8. IHouseService && HouseManager

```
10 references
public interface IHouseService
{
    2 references
    void Add(House house);
    7 references
    void Update(House house);
    2 references
    House GetLastAddedHouse();
    4 references
    List<House> GetAll();
}
```

My house service construction is like this. I implemented it in the Manager section and prepared the transactions.

9. IIncomeService && IncomeManager

```
6 references
public interface IIncomeService
{
    2 references
    List<Income> GetAll();
    2 references
    void Add(Income income);
    3 references
    void Update(Income income);
}
```

My income service construction is like this. I implemented it in the Manager section and prepared the transactions.

10. IOwnerService && OwnerManager

```
22 references
public interface IOwnerService
{
    10 references
    List<Owner> GetAll();
    5 references
    List<Owner> GetAll2();
    2 references
    void Save(Owner owner);
    3 references
    void Update(Owner owner);

}
```

My owner service construction is like this. I implemented it in the Manager section and prepared the transactions.

11. IPlotService && PlotManager

```
public interface IPlotService
{

    void Add(Plot plot);
    7 references
    void Update(Plot plot);
    2 references
    Plot GetLastAddedHouse();
    4 references
    List<Plot> GetAll();
}
```

This is my plot service construction. I implemented it in the Manager section and prepared the transactions.

12. IPropertyService && PropertyManager

```
10 references
public interface IPropertyService
{
    1 reference
    List<Property> GetAll();
    5 references
    void Add(Property property);

    5 references
    Property GetLastAddedProperty();
    9 references
    void Update(Property property);
    5 references
    void Delete(Property property);
}
```

In this service structure, I have defined a function for the delete operation. I remove the incoming data directly from the database according to its id. I used this way in some places to avoid data complexity.

```
public void Delete(Property property)
{
    _propertyDal.Delete(property);
}
```

13. ISaleService && SaleManager

```
4 references
public interface ISaleService
{
    5 references
    void Add(Sale sale);
    1 reference
    List<Sale> GetAll();
    9 references
    void Update(Sale sale);
}
```

My sale service construction is like this. I implemented it in the Manager section and prepared the transactions.

14. IShopService && ShopManager

```
public interface IShopService
{

    void Update(Shop shop);

    void Add(Shop shop);

    Shop GetLastAddedShop();
    4 references
    List<Shop> GetAll();
}
```

My shop service construction is like this. I implemented it in the Manager section and prepared the transactions.

15. IUserService && UserManager

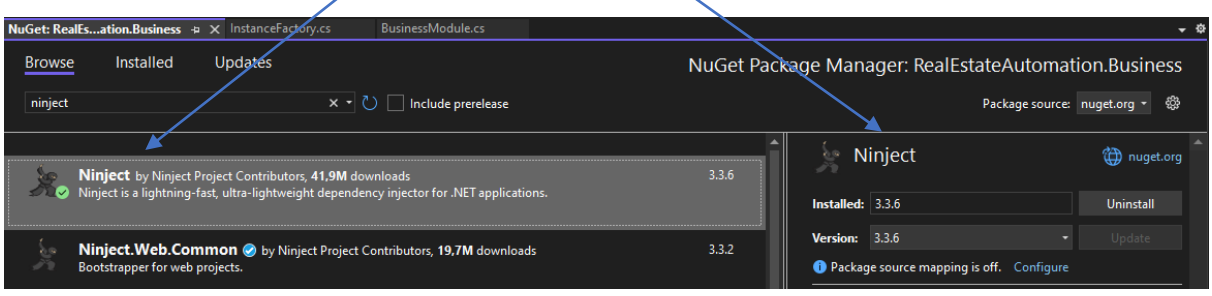
```
public interface IUserService
{

    void Add(User user);
    3 references
    User GetLastAddedUser();
    3 references
    void Delete(User user);
    4 references
    void Update(User user);
    8 references
    List<User> GetAll();
}
```

My user service construction is like this. I implemented it in the Manager section and prepared the transactions.

### 5.4.2 Dependency Resolvers

1. Ninject Integration

Ninject system is a system that allows us to manage our dependencies in the project. Dependency Injection (DI) pattern is prepared functionally and as a Container system used to abstract our dependencies between objects. I integrated this container structure into my project and aimed to manage my object dependencies.



2. Business Module

```
1 reference
public class BusinessModule : NinjectModule
{
    0 references
    public override void Load()
    {
        Bind<ICustomerService>().To<CustomerManager>().InSingletonScope();
        Bind<ICustomerDal>().To<EfCustomerDal>().InSingletonScope();

        Bind<IAdminService>().To<AdminManager>().InSingletonScope();
        Bind<IAdminDal>().To<EfAdminDal>().InSingletonScope();

        Bind<ICityService>().To<CityManager>().InSingletonScope();
        Bind<ICityDal>().To<EfCityDal>().InSingletonScope();

        Bind<ICountyService>().To<CountyManager>().InSingletonScope();
        Bind<ICountyDal>().To<EfCountyDal>().InSingletonScope();

        Bind<IEmployeeService>().To<EmployeeManager>().InSingletonScope();
        Bind<IEmployeeDal>().To<EfEmployeeDal>().InSingletonScope();

        Bind<IExpenseService>().To<ExpenseManager>().InSingletonScope();
        Bind<IExpensesDal>().To<EfExpenseDal>().InSingletonScope();

        Bind<IFieldService>().To<FieldManager>().InSingletonScope();
        Bind<IFieldDal>().To<EfFieldDal>().InSingletonScope();

        Bind<IHouseService>().To<HouseManager>().InSingletonScope();
        Bind<IHouseDal>().To<EfHouseDal>().InSingletonScope();

        Bind<IIncomeService>().To<IncomeManager>().InSingletonScope();
        Bind<IIncomeDal>().To<EfIncomeDal>().InSingletonScope();

        Bind<IOwnerService>().To<OwnerManager>().InSingletonScope();
        Bind<IOwnerDal>().To<EfOwnerDal>().InSingletonScope();

        Bind<IPlotService>().To<PlotManager>().InSingletonScope();
        Bind<IPlotDal>().To<EfPlotDal>().InSingletonScope();

        Bind<IPropertyService>().To<PropertyManager>().InSingletonScope();
        Bind<IPropertyDal>().To<EfPropertyDal>().InSingletonScope();

        Bind<ISaleService>().To<SaleManager>().InSingletonScope();
        Bind<ISaleDal>().To<EfSaleDal>().InSingletonScope();

        Bind<IShopService>().To<ShopManager>().InSingletonScope();
        Bind<IShopDal>().To<EfShopDal>().InSingletonScope();

        Bind<IUserService>().To<UserManager>().InSingletonScope();
        Bind<IUserDal>().To<EfUserDal>().InSingletonScope();

    }
}
```

I prepared a business module class and inherited it from the ninject modulde class and prepared the operations module construction.

In my function, I use the "Bind <> .To<>" structure to create an instance of the manager of whichever service is needed and inject it into the interface. In my "InSingletonScope" function, I specify the lifetime of the dependency and create only one object for each bind operation. I use this way and use the created object as long as the application is open.

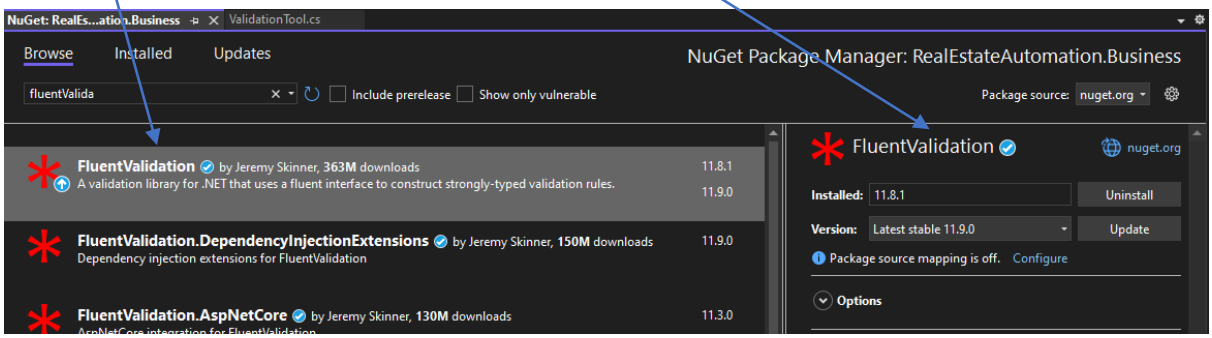
I defined a generic method in this class and I wanted this method to be applied according to the rules in the business module. In this way, I controlled my dependency on objects. By preparing a generic structure, I used the bear function for all my objects.

```
68 references
public class InstanceFactory
{
    68 references
    public static T GetInstance<T>()
    {
        var kernel = new StandardKernel(params modules: new BusinessModule());
        return kernel.Get<T>();
    }
}
```

### 5.4.2 Utilities && Validation Rules

1. Fluent Validation Integration

Fluent Validation is a validation library for checking the accuracy of data. By integrating this system into my project, I was able to control my data during my database operations in my project and get rid of incorrect and incorrect data records. I wrote rules for my entities in it and caught errors in case of deficiency according to the rules and showed these messages to the user in the system.




2. Validation Tool

I prepared a validate method in this way. I received the validator and entity written as parameters in my method and sent it back according to the rules written. If an error was encountered during the check, I captured the messages sent in the validator and the cause of the error and sent it to the user as an error message. The reason I asked for parameters as IValidator is that I actually made my validators as a class structure, but I inherit those classes from Abstractvalidator. I caught all validators in this way because Abstract validator is implemented from IValidator in itself.

```
26 references
public static class ValidationTool
{
    26 references
    public static void Validate(IValidator validator, object entity)
    {
        var context = new ValidationContext<object>(entity);
        var result = validator.Validate(context);

        if (result.Errors.Count > 0)
        {
            var errorMessages :string[] = result.Errors.Select(e :ValidationFailure => $"{e.PropertyName}: {e.ErrorMessage}\n").ToArray();
            string combinedErrorMessage = string.Join("", errorMessages);
            throw new ValidationException(combinedErrorMessage);
        }
    }
}
```

3. Admin Validator

My admin validator class is as shown in the next figure. I inherit AbstractValidator and send which entity control will be done into it. Then I wrote my rules in the constructor of my class and prepared my operations. I wrote a separate message for each rule to send a message to the user when the rules are not followed. In this way, I checked my validation checks in the background and caught it in case of error and sent a message to the user if which rule is incorrect.

```
4 references
public class AdminValidator : AbstractValidator<Admin>
{
    3 references
    public AdminValidator()
    {
        RuleFor(expression: x :Admin => x.NationalityId).MaximumLength(11).WithMessage("Please enter the correct Nationality ID");
        RuleFor(expression: x :Admin => x.NationalityId).MinimumLength(11).WithMessage("Please enter the correct Nationality ID");
        RuleFor(expression: x :Admin => x.NationalityId).NotEmpty().WithMessage("Nationality Id field cannot be empty");
        RuleFor(expression: x :Admin => x.FirstName).NotEmpty().WithMessage("Name field cannot be empty");
        RuleFor(expression: x :Admin => x.LastName).NotEmpty().WithMessage("Last name field cannot be empty");
        RuleFor(expression: x :Admin => x.Phone).NotEmpty().WithMessage("Phone field cannot be empty");
    }
}
```

4. Customer Validator

My Customer validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.

```
3 references
public class CustomerValidator : AbstractValidator<Customer>
{
    2 references
    public CustomerValidator()
    {
        RuleFor(expression: x :Customer => x.NationalityId).MaximumLength(11).WithMessage("Please enter the correct Nationality ID");
        RuleFor(expression: x :Customer => x.NationalityId).MinimumLength(11).WithMessage("Please enter the correct Nationality ID");
        RuleFor(expression: x :Customer => x.NationalityId).NotEmpty().WithMessage("Nationality Id field cannot be empty");
        RuleFor(expression: x :Customer => x.FirstName).NotEmpty().WithMessage("Name field cannot be empty");
        RuleFor(expression: x :Customer => x.LastName).NotEmpty().WithMessage("Last name field cannot be empty");
        RuleFor(expression: x :Customer => x.Phone).NotEmpty().WithMessage("Phone field cannot be empty");
    }
}
```

5. Employee Validator

My Employee validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.

```
4 references
public class EmployeeValidator : AbstractValidator<Employee>
{
    3 references
    public EmployeeValidator()
    {
        RuleFor(expression: x :Employee => x.NationalityId).MaximumLength(11).WithMessage("Please enter the correct Nationality ID");
        RuleFor(expression: x :Employee => x.NationalityId).MinimumLength(11).WithMessage("Please enter the correct Nationality ID");
        RuleFor(expression: x :Employee => x.NationalityId).NotEmpty().WithMessage("Nationality Id field cannot be empty");
        RuleFor(expression: x :Employee => x.FirstName).NotEmpty().WithMessage("Name field cannot be empty");
        RuleFor(expression: x :Employee => x.LastName).NotEmpty().WithMessage("Last name field cannot be empty");
        RuleFor(expression: x :Employee => x.Phone).NotEmpty().WithMessage("Phone field cannot be empty");
    }
}
```

6. Expense Validator

My Expense validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.

```
3 references
public class ExpenseValidator : AbstractValidator<Expense>
{
    2 references
    public ExpenseValidator()
    {
        RuleFor(expression: x :Expense => x.ExpenseDate).NotEmpty().WithMessage("Income Date cannot be empty.");
        RuleFor(expression: x :Expense => x.ExpensePrice).NotEmpty().WithMessage("Income Price cannot be empty.");
        RuleFor(expression: x :Expense => x.ExpensePrice).GreaterThan(valueToCompare: 0).WithMessage("Price cannot be less than 0.");
        RuleFor(expression: x :Expense => x.Description).NotEmpty().WithMessage("Description cannot be empty.");
    }
}
```

7. Field Validator

My Field validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.

```
3 references
public class FieldValidator : AbstractValidator<Field>
{
    2 references
    public FieldValidator()
    {
        RuleFor(expression: x :Field => x.OwnerId).NotEmpty().WithMessage("Owner space cannot be empty.");
        RuleFor(expression: x :Field => x.Area).NotEmpty().WithMessage("Area cannot be empty.");
        RuleFor(expression: x :Field => x.Area).GreaterThan(valueToCompare: 0).WithMessage("Area cannot be less than 0.");
        RuleFor(expression: x :Field => x.Pafta).NotEmpty().WithMessage("Pafta cannot be empty.");
        RuleFor(expression: x :Field => x.City).NotEmpty().WithMessage("City cannot be empty.");
        RuleFor(expression: x :Field => x.County).NotEmpty().WithMessage("County cannot be empty.");
        RuleFor(expression: x :Field => x.Price).NotEmpty().WithMessage("Price cannot be empty.");
        RuleFor(expression: x :Field => x.Price).GreaterThan(valueToCompare: 0).WithMessage("Price cannot be less than 0.");
        RuleFor(expression: x :Field => x.PropertyId).NotEmpty().WithMessage("Please fill in the missing information and try again.");
    }
}
```

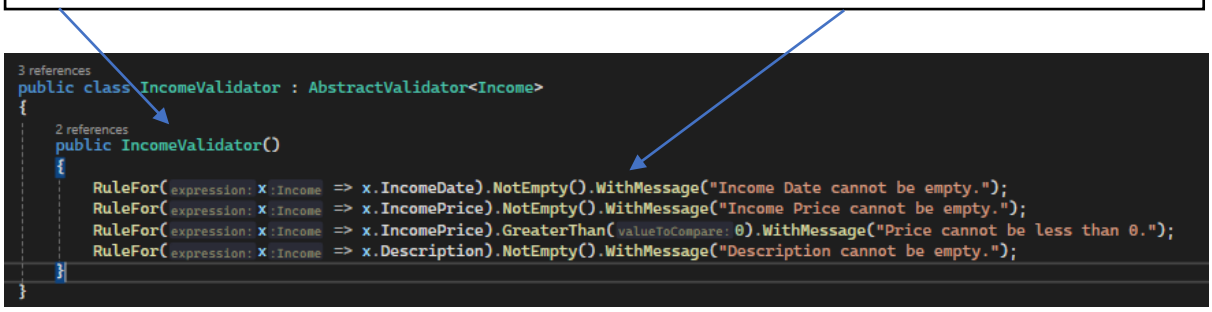
8. House Validator

My House validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.

```
3 references
public class HouseValidator : AbstractValidator<House>
{
    2 references
    public HouseValidator()
    {
        RuleFor(expression: x :House => x.OwnerId).NotEmpty().WithMessage("Owner space cannot be empty.");
        RuleFor(expression: x :House => x.Area).NotEmpty().WithMessage("Area cannot be empty.");
        RuleFor(expression: x :House => x.Area).GreaterThan(valueToCompare: 0).WithMessage("Area cannot be less than 0.");
        RuleFor(expression: x :House => x.HouseType).NotEmpty().WithMessage("House Type cannot be empty.");
        RuleFor(expression: x :House => x.City).NotEmpty().WithMessage("City cannot be empty.");
        RuleFor(expression: x :House => x.County).NotEmpty().WithMessage("County cannot be empty.");
        RuleFor(expression: x :House => x.Price).NotEmpty().WithMessage("Price cannot be empty.");
        RuleFor(expression: x :House => x.Price).GreaterThan(valueToCompare: 0).WithMessage("Price cannot be less than 0.");
    }
}
```

9. Income Validator

My Income validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.



```
3 references
public class IncomeValidator : AbstractValidator<Income>
{
    2 references
    public IncomeValidator()
    {
        RuleFor(expression: x :Income => x.IncomeDate).NotEmpty().WithMessage("Income Date cannot be empty.");
        RuleFor(expression: x :Income => x.IncomePrice).NotEmpty().WithMessage("Income Price cannot be empty.");
        RuleFor(expression: x :Income => x.IncomePrice).GreaterThan(valueToCompare: 0).WithMessage("Price cannot be less than 0.");
        RuleFor(expression: x :Income => x.Description).NotEmpty().WithMessage("Description cannot be empty.");
    }
}
```

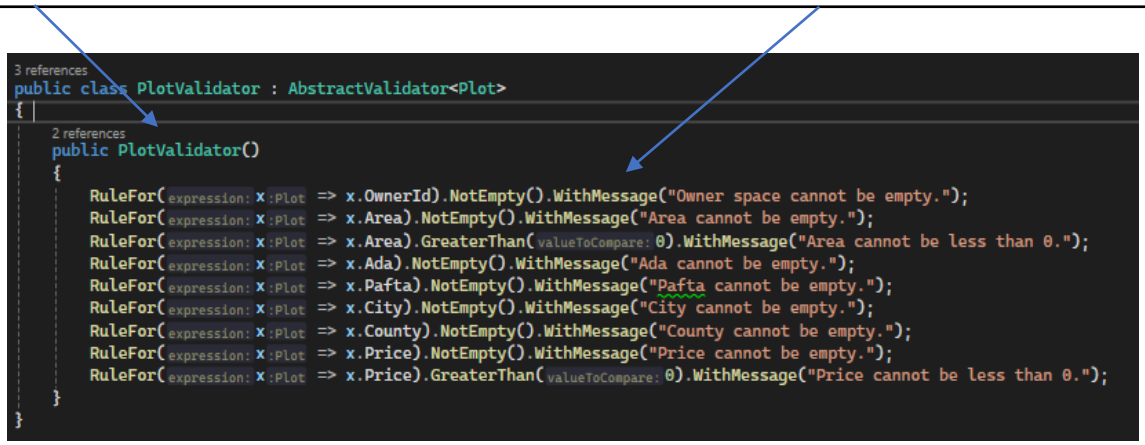
10. Owner Validator

My Owner validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.

```
3 references
public class OwnerValidator : AbstractValidator<Owner>
{
    2 references
    public OwnerValidator()
    {
        RuleFor(expression: x :Owner => x.NationalityId).MaximumLength(11).WithMessage("Please enter the correct Nationality ID");
        RuleFor(expression: x :Owner => x.NationalityId).MinimumLength(11).WithMessage("Please enter the correct Nationality ID");
        RuleFor(expression: x :Owner => x.NationalityId).NotEmpty().WithMessage("Nationality Id field cannot be empty");
        RuleFor(expression: x :Owner => x.FirstName).NotEmpty().WithMessage("Name field cannot be empty");
        RuleFor(expression: x :Owner => x.LastName).NotEmpty().WithMessage("Last name field cannot be empty");
        RuleFor(expression: x :Owner => x.Phone).NotEmpty().WithMessage("Phone field cannot be empty");
    }
}
```

11. Plot Validator

My Plot validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.



```
3 references
public class PlotValidator : AbstractValidator<Plot>
{
    2 references
    public PlotValidator()
    {
        RuleFor(expression: x :Plot => x.OwnerId).NotEmpty().WithMessage("Owner space cannot be empty.");
        RuleFor(expression: x :Plot => x.Area).NotEmpty().WithMessage("Area cannot be empty.");
        RuleFor(expression: x :Plot => x.Area).GreaterThan(valueToCompare: 0).WithMessage("Area cannot be less than 0.");
        RuleFor(expression: x :Plot => x.Ada).NotEmpty().WithMessage("Ada cannot be empty.");
        RuleFor(expression: x :Plot => x.Pafta).NotEmpty().WithMessage("Pafta cannot be empty.");
        RuleFor(expression: x :Plot => x.City).NotEmpty().WithMessage("City cannot be empty.");
        RuleFor(expression: x :Plot => x.County).NotEmpty().WithMessage("County cannot be empty.");
        RuleFor(expression: x :Plot => x.Price).NotEmpty().WithMessage("Price cannot be empty.");
        RuleFor(expression: x :Plot => x.Price).GreaterThan(valueToCompare: 0).WithMessage("Price cannot be less than 0.");
    }
}
```

12. Sale Validator

My Sale validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.

```
3 references
public class SaleValidator : AbstractValidator<Sale>
{
    2 references
    public SaleValidator()
    {
        RuleFor(expression: x :Sale => x.SalePropertyId).NotEmpty().WithMessage("Property cannot be empty, please select a property");
        RuleFor(expression: x :Sale => x.CustomerId).NotEmpty().WithMessage("Customer cannot be empty, please select a customer");
        RuleFor(expression: x :Sale => x.SalePrice).NotEmpty().WithMessage("Price cannot be empty, please entry price");
        RuleFor(expression: x :Sale => x.SalePrice).GreaterThan(valueToCompare: 0).WithMessage("Price cannot be less than 0");
    }
}
```

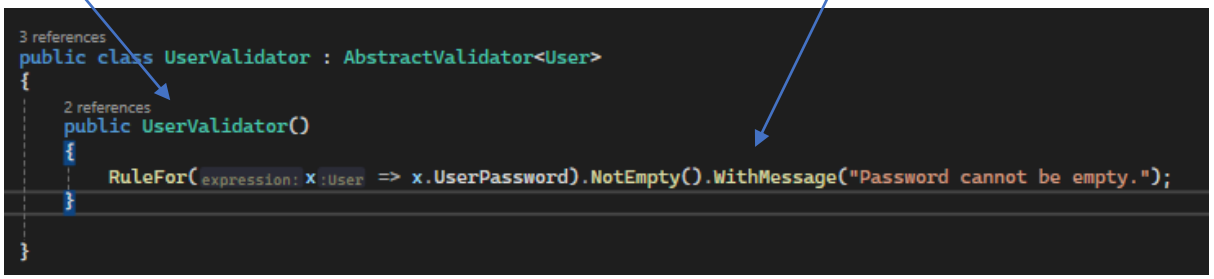
13. Shop Validator

My Shop validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.

```
3 references
public class ShopValidator : AbstractValidator<Shop>
{
    2 references
    public ShopValidator()
    {
        RuleFor(expression: x :Shop => x.OwnerId).NotEmpty().WithMessage("Owner space cannot be empty.");
        RuleFor(expression: x :Shop => x.Area).NotEmpty().WithMessage("Area cannot be empty.");
        RuleFor(expression: x :Shop => x.Area).GreaterThan(valueToCompare: 0).WithMessage("Area cannot be less than 0.");
        RuleFor(expression: x :Shop => x.City).NotEmpty().WithMessage("City cannot be empty.");
        RuleFor(expression: x :Shop => x.County).NotEmpty().WithMessage("County cannot be empty.");
        RuleFor(expression: x :Shop => x.Price).NotEmpty().WithMessage("Price cannot be empty.");
        RuleFor(expression: x :Shop => x.Price).GreaterThan(valueToCompare: 0).WithMessage("Price cannot be less than 0.");
    }
}
```

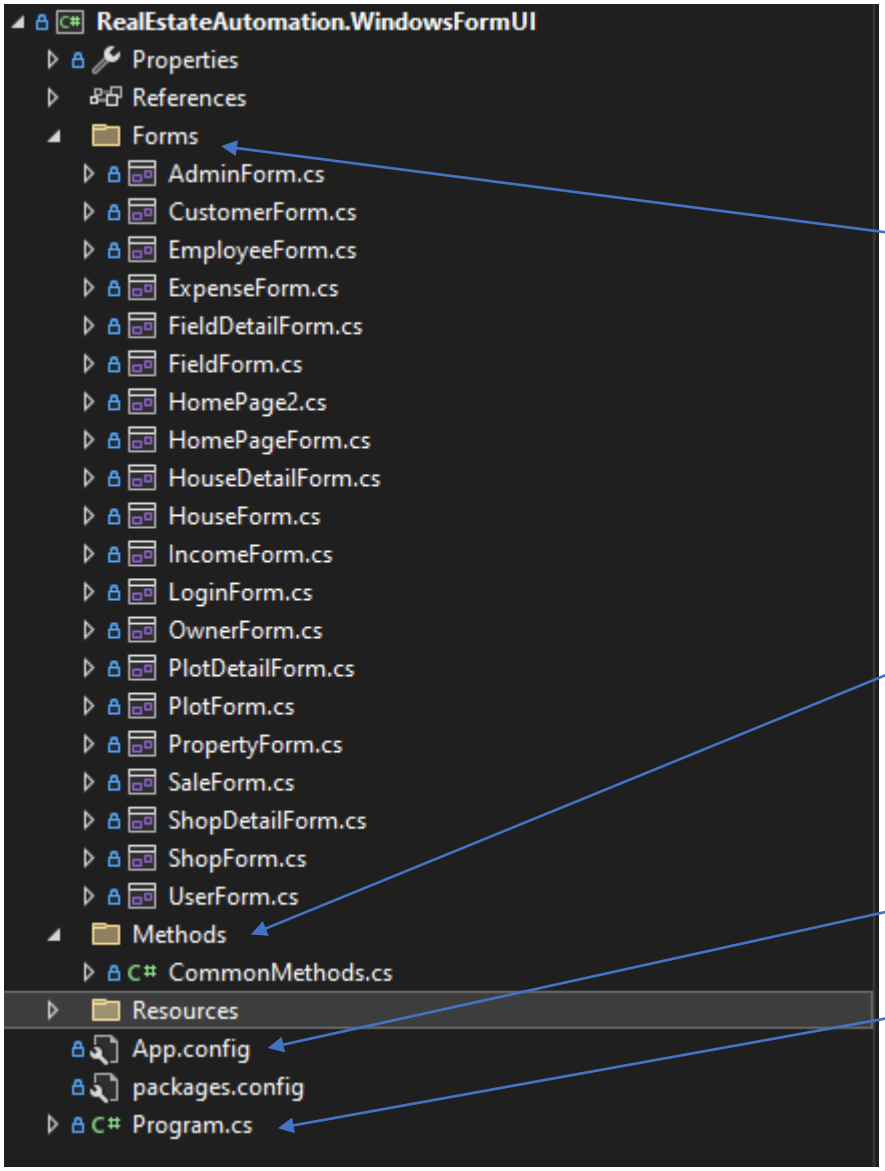
14. User Validator

My User validation class is as in the image below. I called the necessary rules in my constructor and sent a message to the user according to the error status.



```
3 references
public class UserValidator : AbstractValidator<User>
{
    2 references
    public UserValidator()
    {
        RuleFor(expression: x :User => x.UserPassword).NotEmpty().WithMessage("Password cannot be empty.");
    }

}
```

## 5.5 Presentation Layer




My presentation layer is as shown in the next figure. I prepared it according to my needs by folderizing for an organized view. While preparing my screens for the user in the Forms folder, I used it for my common methods in my Method folder. In my App.Config file, I made the connection of my project with Sql Server and connected the system with the database. Finally, in my program.cs class, I completed my system by giving the login form as a start, you can examine the details below.

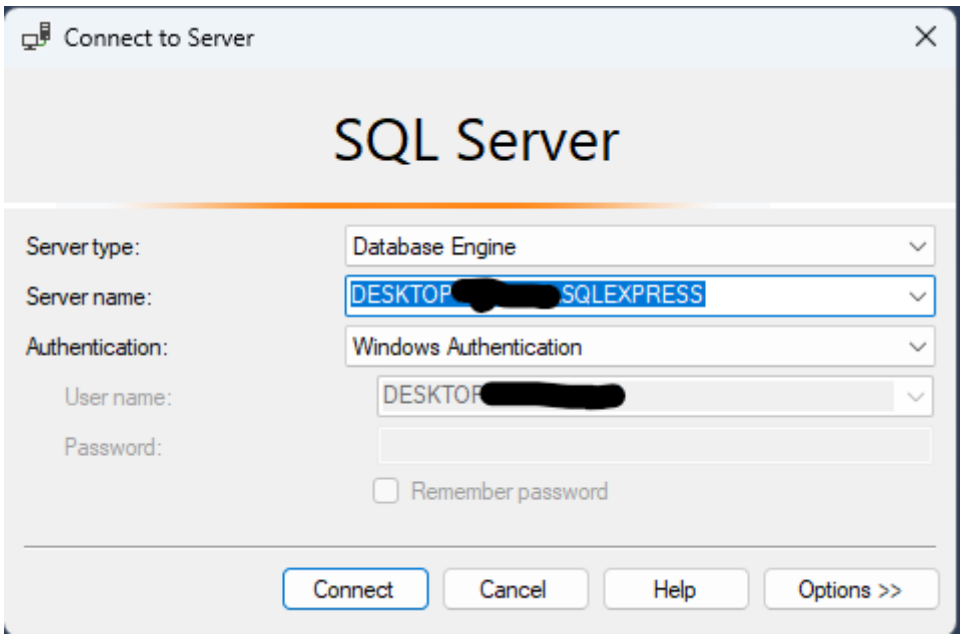
App.config

Connection Strings:

I am writing connection strings for my database connection. In this way, I connect my database and my system to each other.

```
<connectionStrings>
    <add name ="RealEstateAutomationContext"
        connectionString="Data Source=DESKTOP-[redacted]\SQLEXPRESS; initial catalog= RealEstateAutomation;
        integrated security = true"
        providerName="System.Data.SqlClient"/>
</connectionStrings>
```

I send the name under which I will work to the database. Afterwards, I write my context connection and connect the database and the system. Finally, I authenticate, indicate that I am using sqlClient in the system, and complete my connection string process.



### 5.5.1 Methods

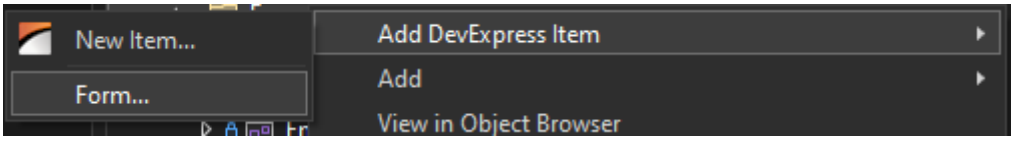
Common Methods

I prepared my common methods class to write the common methods that I use in all classes. I have a method called ExcelTransfer, thanks to this method, the user can transfer the data to Excel if he wants. I set the type of registration above and then perform the registration process according to the user's request and the name he/she gives. A grid view is required to use the method. I take it as a parameter and complete the process.

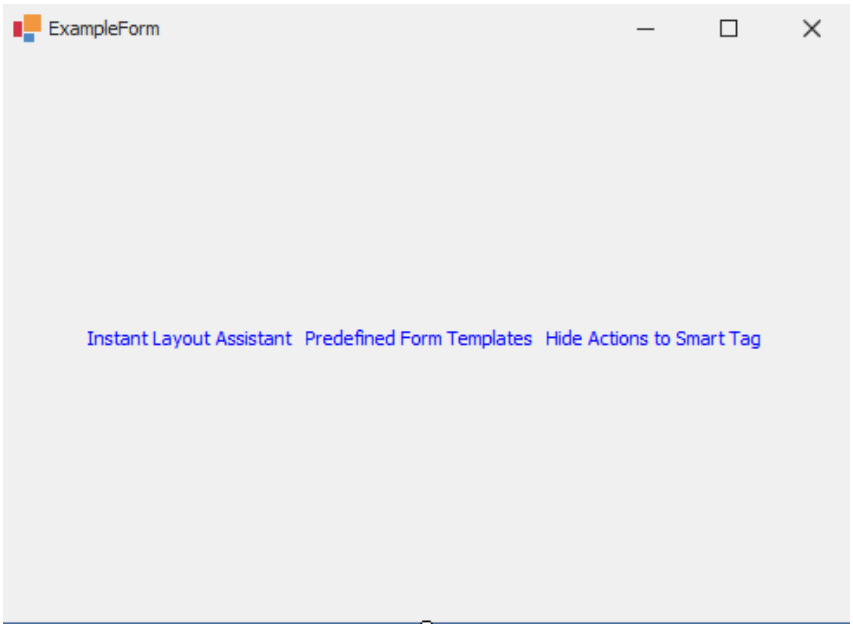
```
22 references
public class CommonMethods
{
    11 references
    public void ExcelTransfer(GridView Gr)
    {
        DialogResult confirmation1 = MessageBox.Show(text: @"Are you sure you want to save to Excel?",
            caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);


        if (confirmation1 == DialogResult.Yes)
        {

            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Xlsx|*.xlsx";
            saveFileDialog.Title = "Select Excel Save Location.";
            saveFileDialog.ShowDialog();

            if (saveFileDialog.FileName != null)
            {
                try
                {
                    Gr.ExportToXlsx(saveFileDialog.FileName);
                    DialogResult confirmation = MessageBox.Show(text: @"Would you like to open the file?",
                        caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
                    if (confirmation == DialogResult.Yes)
                    {
                        Process.Start(saveFileDialog.FileName);
                    }
                }
                catch (Exception)
                {
                    MessageBox.Show(text: @"Your transaction has been canceled...", caption: @"Information", MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
                }
            }
            else
            {
                MessageBox.Show(text: @"Select Excel Save Location.", caption: @"Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
        else
        {
            MessageBox.Show(text: @"Your transaction has been canceled.", caption: @"Information", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }

    }


}
```

### 5.5.2 Forms

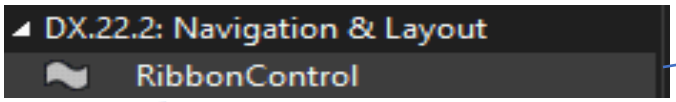
1. Adding and Editing a Form



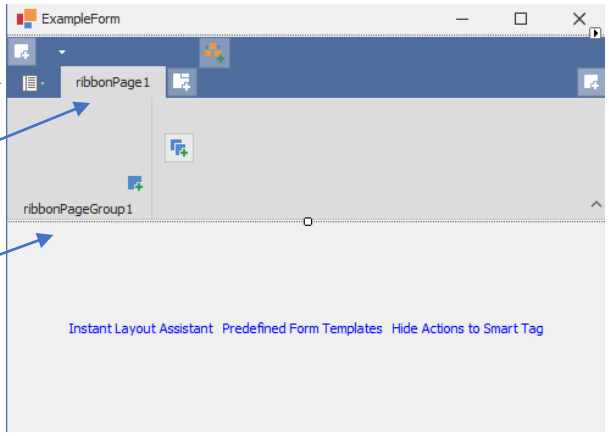

I add a new form by right-clicking on the Forms folder and doing this.

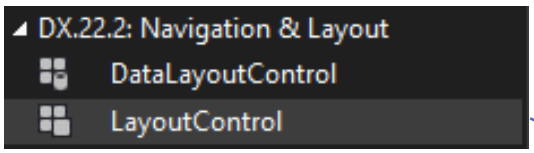



The blank form is as shown in the image below. Now I will use Devexpress Toolbox for my tools and make my Form edits.

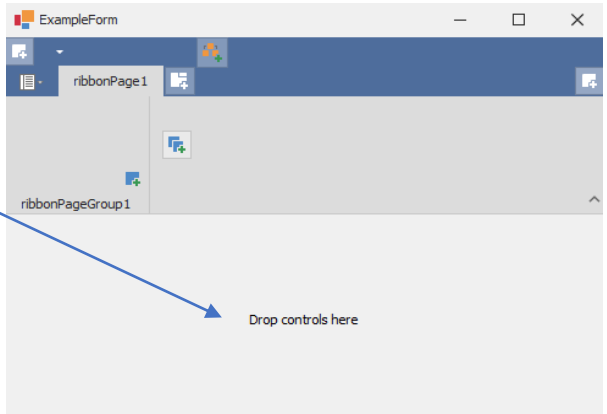


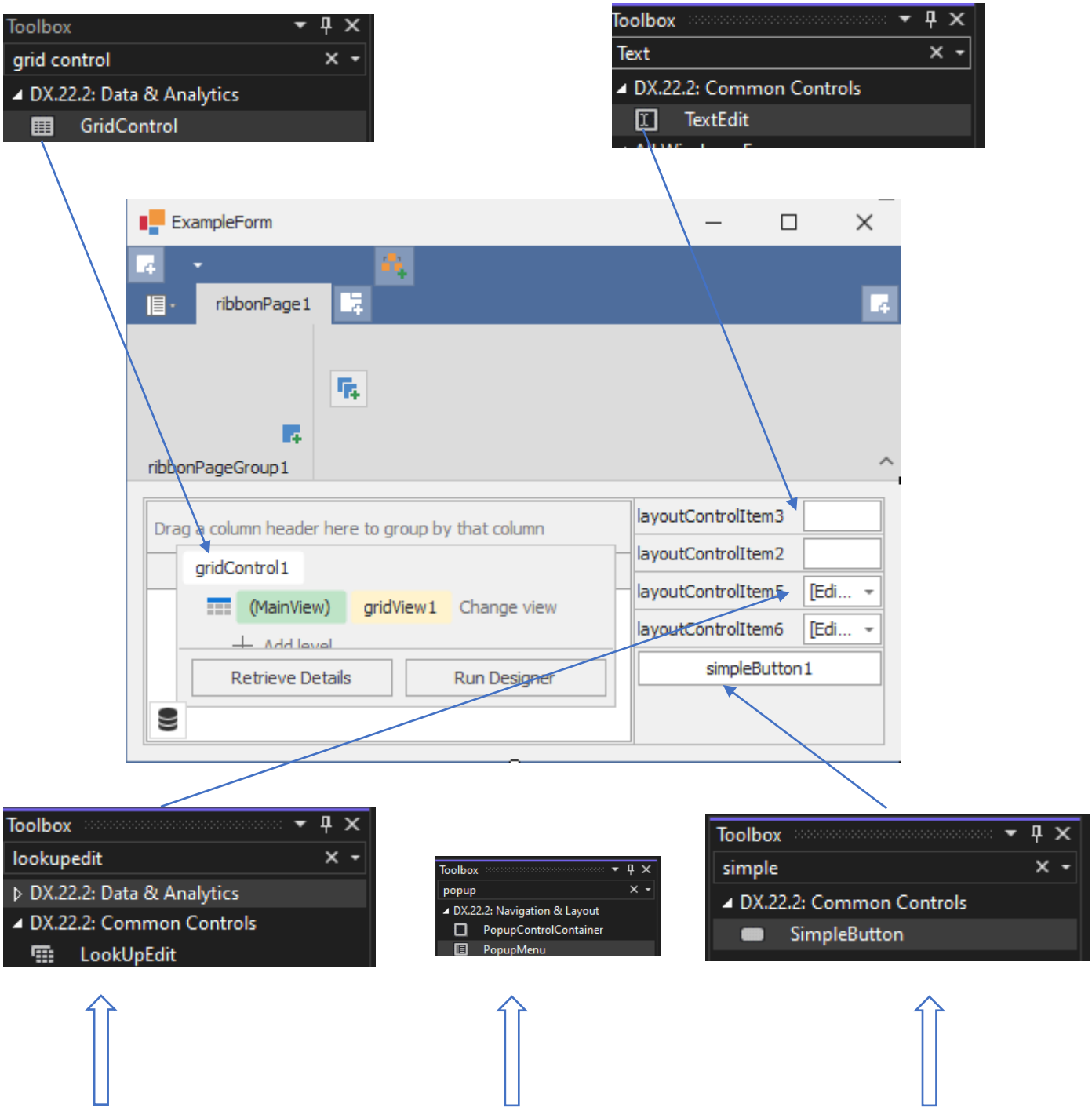

I aimed for a nice look by adding a ribbon control. I carry out the operations by making Ribbon Page and Ribbon group joints. This is prepared as an example only.



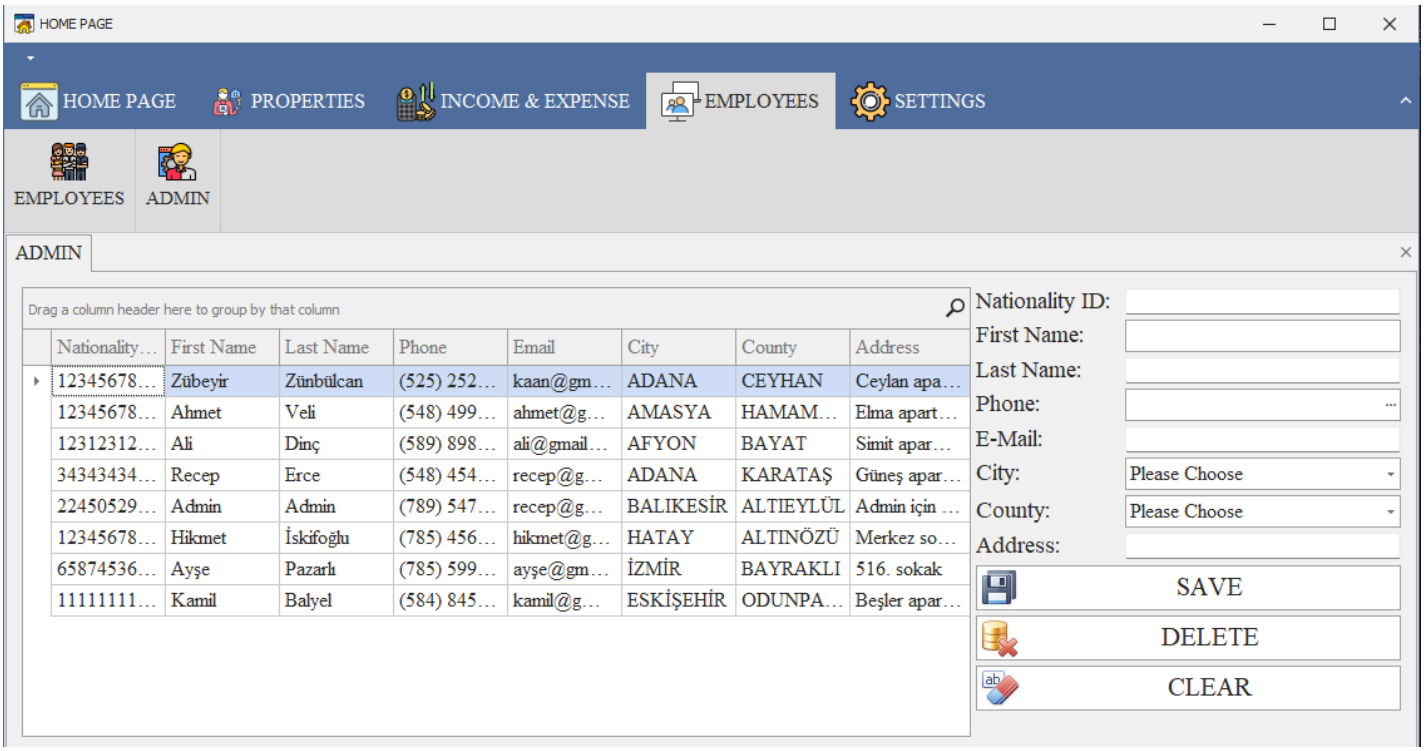

Using Layout control, I place all the tools used (ex: Text Edit, Grid Control, Simple Button, etc.) in a certain order. This is how I control enlargement and any reduction.






The tools I generally use are as shown above. I named the vehicles according to their types to avoid confusion. I used the popup menu to right-click and perform operations. You can view the used versions in more detail below. This is just an example. To explain editing with DevExpress tools.

2. Admin Form




The appearance of the admin form is as shown in the figure above. When you click on the Employess ribbon page and press the Admin button, the screen appears as mdi. While Grid Controde allows the user to display my data, on the right side of the grid, I have the necessary fields for recording, updating and deleting data. You can review the images below to examine the processes in the background. I will talk about my transactions there.

```
4 references
public partial class AdminForm : DevExpress.XtraEditors.XtraForm
{
    0 references
    public AdminForm()
    {
        InitializeComponent();

        _adminService = InstanceFactory.GetInstance<IAdminService>();
        _cityService = InstanceFactory.GetInstance<ICityService>();
        _countyService = InstanceFactory.GetInstance<ICountyService>();
        _userService = InstanceFactory.GetInstance<IUserService>();
    }

    private readonly IAdminService _adminService;
    private readonly IUserService _userService;
    private readonly ICityService _cityService;
    private readonly ICountyService _countyService;
    private readonly CommonMethods _commonMethods = new CommonMethods();
```

While preparing the background operations of the Admin form, I defined all the necessary service structures and created objects from them while the Admin form was started. I checked the dependency using the Ninject module to create entiy. You can find a detailed description of Ninject in the "Dependency Resolvers" section above.

```
1 reference
private void AdminForm_Load(object sender, EventArgs e)
{
    LoadAdmin();
    LoadCity();
}

4 references
private void LoadAdmin()
{
    grcAdmin.DataSource = _adminService.GetAll();

}

1 reference
private void LoadCity()
{
    lkuCity.Properties.DataSource = _cityService.GetAll();
    lkuCity.Properties.DisplayMember = "CityName";
    lkuCity.Properties.ValueMember = "Id";
}
```

While loading the Admin Form, I uploaded data to grid control and lookupedite. While I brought the admin data from my database to the grid control, I brought the city information from my database to lookupedite. I set the values that I wanted to be displayed on Lookupedit and kept in the background and made my transactions ready. While performing the GetAll operation for the database, I connect to the Data Access layer using my service structures and call the necessary operations. You can find a more detailed analysis in the Business module Abstract && Concrete and DataAcces Abstract && Concrete sections above.

```
1 reference
private void LoadCounty()
{
    lkuCounty.Properties.DataSource = _countyService.GetAll(key: Convert.ToInt32(lkuCity.EditValue)).ToList();
    lkuCounty.Properties.DisplayMember = "CountyName";
    lkuCounty.Properties.ValueMember = "Id";
}

1 reference
private void lkuCity_EditValueChanged(object sender, EventArgs e)
{
    LoadCounty();
}
```

I prepared a function like this to pull data from Lookupedite. The function lists the data by County according to the value selected in lookupCity and sent back. I call LookupCity whenever there is a change and check the County list according to the selected city.

```
1 reference
void LoadClick()
{
    if (grwAdmin.FocusedRowHandle >= 0)
    {
        txtId.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "Id").ToString();
        txtNationalityId.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "NationalityId").ToString();
        txtNationalityId.ReadOnly = true;
        txtFirstName.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "FirstName").ToString();
        txtLastName.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "LastName").ToString();
        btnPhone.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "Phone").ToString();
        txtEmail.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "Email").ToString();
        lkuCounty.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "County").ToString();
        lkuCity.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "City").ToString();
        txtAddress.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "Address").ToString();
        txtDeleteFlag.Text = grwAdmin.GetFocusedRowCellValue(fieldName: "DeleteFlag").ToString();
    }
}

1 reference
private void grcAdmin_Click(object sender, EventArgs e)
{
    LoadClick();
}
```

My LoadClick function operations are as shown in the figure. I check inside the grid view and if there is data, I capture the data in the selected row in my textboxes and lookupedits according to the names of the columns and bring them to their respective places. When I click on the GridControle, I call my method and bring the texts to the user to update their data.

```
5 references
void Clear()
{
    txtNationalityId.Text = "";
    txtNationalityId.ReadOnly = false;
    txtFirstName.Text = "";
    txtLastName.Text = "";
    btnPhone.Text = "";
    txtEmail.Text = "";
    lkuCounty.EditValue = null;
    lkuCity.EditValue = null;
    txtAddress.Text = "";
    txtId.Text = "";
    txtDeleteFlag.Text = "";
}
```

My clear method is as shown. I prepared it to clean the data after an addition and use it where necessary. It's a function I use to clean inside texts.

```
1 reference
private void btnExcelTransfer2_ItemClick(object sender, DevExpress.XtraBars.ItemClickEventArgs e)
{
    _commonMethods.ExcelTransfer(grwAdmin);
}
```

When my transfer to Excel button is clicked, I call my ExcelTransfer method in my commonmethods class and send the grid view to be saved in it. You can find a detailed review of the method in the Methods section above.

First of all, I want to talk about this. I have a textid field. I will use this field. I hid it. I used it to check whether a new record is being added or an existing data is being updated according to this field.

I asked the user if he wanted to register and depending on his answer, I continued or canceled the process. I did this using a dialogresult.

When I add a new admin, I first add a new user. For the user, I get the NationalityId as username in the background and set the password as default, then the user can go and change it. I grant the user permission, set the deleteflag value to false, and record the login information for the admin while creating a new admin in the background. With GetLastAddedUser, I get the information of the last registered user and assign the id number to a variable. You can examine my service structures in detail in the Business layer abstract && concrete section above.

While adding an admin, I call my method in my service structure, send the entered data and complete my admin registration process.

After adding data, I call my LoadAdmin method to display the added data and load the data into the grid, and with my Clear method, I clear the added data from text and clear it to add new data.

Since the admin registration process would fail in case of an error, I caught the error, sent the reason for the error to the user as a message, and deleted the resulting user data directly from my database according to the id to avoid data complexity.

Finally, I write the message that will be shown to the user in case the user cancels the registration process without confirming it, with the help of the messagebox and finish the process.

In this area, if my textid field is not empty, I accept it as an update process and perform or cancel the update process according to the response given by the user.

I call the admin update function from my service structure and enter the data of the admin to be updated. Since the AdminId and UserId fields do not change, I get them automatically from the gridview. Then, by reloading the Grid, I list it again and display the update.

```
2 references
void Save()
{
    if (txtId.Text == "")
    {
        DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to save the information?",
            caption: @"Information", MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);

        if (confirmation == DialogResult.Yes)
        {
            _userService.Add(new User
            {
                UserName = txtNationalityId.Text,
                UserPassword = "1234",
                UserAuthorization = 1,
                DeleteFlag = false
            });

            User lastAddedUser = _userService.GetLastAddedUser();
            int newUserId = 0;
            newUserId = lastAddedUser?.Id ?? 0;

            if (newUserId != -1 && newUserId != 0)
            {
                try
                {
                    _adminService.Add(new Admin
                    {
                        UserId = newUserId,
                        NationalityId = txtNationalityId.Text,
                        FirstName = txtFirstName.Text,
                        LastName = txtLastName.Text,
                        Phone = btnPhone.Text,
                        Email = txtEmail.Text,
                        City = lkuCity.Text,
                        County = lkuCounty.Text,
                        Address = txtAddress.Text,
                        DeleteFlag = false
                    });
                }
                catch (Exception e)
                {
                    _userService.Delete(new User
                    {
                        Id = newUserId
                    });

                    MessageBox.Show(
                        text: e.InnerException == null ? e.Message : "This record already exists please check your details",
                        caption: @"Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                LoadAdmin();
                Clear();
            }
            else
            {
                MessageBox.Show(text: @"Your transaction has been canceled.",
                    caption: @"Information", MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
        }
```

```
else if (txtId.Text != "")
{
    DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to update the information?",
        caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (confirmation == DialogResult.Yes)
    {
        _adminService.Update(new Admin
        {
            Id = Convert.ToInt32(grwAdmin.GetRowCellValue(grwAdmin.FocusedRowHandle, fieldName: "Id")),
            UserId = Convert.ToInt32(grwAdmin.GetRowCellValue(grwAdmin.FocusedRowHandle, fieldName: "UserId")),
            NationalityId = txtNationalityId.Text,
            FirstName = txtFirstName.Text,
            LastName = txtLastName.Text,
            Phone = btnPhone.Text,
            Email = txtEmail.Text,
            City = lkuCity.Text,
            County = lkuCounty.Text,
            Address = txtAddress.Text,
            DeleteFlag = false
        });
        LoadAdmin();
        Clear();
    }
    else
    {
        MessageBox.Show(text: @"Your transaction has been canceled.", caption: @"Information", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}
```

In this area, if my textid field is not empty, I accept it as an update process and perform or cancel the update process according to the response given by the user.

As I mentioned before, during the deletion process, I do not delete the data in the database at all, I make an update and set the deleteflag to true. In this way, I can take control of my system if I have used it in other places in the background. I take the necessary information from the texts as above and update it in the background.

```
2 references
void Remove()
{
    DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to delete your information?",
        caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (confirmation == DialogResult.Yes)
    {
        try
        {
            _adminService.Update2(new Admin
            {
                Id = Convert.ToInt32(grwAdmin.GetRowCellValue(grwAdmin.FocusedRowHandle, fieldName: "Id")),
                UserId = Convert.ToInt32(grwAdmin.GetRowCellValue(grwAdmin.FocusedRowHandle, fieldName: "UserId")),
                NationalityId = txtNationalityId.Text,
                FirstName = txtFirstName.Text,
                LastName = txtLastName.Text,
                Phone = btnPhone.Text,
                Email = txtEmail.Text,
                City = lkuCity.Text,
                County = lkuCounty.Text,
                Address = txtAddress.Text,
                DeleteFlag = true
            });

            int userId = Convert.ToInt32(grwAdmin.GetRowCellValue(grwAdmin.FocusedRowHandle, fieldName: "UserId"));
            User updateUser = _userService.GetAll().FirstOrDefault(x :User => x.Id == userId);
            string userName = "";
            string password = "";

            if (updateUser != null)
            {
                userName = updateUser.UserName;
                password = updateUser.UserPassword;
            }

            _userService.Update(new User
            {
                Id = Convert.ToInt32(grwAdmin.GetRowCellValue(grwAdmin.FocusedRowHandle, fieldName: "UserId")),
                UserName = userName,
                UserPassword = password,
                UserAuthorization = 1,
                DeleteFlag = true
            });
        }

        catch (Exception ex)
        {
            MessageBox.Show(
                text: ex.InnerException == null ? ex.Message : "This record already exists please check your details",
                caption: "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);

        }

        LoadAdmin();
        Clear();
    }
    else
    {
        MessageBox.Show(text: @"Your transaction has been canceled.", caption: @"Information", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }

}
```

Here, I used the update2 function from my service structure because in case of an error, I need to catch that error here, so I catch the error here, not in my service structure. I used it this way in order not to spoil my original update structure.

While the admin is deleting, I update the user information of that deleted admin and cancel the login information. In this way, I control the admin's login to the system. I take the information based on the UserId I receive, paste it into variables, and update the user.

If the transaction is cancelled, I send a message to the user saying your transaction has been cancelled.

In case of an error, I catch it and show the user the reason for the error as a message.

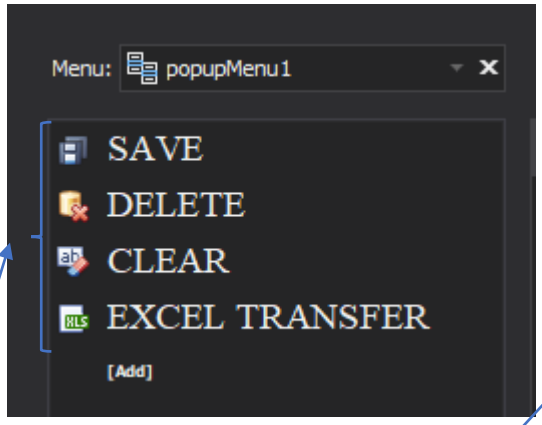
At the end of the process, I reload the gridcontrol data and clear the texts with clear.

I called the operations I did above when the button was clicked and carried out my operations. I have 2 buttons because I can perform these operations with right click, so I called the function when they clicked.

```
1 reference
private void btnSave2_ItemClick(object sender, DevExpress.XtraBars.ItemClickEventArgs e)
{
    Save();
}

1 reference
private void btnRemove2_ItemClick(object sender, DevExpress.XtraBars.ItemClickEventArgs e)
{
    Remove();
}

1 reference
private void btnSave_Click(object sender, EventArgs e)
{
    Save();
}

1 reference
private void btnRemove_Click(object sender, EventArgs e)
{
    Remove();
}
```

```
1 reference
private void btnClear_Click(object sender, EventArgs e)
{
    DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to clear the boxes?",
        caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (confirmation == DialogResult.Yes)
    {
        Clear();
    }
    else
    {
        MessageBox.Show(text: @"Your transaction has been canceled.", caption: @"Information", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}

1 reference
private void btnClear2_ItemClick(object sender, DevExpress.XtraBars.ItemClickEventArgs e)
{
    DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to clear the boxes?",
        caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (confirmation == DialogResult.Yes)
    {
        Clear();
    }
    else
    {
        MessageBox.Show(text: @"Your transaction has been canceled.", caption: @"Information", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}
```

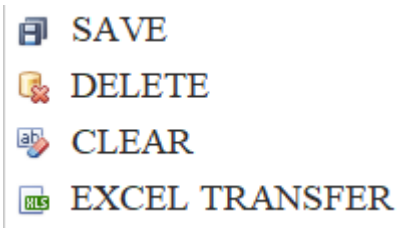
In my Clear buttons, I had the user perform the clear operation based on the user's questions and answers.



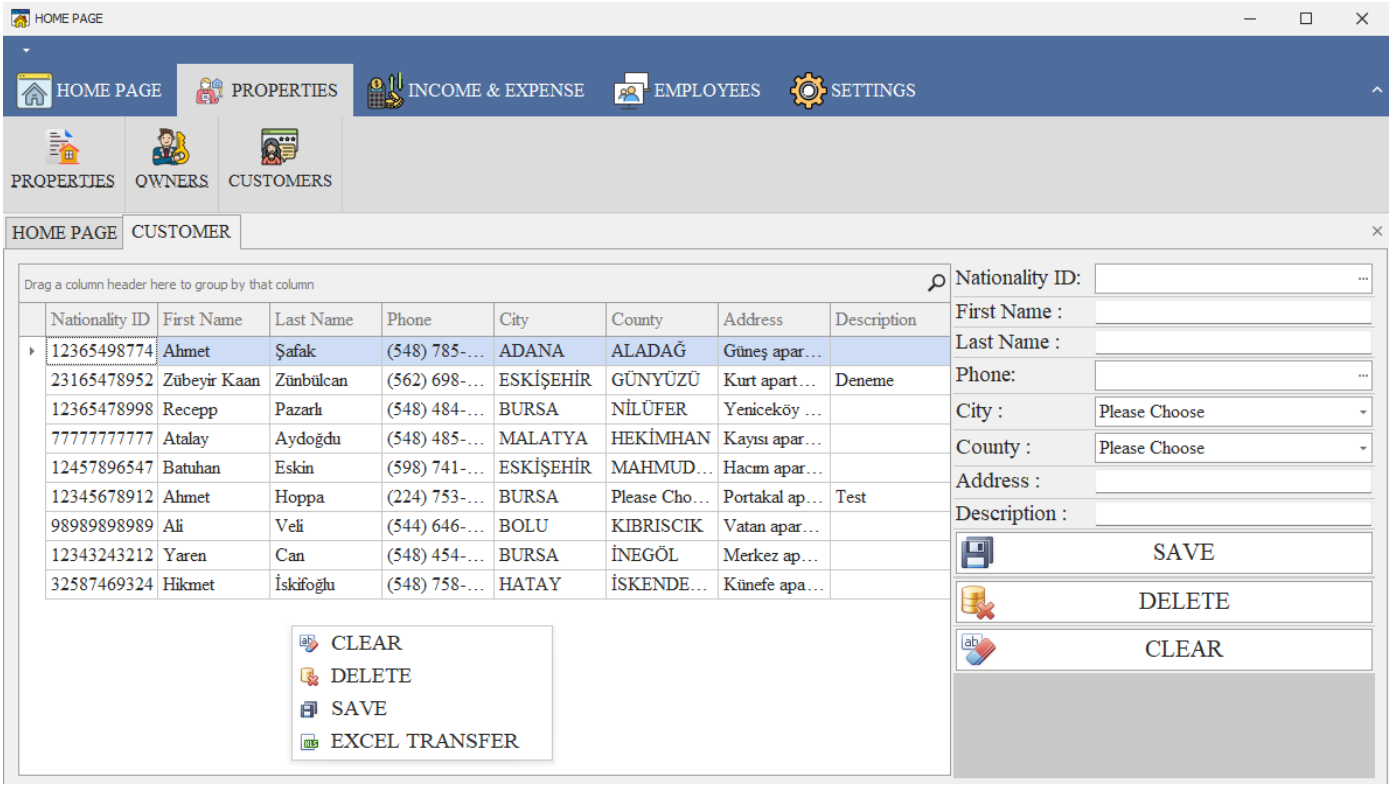

```
1 reference
private void grcAdmin_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Right)
    {
        var position :Point = MousePosition;
        grwAdmin.Focus();
        popupMenu1.ShowPopup(position);
    }
}
```

I added the buttons as shown in the popupMenu from the ribbon, drew them, named them and prepared them. When the mousedown event of the Gridcontrol occurs, I check and if the right click, I focus on that grw and show my popup menu. Images are as in the figures.

The view when right click is as shown in the figure below.



3. Customer Form




The image of the Customer form is as shown in the figure above. The operations in the background are similar to the operations I explained in the admin form. The difference between them is service. In the Customer Form, I prepare all operations from the Customer service structure as functions and call them when necessary. It is also available in this form when I right click. If you are curious about the functions it contains, you can examine the admin section in detail.

```
private readonly ICustomerService _customerService;
```

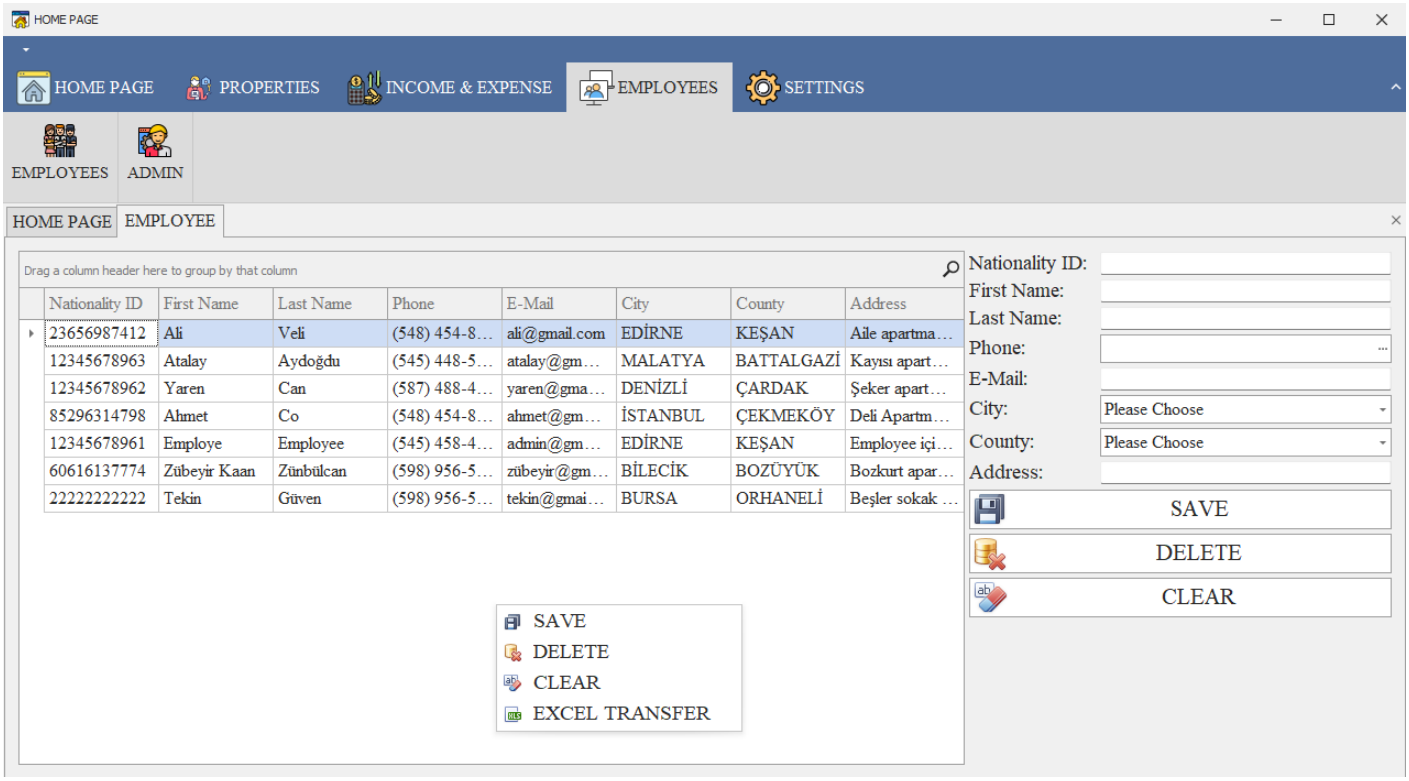
```
1 reference
public CustomerForm()
{
    InitializeComponent();

    _customerService = InstanceFactory.GetInstance<ICustomerService>();
```

```
_customerService.Add(new Customer
{
    NationalityId = btnNationalityId.Text,
    FirstName = txtFirstName.Text,
    LastName = txtLastName.Text,
    Phone = btnPhone.Text,
    City = lkuCity.Text,
    County = lkuCounty.Text,
    Address = txtAddress.Text,
    Description = txtDescription.Text,
    DeleteFlag = false
});
```

```
_customerService.Update(new Customer()
{
    Id = Convert.ToInt32(grwCustomers.GetRowCellValue
        (grwCustomers.FocusedRowHandle, fieldName: "Id")),
    NationalityId = btnNationalityId.Text,
    FirstName = txtFirstName.Text,
    LastName = txtLastName.Text,
    Phone = btnPhone.Text,
    City = lkuCity.Text,
    County = lkuCounty.Text,
    Address = txtAddress.Text,
    Description = txtDescription.Text,
    DeleteFlag = false
});
```

```
_customerService.Update(new Customer()
{
    Id = Convert.ToInt32(grwCustomers. // GridView
        GetRowCellValue(grwCustomers.FocusedRowHandle, fieldName: "Id")),
    NationalityId = btnNationalityId.Text,
    FirstName = txtFirstName.Text,
    LastName = txtLastName.Text,
    Phone = btnPhone.Text,
    City = lkuCity.Text,
    County = lkuCounty.Text,
    Address = txtAddress.Text,
    Description = txtDescription.Text,
    DeleteFlag = true
});
```

4. Employee Form




The image of the Employee form is as shown in the figure above. The operations in the background are similar to the operations I explained in the admin form. The difference between them is service. In the Employee Form, I prepare all operations as functions in my Employee service structure and call them when necessary. It is also available in this form when I right click. If you are curious about the functions it contains, you can examine the admin section in detail.

```
private readonly IEmployeeService _employeeService;
```

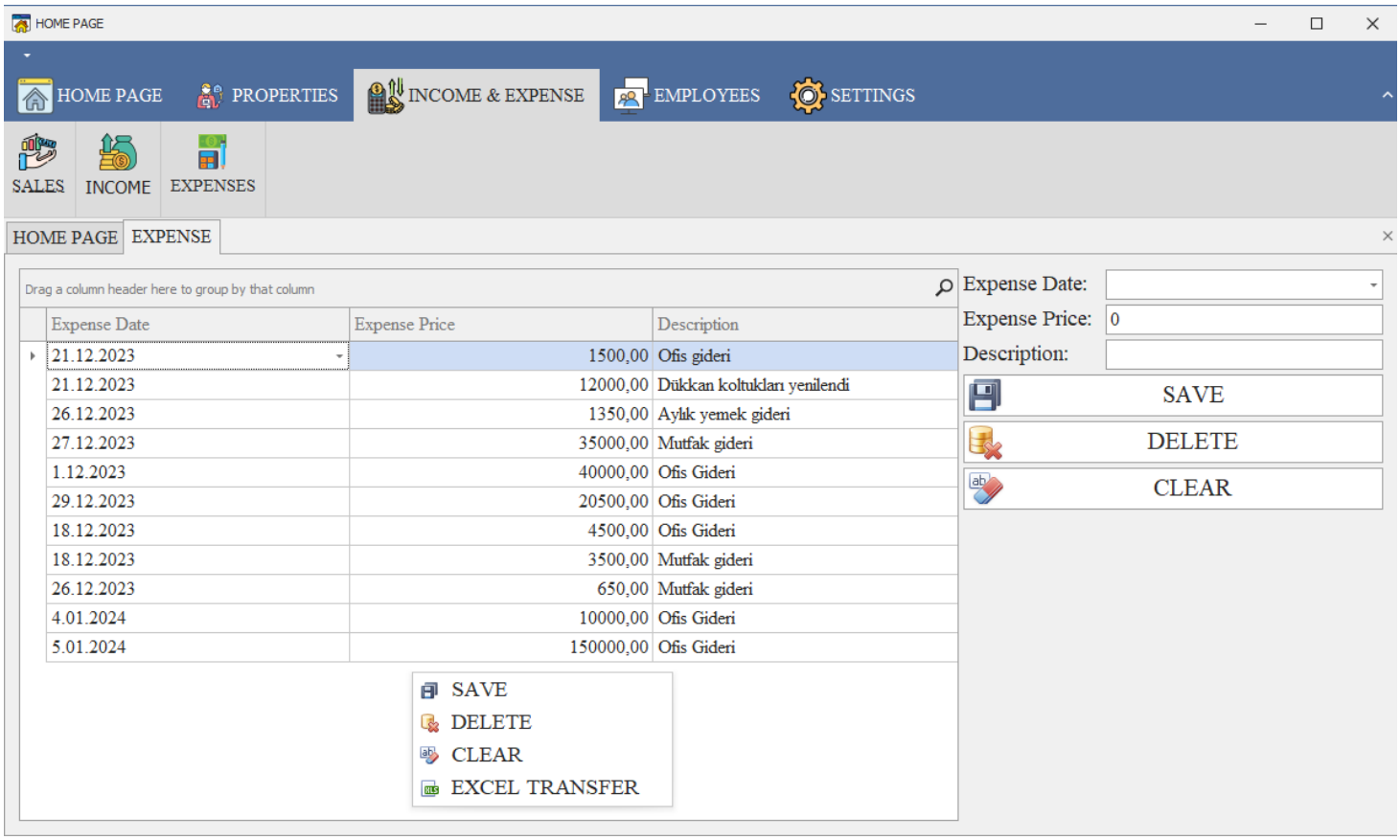
```
InitializeComponent();
_employeeService = InstanceFactory.GetInstance<IEmployeeService>();
```

```
_employeeService.Add(new Employee
{
    UserId = newUserId,
    NationalityId = txtNationalityId.Text,
    FirstName = txtFirstName.Text,
    LastName = txtLastName.Text,
    Phone = btnPhone.Text,
    Email = txtEmail.Text,
    City = lkuCity.Text,
    County = lkuCounty.Text,
    Address = txtAddress.Text,
    DeleteFlag = false
});
```

```
_employeeService.Update(new Employee()
{
    Id = Convert.ToInt32(grwEmployee. // GridView
        GetRowCellValue(grwEmployee.FocusedRowHandle, fieldName: "Id")),
    UserId = Convert.ToInt32(grwEmployee. // GridView
        GetRowCellValue(grwEmployee.FocusedRowHandle, fieldName: "UserId")),
    NationalityId = txtNationalityId.Text,
    FirstName = txtFirstName.Text,
    LastName = txtLastName.Text,
    Phone = btnPhone.Text,
    Email = txtEmail.Text,
    City = lkuCity.Text,
    County = lkuCounty.Text,
    Address = txtAddress.Text,
    DeleteFlag = false
});
```

```
_employeeService.Update2(new Employee
{
    Id = Convert.ToInt32(grwEmployee. // GridView
        GetRowCellValue(grwEmployee.FocusedRowHandle, fieldName: "Id")),
    UserId = Convert.ToInt32(grwEmployee. // GridView
        GetRowCellValue(grwEmployee.FocusedRowHandle, fieldName: "UserId")),
    NationalityId = txtNationalityId.Text,
    FirstName = txtFirstName.Text,
    LastName = txtLastName.Text,
    Phone = btnPhone.Text,
    Email = txtEmail.Text,
    City = lkuCity.Text,
    County = lkuCounty.Text,
    Address = txtAddress.Text,
    DeleteFlag = true
});
```

5. Expense Form




The image of the Expense form is as shown in the figure above. The operations in the background are similar to the operations I explained in the admin form. The difference between them is the service. In the Expense Form, I prepare all operations from the Expense service structure as functions and call them when necessary and when pressed. It is also available in this form when I right click. If you are curious about the functions it contains, you can examine the admin section in detail.

```
private readonly IExpenseService _expenseService;
```

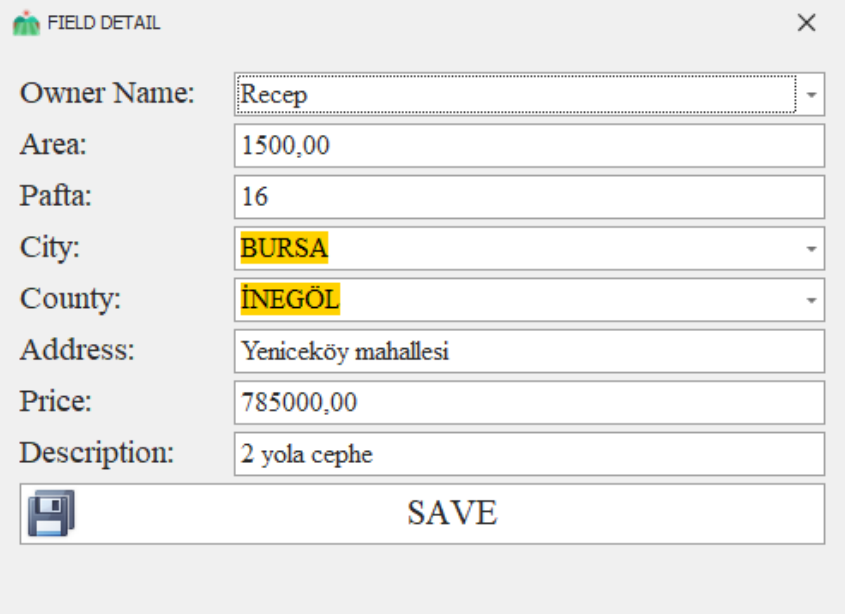
```
InitializeComponent();
_expenseService = InstanceFactory.GetInstance<IExpenseService>();
```

```
_expenseService.Add(new Expense
{
    ExpensePrice = Convert.ToDecimal(txtExpensePrice.Text),
    ExpenseDate = string.IsNullOrWhiteSpace(dteExpenseDate.Text)
        ? (DateTime?)null : Convert.ToDateTime(dteExpenseDate.DateTime.Date),
    Description = txtDescription.Text,
    DeleteFlag = false
});
```

```
_expenseService.Update(new Expense
{
    Id = Convert.ToInt32(grwExpense. // GridView
        GetRowCellValue(grwExpense.FocusedRowHandle, fieldName: "Id")),
    ExpensePrice = Convert.ToDecimal(txtExpensePrice.Text),
    ExpenseDate = string.IsNullOrWhiteSpace(dteExpenseDate.Text)
        ? (DateTime?)null : Convert.ToDateTime(dteExpenseDate.DateTime.Date),
    Description = txtDescription.Text,
    DeleteFlag = true
});
```

```
_expenseService.Update(new Expense
{
    Id = Convert.ToInt32(grwExpense. // GridView
        GetRowCellValue(grwExpense.FocusedRowHandle, fieldName: "Id")),
    ExpensePrice = Convert.ToDecimal(txtExpensePrice.Text),
    ExpenseDate = string.IsNullOrWhiteSpace(dteExpenseDate.Text)
        ? (DateTime?)null : Convert.ToDateTime(dteExpenseDate.DateTime.Date),
    Description = txtDescription.Text,
    DeleteFlag = false
});
```

6. Field Detail Form




The detail screen of my Field Detail table is as shown in the figure. I use this screen as a detail screen for the field that the user double-clicks on in the Property table. In this screen, I enable the data to display the information and update it.

```
5 references
public partial class FieldDetailForm : DevExpress.XtraEditors.XtraForm
{


    1 reference
    public FieldDetailForm()
    {
        InitializeComponent();
        _fieldService = InstanceFactory.GetInstance<IFieldService>();
        _ownerService = InstanceFactory.GetInstance<IOwnerService>();
        _cityService = InstanceFactory.GetInstance<ICityService>();
        _countyService = InstanceFactory.GetInstance<ICountyService>();
    }

    private static IFieldService _fieldService;
    private readonly IOwnerService _ownerService;
    private readonly ICityService _cityService;
    private readonly ICountyService _countyService;


    1 reference
    private void FieldDetailForm_Load(object sender, EventArgs e)
    {
        LoadOwner();
        LoadCity();
        LoadCounty();
        LoadField();
        txtId.Text = ItemId.ToString();

    }

    public int ItemId;
```

In the figure, I call the necessary Service structures and create sample objects. I call my methods and use functions while the form is loading. Finally, I assign my itemId variable to the txtid field and transfer it to text. The purpose of my ItemId variable is to capture and send the Id of the property double-clicked line at the moment the variable is clicked.

```
2 references
void LoadField()
{
    using (RealEstateAutomationContext context = new RealEstateAutomationContext())
    {
        var entity:IQueryable<{Id,PropertyId,…}> = from f:Field in context.Fields
                    join p:Property in context.Properties on f.PropertyId equals p.Id
                    join o:Owner in context.Owners on f.OwnerId equals o.Id
                    join ci:City in context.Cities on f.City equals ci.Id
                    join co:County in context.Counties on f.County equals co.Id
                    select new
                    {
                        Id = f.Id,
                        PropertyId = f.PropertyId,
                        OwnerId = f.OwnerId,
                        Area = f.Area,
                        Pafta = f.Pafta,
                        City = ci.CityName,
                        County = co.CountyName,
                        Address = f.Address,
                        Price = f.Price,
                        Description = f.Description,
                        DeleteFlag = f.DeleteFlag,
                    };

        var field:IEnumerable<{Id,PropertyId,…}> = entity.ToList().Where(x:{Id,PropertyId,…} => x.Id == ItemId);

        var current:{Id,PropertyId,…} = field.FirstOrDefault(x:{Id,PropertyId,…} => x.Id == ItemId);
        if (current != null)
        {
            txtId.Text = current.Id.ToString();

            txtPropertyType.Text = current.PropertyId.ToString();

            lkuOwnerId.EditValue = current.OwnerId;
            lkuCity.Text = current.City.ToString();
            lkuCounty.Text = current.County.ToString();

            txtArea.Text = current.Area.ToString();
            txtPafta.Text = current.Pafta;
            txtAddress.Text = current.Address;
            txtPrice.Text = current.Price.ToString();
            txtDescription.Text = current.Description;
        }
        else
        {
            MessageBox.Show(text:@"There was an error loading the information. Please try again."
                , caption:@"Information", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

Due to necessity, I added a new item to my load field process, created a new entity in this field, performed the necessary join operations for this entity, and set the data as a new object. Then, using the ItemId variable from my Property field, I pulled the field data with that id into my variable. Then, if the current I upload to the id below is not empty, I make the necessary uploads into the texes. If there is a problem during the installation, I return an error and inform the user.

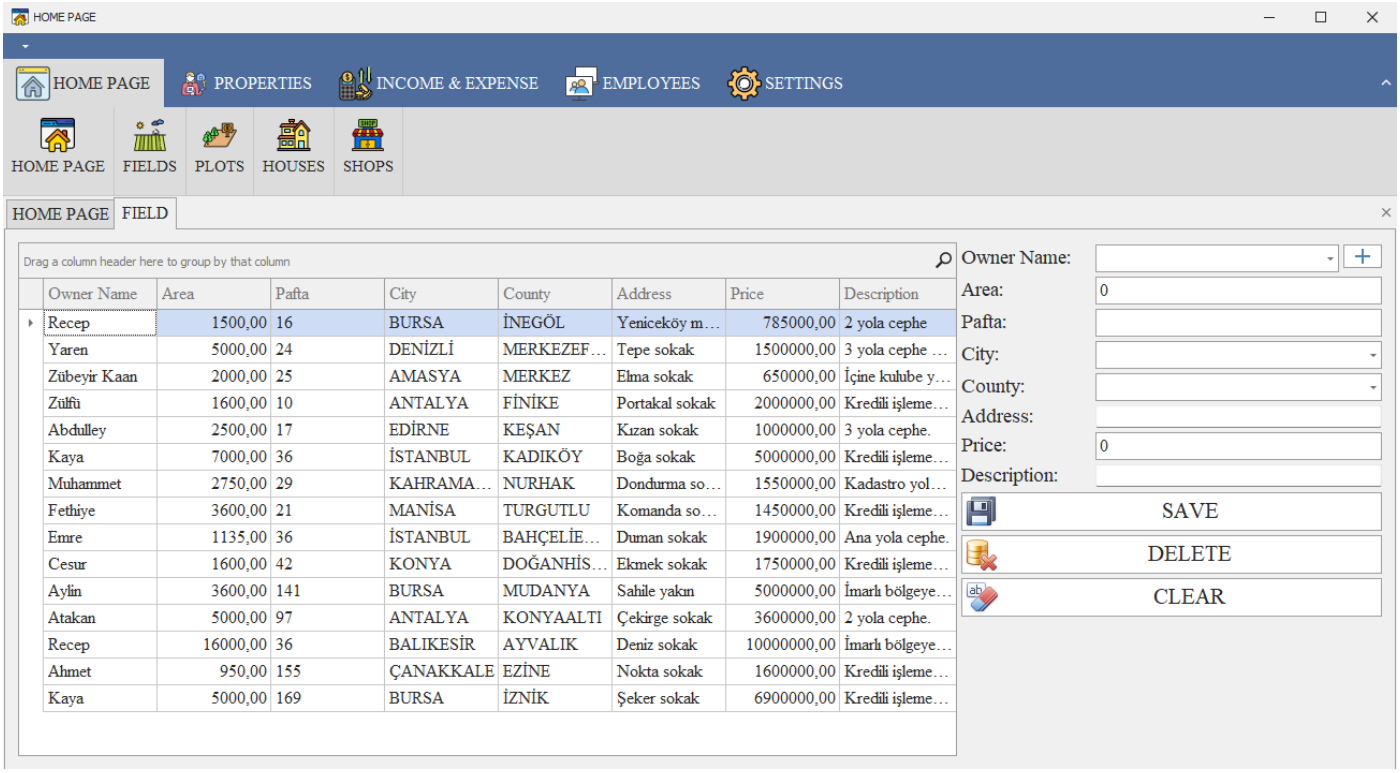
```
1 reference
void LoadOwner()
{
    lkuOwnerId.Properties.DataSource = _ownerService.GetAll();
    lkuOwnerId.Properties.DisplayMember = "FirstName";
    lkuOwnerId.Properties.ValueMember = "Id";
}

1 reference
private void LoadCity()
{
    lkuCity.Properties.DataSource = _cityService.GetAll();
    lkuCity.Properties.DisplayMember = "CityName";
    lkuCity.Properties.ValueMember = "Id";
}


2 references
private void LoadCounty()
{
    lkuCounty.Properties.DataSource = _countyService.GetAll(key:Convert.ToInt32(lkuCity.EditValue)).ToList();
    lkuCounty.Properties.DisplayMember = "CountyName";
    lkuCounty.Properties.ValueMember = "Id";
}

1 reference
private void lkuCity_EditValueChanged(object sender, EventArgs e)
{
    LoadCounty();
}
```

My uploading process to look up edits is as shown.

```
1 reference
void Save()
{
    if (txtId.Text != "")
    {
        DialogResult confirmation = MessageBox.Show(text:@"Are you sure you want to update the information?",
            caption:@"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

        if (confirmation == DialogResult.Yes)
        {
            try
            {
                _fieldService.Update(new Field
                {
                    Id = Convert.ToInt32(txtId.Text),
                    PropertyId = Convert.ToInt32(txtPropertyType.Text),
                    OwnerId = Convert.ToInt32(lkuOwnerId.EditValue),
                    Area = Convert.ToDecimal(txtArea.Text),
                    Pafta = txtPafta.Text,
                    City = Convert.ToInt32(lkuCity.EditValue),
                    County = Convert.ToInt32(lkuCounty.EditValue),
                    Address = txtAddress.Text,
                    Price = Convert.ToDecimal(txtPrice.Text),
                    Description = txtDescription.Text,
                    Sold = false,
                    DeleteFlag = false
                });
                LoadField();
            }
            catch (Exception)
            {
                MessageBox.Show(text:@"Incorrect data entry, Please fill in the missing fields ",
                    caption:@"Information", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
            }

        }
        else
        {
            MessageBox.Show(text:@"Your transaction has been canceled.",
                caption:@"Information", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }
    }

}

1 reference
private void btnSave_Click(object sender, EventArgs e)
{
    Save();
}
```

The recording method is as shown in the figure. I take the necessary data from the texts and perform the registration process. In case of an error, I send a message to the user and show the error. Finally, I call the function when the button is clicked.

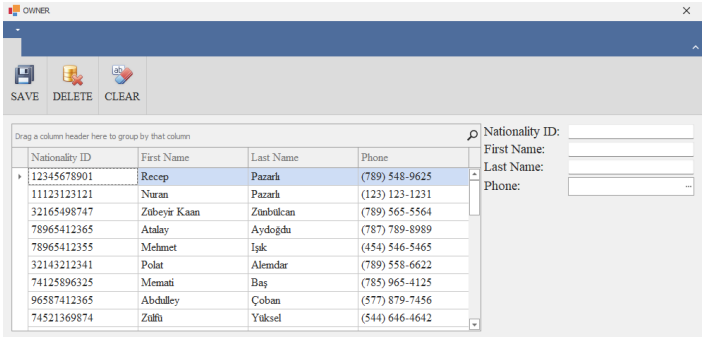
7. Field Form




The appearance of the Field form is as shown in the figure above. The operations in the background are similar to the operations I explained in the admin form. The difference between them is service. In the Field Form, I prepare all operations from the Field service structure as functions and call them when necessary. It is also available in this form when I right click. If you are curious about the functions it contains, you can examine the admin section in detail.

```
private readonly IFieldService _fieldService;
```

```
InitializeComponent();
_fieldService = InstanceFactory.GetInstance<IFieldService>();
```




I have an extra plus button in this area, next to the owner textbox. When I click it, I open the Owner form as shown in the figure, allowing the user to easily add it from there. When the form is closed after the addition is made, I automatically load lookupedit again and show the newly added owner to the user.

```
2 references
void Save()
{
    if (txtId.Text == "")
    {
        DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to save the information?"
            , caption: @"Information", MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);

        if (confirmation == DialogResult.Yes)
        {
            _propertyService.Add(new Property
            {
                PropertyType = "Field",
                DeleteFlag = false
            });

            Property lastAddedProperty = _propertyService.GetLastAddedProperty();
            int newPropertyId = 0;
            newPropertyId = lastAddedProperty?.Id ?? -1;


            if (newPropertyId != -1 && newPropertyId != 0)
            {
                try
                {
                    _fieldService.Add(new Field
                    {
                        OwnerId = Convert.ToInt32(lkuOwnerId.EditValue),
                        PropertyId = Convert.ToInt32(newPropertyId),
                        Area = Convert.ToDecimal(txtArea.Text),
                        Pafta = txtPafta.Text,
                        City = Convert.ToInt32(lkuCity.EditValue),
                        County = Convert.ToInt32(lkuCounty.EditValue),
                        Address = txtAddress.Text,
                        Price = Convert.ToDecimal(txtPrice.Text),
                        Description = txtDescription.Text,
                        Sold = false,
                        DeleteFlag = false
                    });

                    Field lastAddedField = _fieldService.GetLastAddedField();
                    int newFieldId = 0;
                    newFieldId = lastAddedField?.Id ?? -1;

                    if (newFieldId != -1 && newFieldId != 0)
                    {
                        _propertyService.Update(new Property
                        {
                            Id = newPropertyId,
                            ReferenceId = newFieldId,
                            PropertyType = "Field",
                            DeleteFlag = false
                        });
                    }
                }
                catch (Exception e)
                {
                    _propertyService.Delete(new Property
                    {
                        Id = newPropertyId
                    });

                    MessageBox.Show(
                         text: e.InnerException == null ? e.Message : "This record already exists please check your details",
                           caption: @"Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
            LoadField();
            Clear();
        }
```

```
else if (txtId.Text != "")
{
    DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to update the information?",
        caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (confirmation == DialogResult.Yes)
    {
        _fieldService.Update(new Field
        {
            Id = Convert.ToInt32(grwField.GetRowCellValue(grwField.FocusedRowHandle, fieldName: "Id")),
            PropertyId = Convert.ToInt32(txtPropertyType.Text),
            OwnerId = Convert.ToInt32(lkuOwnerId.EditValue),
            Area = Convert.ToDecimal(txtArea.Text),
            Pafta = txtPafta.Text,
            City = Convert.ToInt32(lkuCity.EditValue),
            County = Convert.ToInt32(lkuCounty.EditValue),
            Address = txtAddress.Text,
            Price = Convert.ToDecimal(txtPrice.Text),
            Description = txtDescription.Text,
            Sold = false,
            DeleteFlag = false
        });
        LoadField();
        Clear();

    }
    else
    {
        MessageBox.Show(text: @"Your transaction has been canceled.", caption: @"Information", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}
```

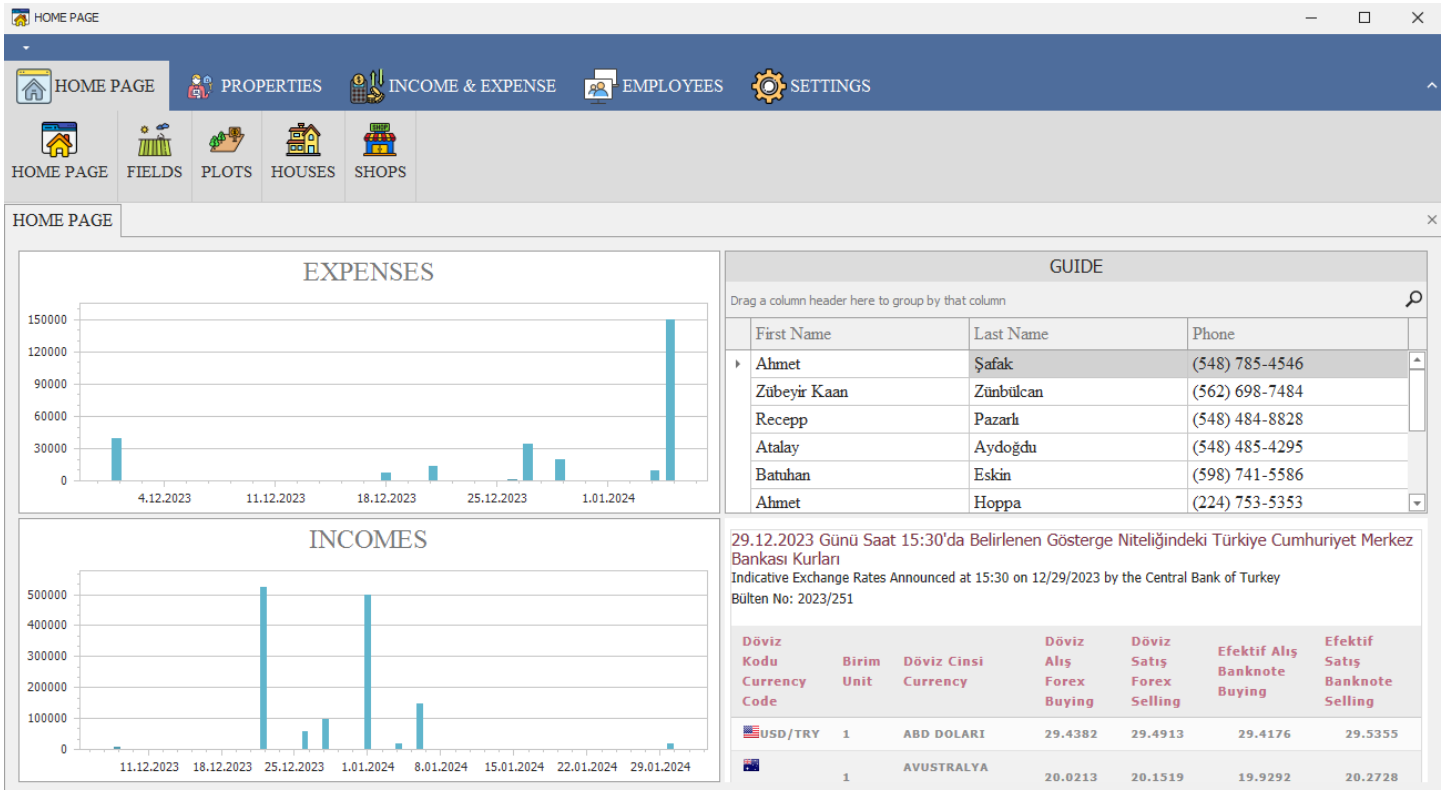
The operations in my Save, Update and Delete method are as shown in the figure. While doing the save operation, I first create a property object. I capture the id of the object I created, assign it to the variable, and save it in the propertyid field of the field table when adding the Field. Then I capture the added field id field and assign it to a variable. Then I update the Property and add the added filed id into it. This is how I complete my method. In case of an error, I catch the error, return it to the user, and prevent complexity by deleting the created property from the database. In my update function, I take the necessary updates from the texts and update the data. In the delete function, while updating the Field, I mark deleteflag as true, then set deleteflag as true in the property field and leave it deleted in the background.

```
void Remove()
{
    DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to update the information?",
        caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (confirmation == DialogResult.Yes)
    {
        try
        {
            _fieldService.Update(new Field
            {
                Id = Convert.ToInt32(grwField.GetRowCellValue(grwField.FocusedRowHandle, fieldName: "Id")),
                PropertyId = Convert.ToInt32(grwField.GetRowCellValue(grwField.FocusedRowHandle, fieldName: "PropertyId")),
                OwnerId = Convert.ToInt32(lkuOwnerId.EditValue),
                Area = Convert.ToDecimal(txtArea.Text),
                Pafta = txtPafta.Text,
                City = Convert.ToInt32(lkuCity.EditValue),
                County = Convert.ToInt32(lkuCounty.EditValue),
                Address = txtAddress.Text,
                Price = Convert.ToDecimal(txtPrice.Text),
                Description = txtDescription.Text,
                Sold = false,
                DeleteFlag = true
            });

            _propertyService.Update(new Property
            {
                Id = Convert.ToInt32(grwField.GetRowCellValue(grwField.FocusedRowHandle, fieldName: "PropertyId")),
                ReferenceId = Convert.ToInt32(grwField.GetRowCellValue(grwField.FocusedRowHandle, fieldName: "Id")),
                PropertyType = "Field",
                DeleteFlag = true
            });

            LoadField();
            Clear();

        }
        catch (Exception ex)
        {
            MessageBox.Show(
                text: ex.InnerException == null ? ex.Message : "This record already exists please check your details",
                caption: @"Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

    }
    else
    {
        MessageBox.Show(text: @"Your transaction has been canceled.", caption: @"Information", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}
```

```
private OwnerForm _ownerForm = new OwnerForm();
1 reference
private void btnOwnerAdd_Click(object sender, EventArgs e)
{

    _ownerForm.ribbonControl1.Visible = true;
    _ownerForm.lcSave.Visibility = LayoutVisibility.Never;
    _ownerForm.lcDelete.Visibility = LayoutVisibility.Never;
    _ownerForm.lcClear.Visibility = LayoutVisibility.Never;
    _ownerForm.ShowDialog();
    LoadOwner();
}
```

After making the necessary checks on the function of the Add Owner button, I open the form as showdialog and when it is closed, I upload it to the grid control again.

8. Home Page2 Form




On this screen, I made some additions to the home page and created the home page layout by offering ease of viewing and statistics to the user.

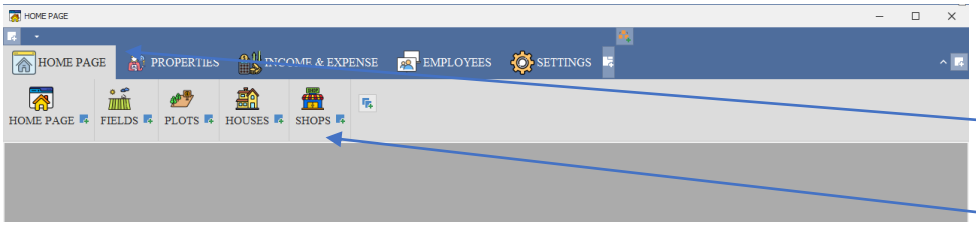
```
1 reference
void LoadChartExpense()
{
    using (var context = new RealEstateAutomationContext())
    {
        var query :IQueryable<(ExpenseDate,TotalExpense)> = from e :Expense in context.Expenses
                    where e.DeleteFlag == false
                    group e by e.ExpenseDate
                into g :IGrouping<DateTime?,Expense>
                    select new
                    {
                        ExpenseDate = g.Key,
                        TotalExpense = g.Sum(x :Expense => x.ExpensePrice)
                    };

        var expenseList = query.ToList();

        if (chartControlExpense.Series["Expenses"] == null)
        {
            chartControlExpense.Series.Add(new Series( name: "Expenses", ViewType.Bar));
        }

        Series series = chartControlExpense.Series["Expenses"];

        foreach (var e :(ExpenseDate,TotalExpense) in expenseList)
        {
            series.Points.Add(new SeriesPoint( argument: Convert.ToDateTime(e.ExpenseDate),
                params values: Convert.ToDouble(e.TotalExpense)));
        }
    }
}
```

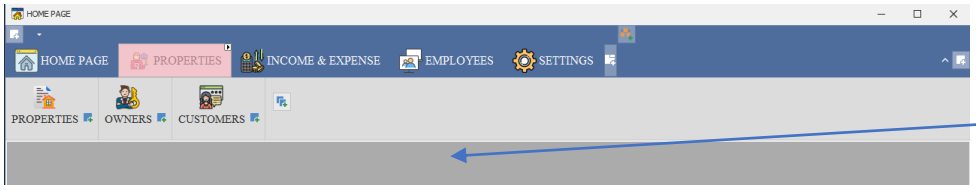
```
1 reference
void LoadChartIncome()
{
    using (var context = new RealEstateAutomationContext())
    {
        var query :IQueryable<(IncomeDate,TotalIncome)> = from i :Income in context.Incomes
                    where i.DeleteFlag == false
                    group i by i.IncomeDate
                into g :IGrouping<DateTime?,Income>
                    select new
                    {
                        IncomeDate = g.Key,
                        TotalIncome = g.Sum(x :Income => x.IncomePrice)
                    };

        var incomeList = query.ToList();

        if (chartControlIncome.Series["Incomes"] == null)
        {
            chartControlIncome.Series.Add(new Series( name: "Incomes", ViewType.Bar));
        }

        Series series = chartControlIncome.Series["Incomes"];

        foreach (var e :(IncomeDate,TotalIncome) in incomeList)
        {
            series.Points.Add(new SeriesPoint( argument: Convert.ToDateTime(e.IncomeDate),
                params values: Convert.ToDouble(e.TotalIncome)));
        }
    }
}
```

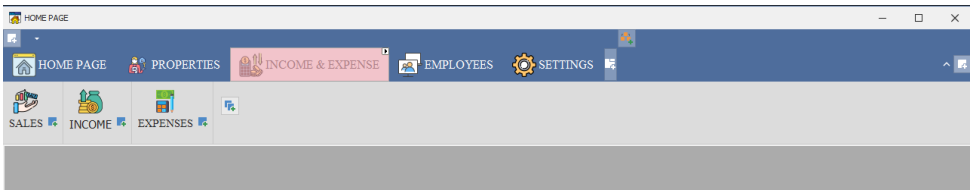
```
1 reference
void LoadCurrency()
{
    webBrowser1.Navigate( urlString: "https://tcmb.gov.tr/kurlar/today.xml");
}

1 reference
private void LoadCustomer()
{
    grcCustomer.DataSource = _customerService.GetAll();
}
```

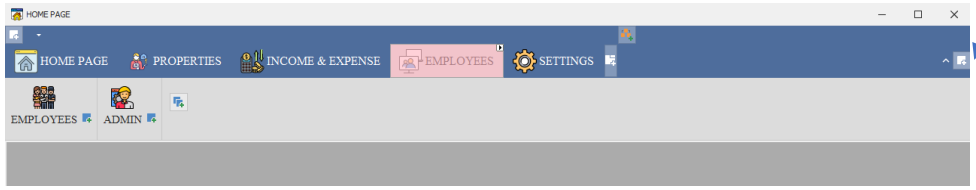
For the home page, I call the functions from my service structures as shown in the figures and call them while the screen is loading.

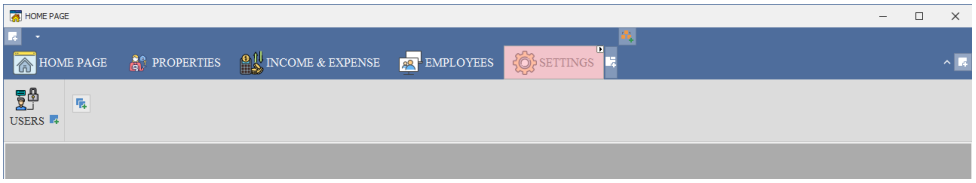
9. Home Page Form




My Homepage Form is as shown in the figures. I created ribbon pages and created pages according to their fields, and inside the ribbon pages, I created a ribbonpagegroup and added buttons to them. When the button is clicked, I upload the desired form to the mdi parent field that I placed inside the form.

```
14 references
public void OpenForm<T>() where T : Form, new()
{
    // MDI çocuk formlar arasında belirtilen form türünü ara
    foreach (Form form in this.MdiChildren)
    {
        if (form is T)
        {
            // Form zaten açıksa ona odaklan ve metoddan çık
            form.Activate();
            return;
        }
    }

    // Form açık değilse yeni bir örnek oluştur ve göster
    T newForm = new T
    {
        MdiParent = this
    };
    newForm.Show();
}
```

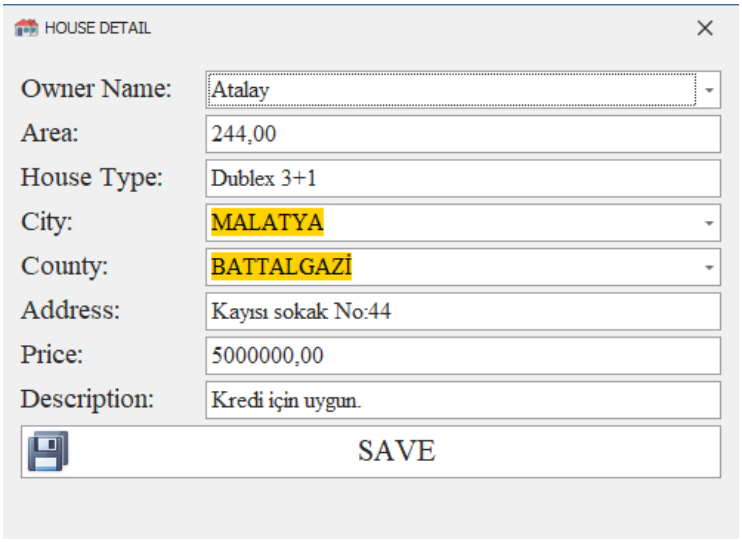
I wrote a generic function as shown in the figure to open the forms in the mdi parent. I send the form I want to open into it and perform the operation. If the form is open, I do not have it opened again. If it is closed, the form can be opened.

```
1 reference
private void btnCustomer_ItemClick(object sender, DevExpress.XtraBars.ItemClickEventArgs e)
{
    OpenForm<CustomerForm>();
}
```

```
1 reference
public void PanelControl()
{
    try
    {
        if (LoginForm.EnteredUserAuthorization == 1)
        {
            ribbonPageEmployees.Visible = true;
            ribbonPageGroupProperties.Visible = true;
        }
        else if (LoginForm.EnteredUserAuthorization == 2)
        {
            ribbonPageEmployees.Visible = false;
            ribbonPageGroupProperties.Visible = false;
        }
    }
    catch (Exception)
    {
        MessageBox.Show(text: @"Error Loading System. Please try again.", caption: @"Information",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }


}
```

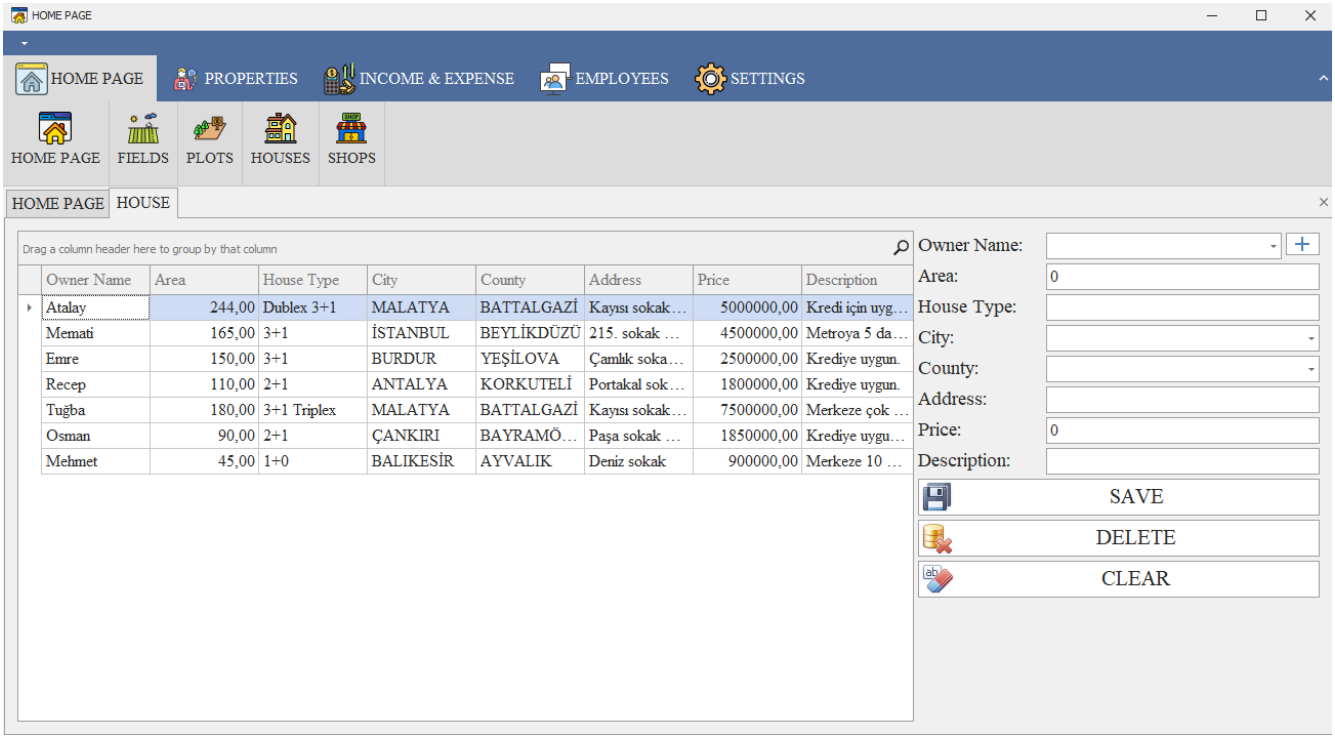
I wrote a method that performs some checks based on authorization in the panel control process. I call this method while the form is loading.

10. House Detail Form



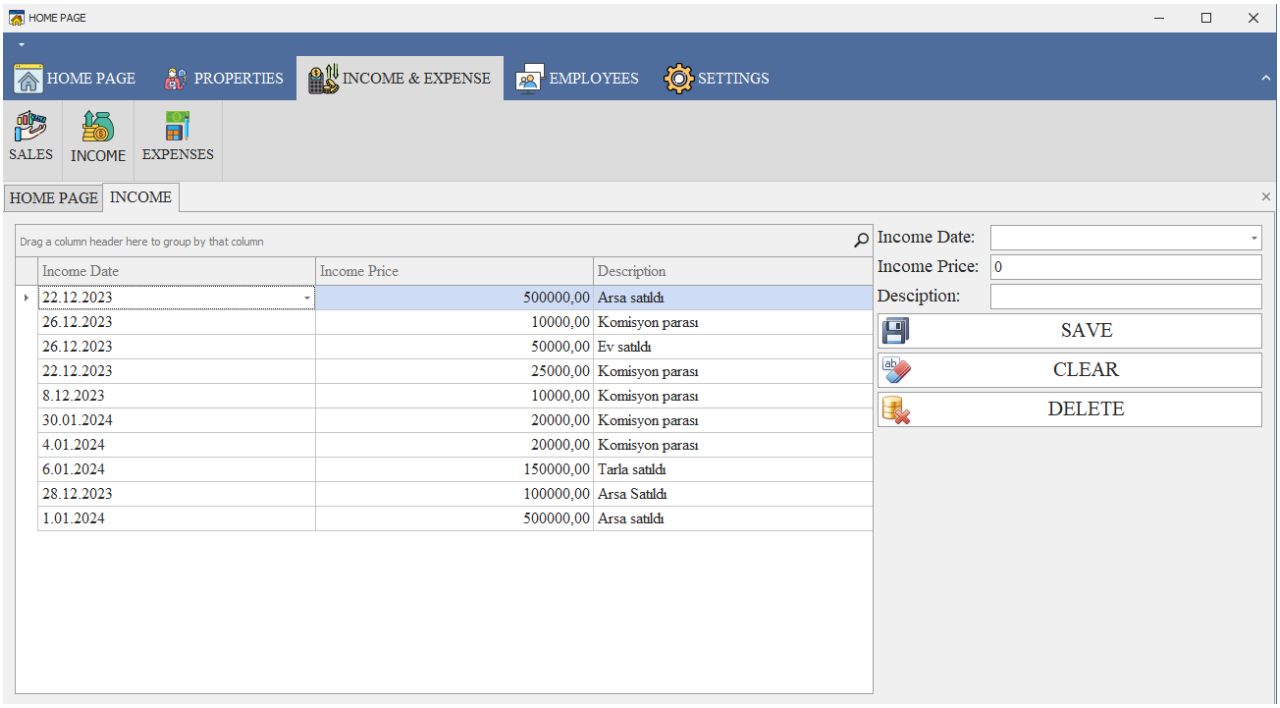

My House Detail form is as shown. The procedures are as in the Field Detail Form. The only difference between them is the Service structure. If you want to make a detailed review, you can review the Field Detail Form section above.

11. House Form



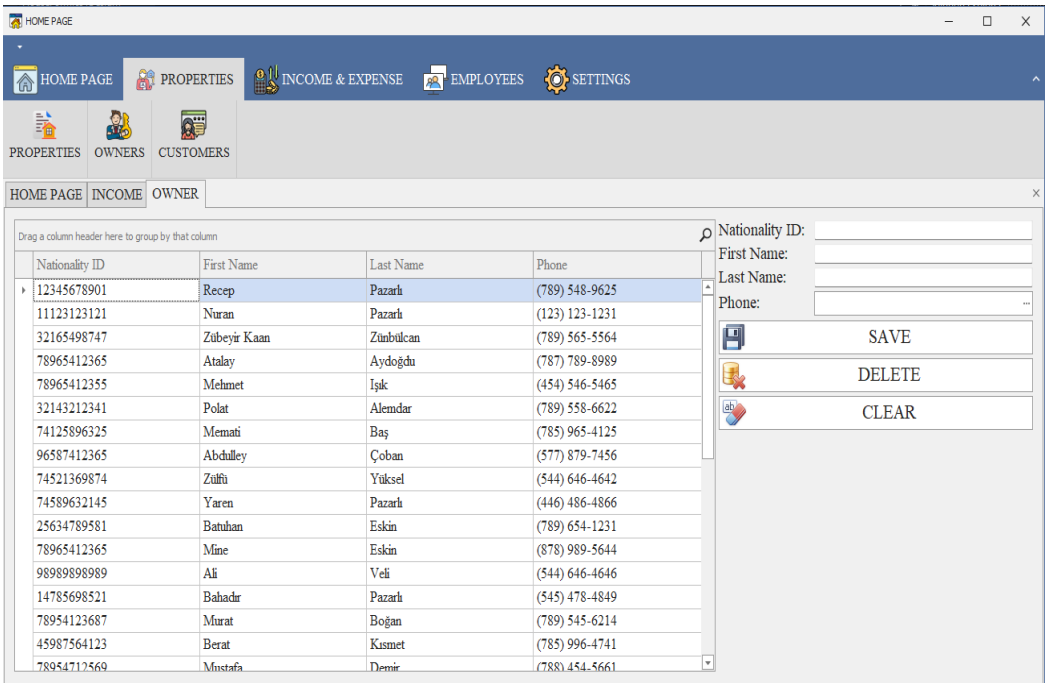

House Form is as shown in the figure above. The operations in the background are similar to the operations performed in the Field Form, the only difference between them is the service structures. If you want to make a detailed review, you can examine the Field Form field above.

12. Income Form




The image of the Income form is as shown in the figure above. The operations in the background are similar to the operations I explained in the admin form. The difference between them is the service. In the Income Form, I prepare all operations from the Income service structure as functions and call them when necessary and when pressed. It is also available in this form when I right click. If you are curious about the functions it contains, you can examine the admin section in detail.

13. Owner Form




The image of the Owner form is as shown in the figure above. The operations in the background are similar to the operations I explained in the admin form. The difference between them is the service. In the Owner Form, I prepare all operations from the Owner service structure as functions and call them when necessary and when pressed. It is also available in this form when I right click. If you are curious about the functions it contains, you can examine the admin section in detail.

14. Login Form


The design of my login screen is as shown in the figure below.

AuthorizationController as in the figures. According to the incoming data, I check the username and password.

```
7 references
public partial class LoginForm : DevExpress.XtraEditors.XtraForm
{
    1 reference
    public LoginForm()
    {
        InitializeComponent();
        _userService = InstanceFactory.GetInstance<IUserService>();
    }

    private readonly IUserService _userService;
    public static int EnteredUserId;
    public static int EnteredUserAuthorization;
    public string EnteredUserName;
    public string EnteredUserPassword;

    2 references
    private void OpenForm(Form form)
    {

        form.Show();
    }
```

For the login form, I add the Service structure to my system and create its object. Then, I capture all the data of the logged in user and assign it to the functions to use in the background.

```
1 reference
private bool AuthorizationControl1(string userName, string password, int authorization)
{
    var query :IEnumerable<User> = _userService.GetAll().Where(x :User =>
        x.UserName == userName && x.UserPassword == password && x.UserAuthorization == 1);

    if (query.Any())
    {
        return true;
    }
    else
    {
        return false;
    }
}

1 reference
private bool AuthorizationControl2(string userName, string password, int authorization)
{
    var query :IEnumerable<User> = _userService.GetAll().Where(x :User =>
        x.UserName == userName && x.UserPassword == password && x.UserAuthorization == 2);

    if (query.Any())
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

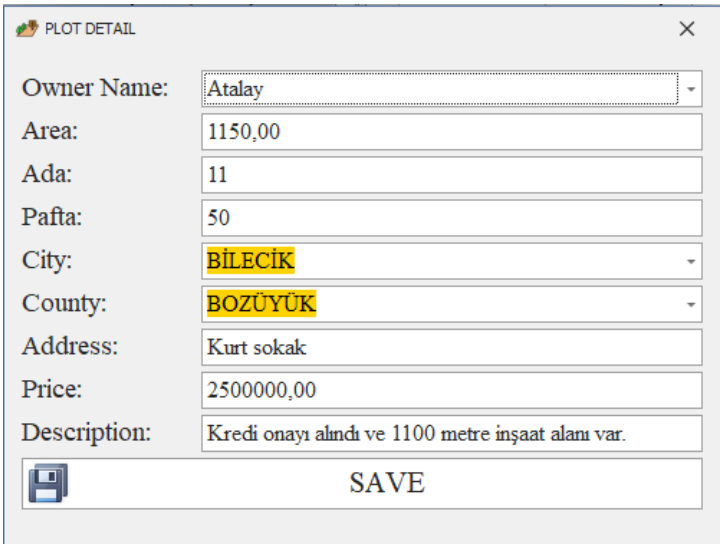
```
1 reference
private void btnLogin_Click(object sender, EventArgs e)
{
    if (txtUserName.Text == "" && txtPassword.Text == "")
    {
        MessageBox.Show(text: @"Username and Password cannot be empty.", caption: @"Information",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else if (txtUserName.Text == "")
    {
        MessageBox.Show(text: @"Username cannot be empty.", caption: @"Information",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else if (txtPassword.Text == "")
    {
        MessageBox.Show(text: @"Password cannot be empty.", caption: @"Information",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        var enteredUser = _userService.GetAll() // List<User>
            .FirstOrDefault(x :User => x.UserPassword == txtPassword.Text && x.UserName == txtUserName.Text);

        if (enteredUser != null)
        {
            EnteredUserId = enteredUser.Id;
            EnteredUserAuthorization = enteredUser.UserAuthorization;
            EnteredUserName = enteredUser.UserName;
            EnteredUserPassword = enteredUser.UserPassword;
        }

        if (AuthorizationControl1(txtUserName.Text, txtPassword.Text, authorization: 1))
        {
            OpenForm(new HomePageForm());
            this.Hide();
        }
        else if (AuthorizationControl2(txtUserName.Text, txtPassword.Text, authorization: 2))
        {
            OpenForm(new HomePageForm());
            this.Hide();
        }
        else
        {
            MessageBox.Show(text: @"Username or password is incorrect, please try again.", caption: @"Information",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }
}
```

Login button operations are as shown. I check the texts and show them to the user with a message depending on their status. If all the texts are successful, I enter the information for the logged in user and assign that information to variables in the background. Afterwards, I open the form according to the Authorization procedures. If there is a failure, I show this to the user again with a message.

```
1 reference
private void btnQuit_Click(object sender, EventArgs e)
{
    DialogResult Confirmation = MessageBox.Show(text: @"Are you sure you want to close?",
        caption: @"Information", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (Confirmation == DialogResult.Yes)
    {
        this.Close();
    }
}
```

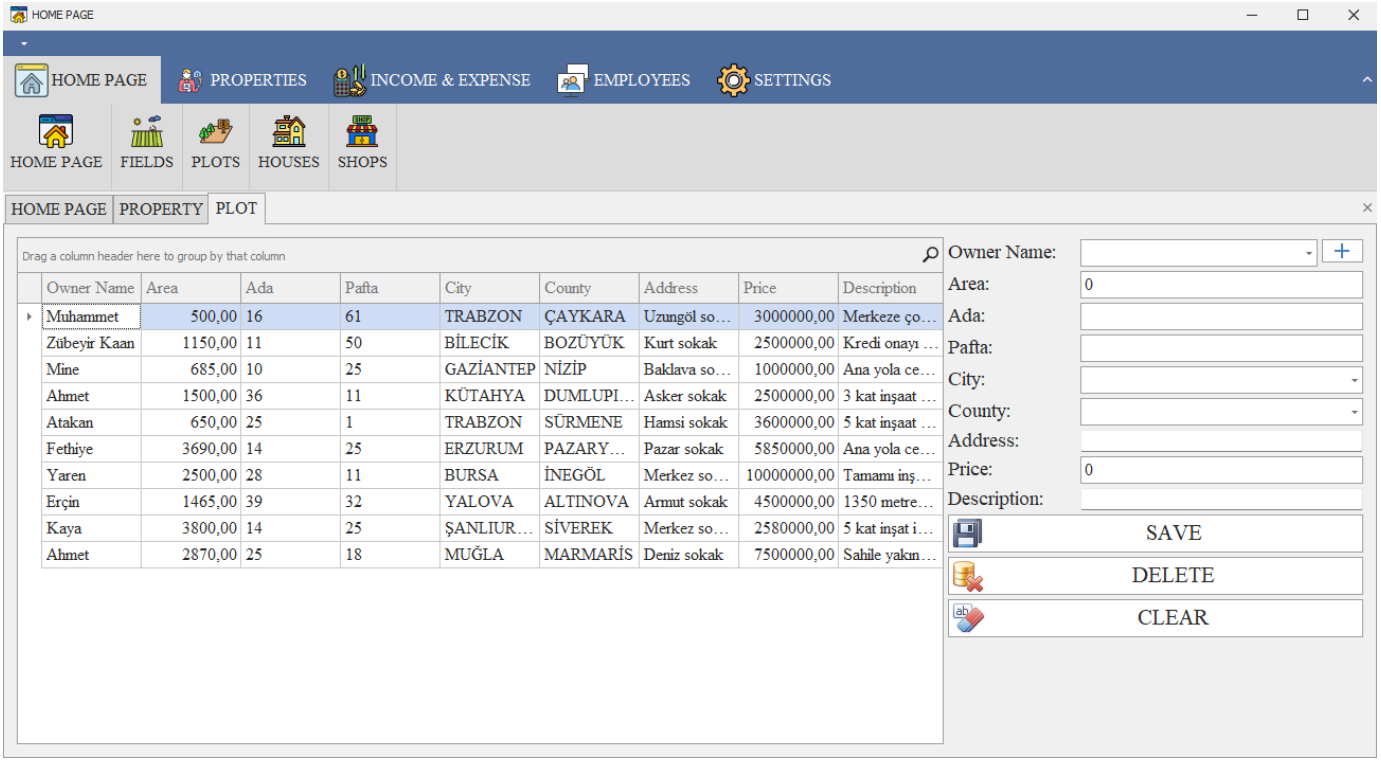
My Quit button operations are as shown in the figure. If it is successful, I close the form directly.

15. Plot Detail Form



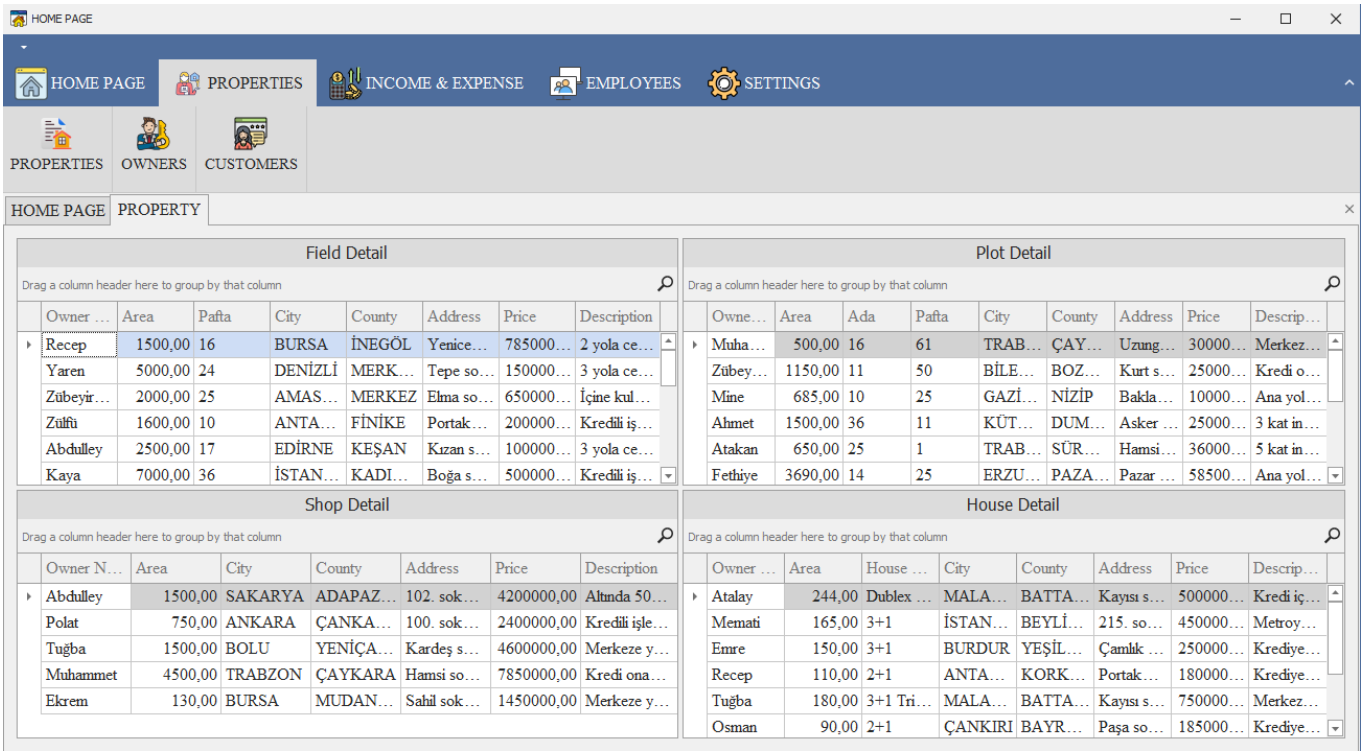

My Plot Detail form is as shown. The procedures are as in the Field Detail Form. The only difference between them is the Service structure. If you want to make a detailed review, you can review the Field Detail Form section above.

16. Plot Form




Plot Form is as shown in the figure above. The operations in the background are similar to the operations performed in the Field Form, the only difference between them is the service structures. If you want to make a detailed review, you can examine the Field Form field above.

17. Property Form



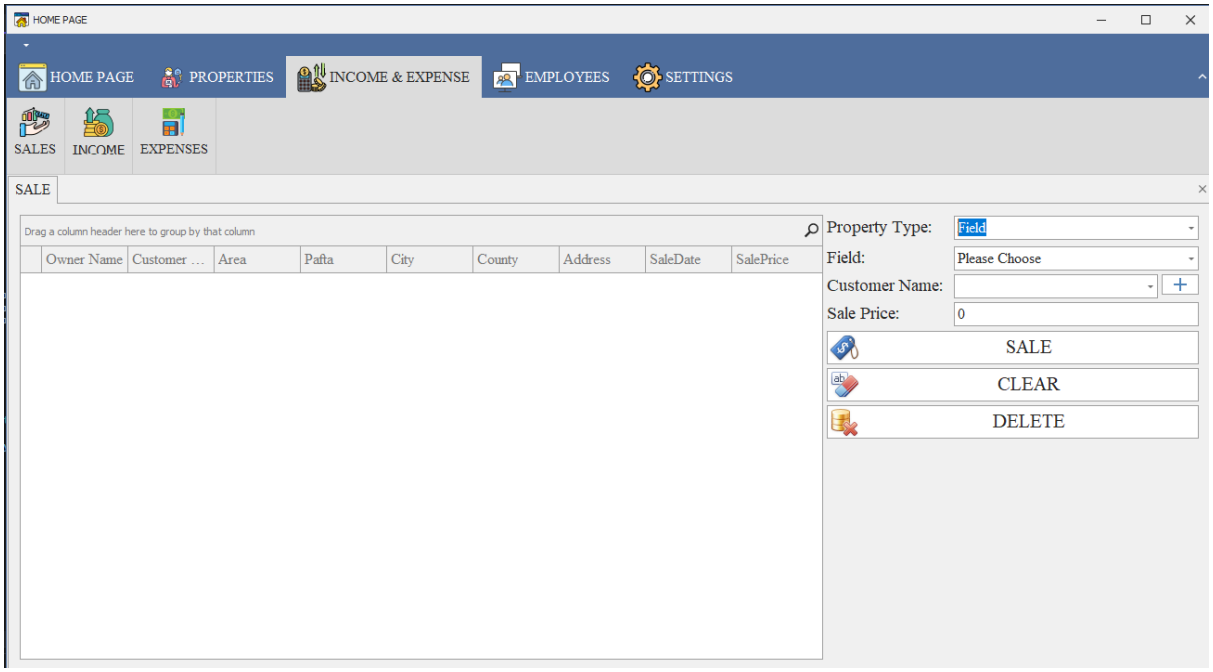

My Property Form screen is as shown above. In this form, I upload the forms to the transaction grid controller and enable all properties to be displayed on a single screen.

```
0 references
public PropertyForm()
{
    InitializeComponent();

    _fieldService = InstanceFactory.GetInstance<IFieldService>();
    _houseService = InstanceFactory.GetInstance<IHouseService>();
    _plotService = InstanceFactory.GetInstance<IPlotService>();
    _shopService = InstanceFactory.GetInstance<IShopService>();
}


private readonly IHouseService _houseService;
private readonly IPlotService _plotService;
private readonly IShopService _shopService;
private readonly IFieldService _fieldService;

1 reference
private void PropertyForm_Load(object sender, EventArgs e)
{
    LoadField();
    LoadShop();
    LoadHouse();
    LoadPlot();
}
```

I called the necessary service structures, prepared my functions with GetAll methods and called them during loading. When the GridControls were double-clicked, I captured the id of the property in the clicked row and sent it to the detail form. In this way, I ensured that the data was displayed in detail in the detail form. You can review the Field Detail Form field above for this detail form image and operations.

```
1 reference
private void grcField_DoubleClick(object sender, EventArgs e)
{

    FieldDetailForm fieldDetailForm = new FieldDetailForm();
    fieldDetailForm.ItemId = Convert.ToInt32(grwField.GetRowCellValue(grwField.FocusedRowHandle, fieldName: "Id"));
    fieldDetailForm.ShowDialog();
    LoadField();
}
```

18. Sale Form




My sale form screen is as shown in the figure above. I load grid controls according to the Property type selected in the form and control them in the background. I put the Field Image here, the same image occurs for other property types.

```
1 reference
void LoadClick()
{
    switch (cmbPropertyType.Text)
    {
        case "Field":

            LoadLookUpFieldClick();

            if (grwSale.FocusedRowHandle >= 0)
            {
                txtId.Text = grwSale.GetFocusedRowCellValue( fieldName: "Id").ToString();
                lkuField.EditValue = grwSale.GetFocusedRowCellValue( fieldName: "SalePropertyId");
                lkuCustomerId.EditValue = grwSale.GetFocusedRowCellValue( fieldName: "CustomerId");
                txtSalePrice.Text = grwSale.GetFocusedRowCellValue( fieldName: "SalePrice").ToString();
                txtSaleDate.Text = grwSale.GetFocusedRowCellValue( fieldName: "SaleDate").ToString();
            }

            break;
```

As shown in this figure, I load data into the Grid according to the user's choice.

```
3 references
void LoadField()
{
    using (RealEstateAutomationContext context = new RealEstateAutomationContext())
    {
        var entity2 :IQueryable<{Id,PropertyId,...}> = from f :Field in context.Fields
                    join p :Property in context.Properties on f.PropertyId equals p.Id
                    join o :Owner in context.Owners on f.OwnerId equals o.Id
                    join ci :City in context.Cities on f.City equals ci.Id
                    join co :County in context.Counties on f.County equals co.Id
                    select new
                    {
                        Id = f.Id,
                        PropertyId = f.PropertyId,
                        OwnerId = f.OwnerId,
                        OwnerName = o.FirstName,
                        Area = f.Area,
                        Pafta = f.Pafta,
                        City = ci.CityName,
                        County = co.CountyName,
                        Address = f.Address,
                        Price = f.Price,
                        Description = f.Description,
                        Sold = f.Sold,
                        DeleteFlag = f.DeleteFlag,
                    };

        lkuField.Properties.DataSource = entity2.ToList().Where(x :{Id,PropertyId,...} => x.DeleteFlag == false && x.Sold == false);
        lkuField.Properties.DisplayMember = "OwnerName";
        lkuField.Properties.ValueMember = "PropertyId";
    }
}
```

In this way, I bring them to lookupedits to select the Property that will be selected for Sale.

```
4 references
private void LoadSaleField()
{
    using (RealEstateAutomationContext context = new RealEstateAutomationContext())
    {
        var entity = from s in context.Sales
                     join p in context.Properties on s.SalePropertyId equals p.Id
                     join f in context.Fields on p.ReferenceId equals f.Id
                     join o in context.Owners on s.OwnerId equals o.Id
                     join ci in context.Cities on f.City equals ci.Id
                     join co in context.Counties on f.County equals co.Id
                     join c in context.Customers on s.CustomerId equals c.Id
                     where p.PropertyType == "Field"
                     select new
                     {
                         Id = s.Id,


                         SalePropertyId = s.SalePropertyId,
                         SalePropertyType = s.SalePropertyType,
                         OwnerId = s.OwnerId,
                         CustomerId = s.CustomerId,
                         OwnerName = o.FirstName,
                         CustomerName = c.FirstName,
                         Area = f.Area,
                         Pafta = f.Pafta,
                         City = ci.CityName,
                         County = co.CountyName,
                         Address = f.Address,
                         SalePrice = s.SalePrice,
                         SaleDate = s.SaleDate,
                         Price = f.Price,
                         DeleteFlag = f.DeleteFlag,
                         Description = f.Description,
                         Sold = f.Sold,
                         SaleDeleteFlag = s.DeleteFlag,
                     };


        grcSale.DataSource = entity.ToList().Where(x => x.SaleDeleteFlag == false);

        grwSale.Columns["Ada"].Visible = false;
        grwSale.Columns["HouseType"].Visible = false;
        grwSale.Columns["Pafta"].Visible = true;

        lycField.Visibility = LayoutVisibility.Always;
        lycShop.Visibility = LayoutVisibility.Never;
        lycHouse.Visibility = LayoutVisibility.Never;
        lycPlot.Visibility = LayoutVisibility.Never;
    }
}
```

In this function, I prepared the grid control upload process for the field to be sold. I hide the columns that are not needed for the property I am processing. I unhide the necessary columns.

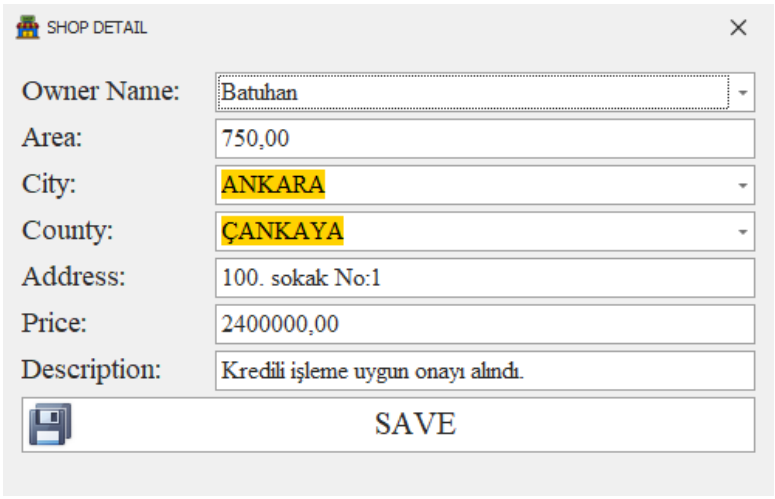
My sales function is as shown in the figure. For the operations, I proceed with my switch case structure according to the property type in the background. I continue the process by creating my variables. Then, I captured the ID of the property selected to be sold from the look up edit (all operations I explained here and above through the field are valid for other properties as well) and recorded it in the database accordingly. After adding a record for sale, I performed an update on that sold property and marked this property as sold. I prepared a similar structure for the update and delete functions.

```
2 references
void Sale()
{
    if (txtId.Text == "")
    {

        DialogResult confirmation = MessageBox.Show(text: @"Are you sure you want to save the information?",
            caption: @"Information", MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);

        if (confirmation == DialogResult.Yes)
        {
            int selectedFieldPropertyId = 0;
            int selectedHousePropertyId = 0;
            int selectedPlotPropertyId = 0;
            int selectedShopPropertyId = 0;
            int selectedId = 0;
            int selectedOwnerId = 0;
            decimal selectedArea = 0;
            string selectedAda = "";
            string selectedPafta = "";
            string selectedHouseType = "";
            int selectedCity = 0;
            int selectedCounty = 0;
            string selectedAddress = "";
            decimal selectedPrice = 0;
            string selectedDescription = "";
            bool selectedSold = false;
            bool selectedDeleteFlag = false;
```

```
switch (cmbPropertyType.Text)
{
    case "Field":

        if (lkuField.EditValue == null)
        {
            MessageBox.Show(text: @"Field cannot be empty, please select a field.",
                caption: @"Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
            break;
        }
        else
        {
            selectedFieldPropertyId = (int)lkuField.EditValue;
        }

        var selectedField = _fieldService.GetAll() // List<Field>
            .FirstOrDefault(f => f.PropertyId == selectedFieldPropertyId);

        if (selectedField != null)
        {
            selectedId = selectedField.Id; // Seçilen Field'ın Id'sini al
            selectedOwnerId = selectedField.OwnerId; // Seçilen Field'ın OwnerId'sini al
            selectedArea = selectedField.Area;
            selectedPafta = selectedField.Pafta;
            selectedCity = selectedField.City;
            selectedCounty = selectedField.County;
            selectedAddress = selectedField.Address;
            selectedPrice = selectedField.Price;
            selectedDescription = selectedField.Description;
            selectedSold = selectedField.Sold;
            selectedDeleteFlag = selectedField.DeleteFlag;
        }
```

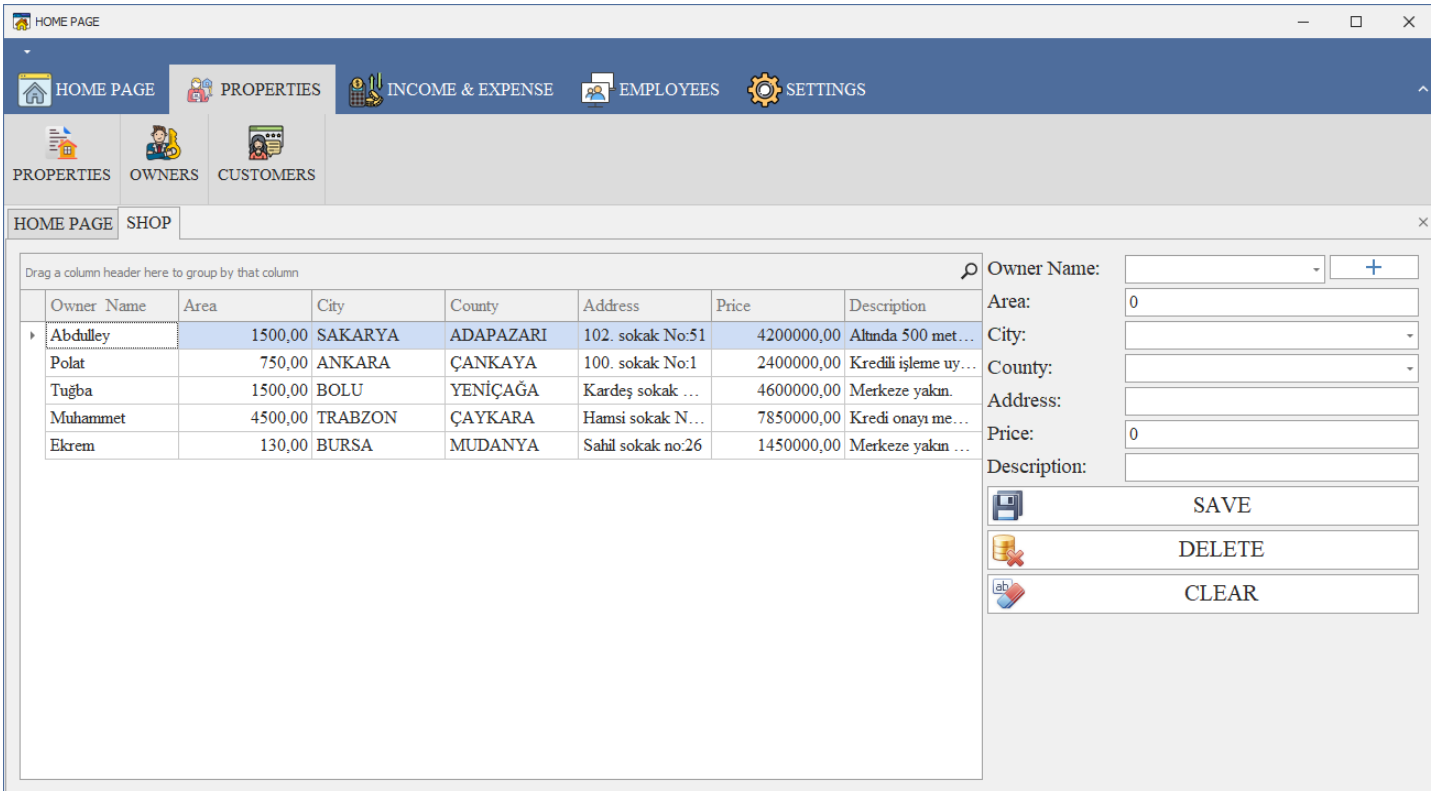
```
if (selectedOwnerId != 0 && selectedId != 0)
{
    _saleService.Add(new Sale
    {
        OwnerId = selectedOwnerId,
        SalePropertyId = selectedFieldPropertyId,
        SalePropertyType = "Field",
        CustomerId = Convert.ToInt32(lkuCustomerId.EditValue),
        SaleDate = DateTime.Now,
        SalePrice = Convert.ToDecimal(txtSalePrice.Text),
        DeleteFlag = false
    });

    _fieldService.Update(new Field
    {
        Id = selectedId,
        PropertyId = selectedFieldPropertyId,
        OwnerId = selectedOwnerId,
        Area = selectedArea,
        Pafta = selectedPafta,
        City = selectedCity,
        County = selectedCounty,
        Address = selectedAddress,
        Price = selectedPrice,
        Description = selectedDescription,
        Sold = true,
        DeleteFlag = selectedDeleteFlag
    });
}

LoadSaleField();

break;
```

19. Shop Detail Form



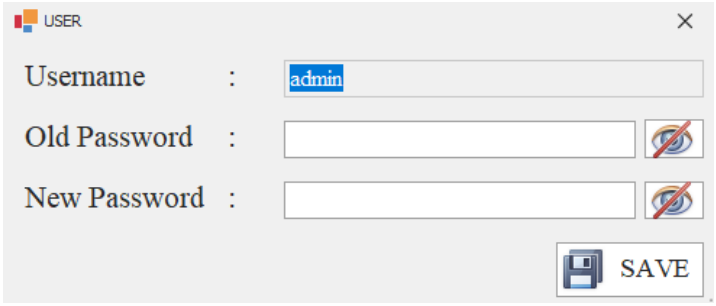

My Shop Detail form is as shown. The procedures are as in the Field Detail Form. The only difference between them is the Service structure. If you want to make a detailed review, you can review the Field Detail Form section above.

20. Shop Form




Shop Form is as shown in the figure above. The operations in the background are similar to the operations performed in the Field Form, the only difference between them is the service structures. If you want to make a detailed review, you can examine the Field Form field above.

21. User Form




The userform is as shown in the figure. The user can update his/her password after logging in to the dormitory.

```
1 reference
private bool Control(string userName, string password)
{
    var query :IEnumerable<User> = _userService.GetAll().Where(x :User => x.UserName == userName && x.UserPassword == password);

    if (query.Any())
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

My function is to check the username and old password before saving the user's new password.

```
void LoadUser()
{
    var enteredUser = _userService.GetAll() // List<User>
        .FirstOrDefault(x :User => x.Id == LoginForm.EnteredUserId);

    if (enteredUser != null)
    {
        UserId = enteredUser.Id;
        UserAuthorization = enteredUser.UserAuthorization;
        UserName = enteredUser.UserName;
        UserPassword = enteredUser.UserPassword;
    }

    txtUsername.Text = UserName;
}

1 reference
private void UserForm_Load(object sender, EventArgs e)
{
    LoadUser();
}
```

I assign user data to variables according to the User ID of the user who logs in from the login screen. and I print the username of the user into the text.

```
1 reference
private void btnOldPasswordHidden_Click(object sender, EventArgs e)
{
    txtOldPassword.Properties.UseSystemPasswordChar = false;
    btnOldPasswordHidden.Visible = false;
    btnOldPasswordVisible.Visible = true;
}

1 reference
private void btnOldPasswordVisible_Click(object sender, EventArgs e)
{
    txtOldPassword.Properties.UseSystemPasswordChar = true;
    btnOldPasswordHidden.Visible = true;
    btnOldPasswordVisible.Visible = false;
}

1 reference
private void btnNewPasswordHidden_Click(object sender, EventArgs e)
{
    txtNewPassword.Properties.UseSystemPasswordChar = false;
    btnNewPasswordHidden.Visible = false;
    btnNewPasswordVisible.Visible = true;
}

1 reference
private void btnNewPasswordVisible_Click(object sender, EventArgs e)
{

    txtNewPassword.Properties.UseSystemPasswordChar = true;
    btnNewPasswordHidden.Visible = true;
    btnNewPasswordVisible.Visible = false;
}
```

In this way, when the password on and off buttons are clicked in transactions, I turn the paswordchar feature on and off so that the user can view his old and new password.

```
0 references
internal static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    0 references
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new LoginForm());
    }
}
```

Finally, I select the start form of my project as Login and finish my project.

# 6. Conclusion

## 6.1 Benefits

### a. Benefits to users :

1. In this system, while the user used to use old systems to record business data, now the registration process will be very easy thanks to Real Estate automation.

2. In this way, the user will be able to save time in processes such as data recording, data viewing, data control and access information quickly and accurately.

3. Thanks to the new and usable screens prepared for the user, using the program will be much more comfortable and easier than other projects.

4. This project will make you different from other companies in the real estate world and you will have the opportunity to clearly and clearly examine the management of your company and the control of your business through income and expense analysis.

### b. Benefits to me :

1. I think this project will give me a great progress in my career. Thanks to this project, I will learn a lot of new things about software development, C# language, Entity Framework, DevExpress and Sql.

2. The fact that I will use this project in my office will give me the opportunity to see the usage part of the project and update and renew the project in case of any deficiencies and errors. In this way, it will be very easy to see the shortcomings and I will see the use of the project.

3. This project will help me to gain knowledge for other projects. In this way, it will support me to take projects in the business world in the future.

4. This project will help me gain trust with customers in the software world and keep me in the industry.

## 6.2 Ethics

- **Property Ethics:** The project complies with confidentiality standards and is restricted by authorization procedures. Care has been taken in this regard in terms of data security.

- **Ethics of Equality:** In the project, every user is treated equally and everything is prepared this way.

- **Customer Ethics:** In the project, customer confidentiality is taken into consideration and no data is used without permission.

- **Employer ethics:** The project worked for customer satisfaction.

- **Myself:** I worked for the project in a way that was always open to innovations and ready for the team, and tried to manage the process as controlled as possible.

**Why did I choose this project?**

One of the biggest reasons why I chose this project is that I am on both the client and the engineer side of the business and I will actually use the project in my own Real Estate office on a continuous basis. I have been in the real estate sector for more than 10 years. I have observed where there are deficiencies, mistakes and problems. Considering these, I chose to develop such a project. Thus, I can make the development and maintenance I want at any time.

## 6.3 Future Works

In the future, I plan to develop a system between the client and the office by preparing an integrated web application for my project, allowing them to view the goods in the hands of the real estate company, and providing direct communication with the specified numbers. Thus, a fast transportation will be provided to the customer. For this development, I plan to design my project on a layered architecture and integrate it easily in the future.

# References


ASLAN, R. M. (2019, Mar 17). *Katmanlı Mimari*. Medium: https://medium.com/kodcular/katmanl%C4%B1-mimari-9fb34ef8c376 adresinden alındı

Demırbas, E. (2022, Sep 14). *C# Katmanlı Mimari İle Northwind Veri Tabanı Tasarımı*. Medium: https://medium.com/@emre_79045/c-katmanl%C4%B1-mimari-i%CC%87le-northwind-veri-taban%C4%B1-tasar%C4%B1m%C4%B1-5f9f1c9d00e1 adresinden alındı

*Layout Control*. (2023, Apr 28). Dev Express: https://docs.devexpress.com/WindowsForms/3407/controls-and-libraries/form-layout-managers/layout-and-data-layout-controls/layout-control/layout-control adresinden alındı

Smith, A. (2021, Mar 8). *Connect to SQL Server database with appconfig, C#*. Microsoft: https://learn.microsoft.com/en-us/answers/questions/304395/connect-to-sql-server-database-with-appconfig-c adresinden alındı

*SQL Server Türkiye İl ve İlçe Veritabanı 2021*. (2021, May 15). Tasarım Kodlama: https://www.tasarimkodlama.com/veritabani/sql-server/sql-il-ilce-veritabani-scripti-guncel/ adresinden alındı

*What is Entity Framework?* (tarih yok). Entity Framework Tutorial: https://www.entityframeworktutorial.net/entityframework6/what-is-entityframework.aspx adresinden alındı

Yıldız, G. (2016, Apr 22). *Dependency Injection(DI) – Ninject*. www.gencayyildiz.com: https://www.gencayyildiz.com/blog/dependency-injectiondi-ninject/ adresinden alındı

Yüksel, E. (2023, Jan 27). *Fluent Validation Nedir? .NET Core'da Fluent Validation Kullanımı*. Medium: https://medium.com/@ecanyuksel/fluent-validation-b9b70c378056#:~:text=Fluent%20validation%3B%20bir%20nesnenin%20ge%C3%A7erli,s%C4%B1n%C4%B1fland%C4%B1r%C4%B1lm%C4%B1%C5%9F%20bir%20dizi%20kural%20sa%C4%9Flamaktad%C4%B1r. adresinden alındı