Fejlesztői Dokumentáció

A projekt részei

1. Model
    a. CustomerDTO.cs
    b. CUstomerService.cs
    c. PdfService.cs
    d. ProductDTO.cs
    e. PurchaseDTO.cs
    f. PurchaseService.cs
2. ViewModel
    a. Base
        i. BaseViewModel.cs
        ii. DefaultCommand.cs
    b. ProductViewModel.cs
3. View (TO_DO)

Általános tudnivalók

A programm MVVM architektúra alapján készült, így 3 részre van osztva:

A Model tartalmazza az üzleti logikát

A View (nézet) tartalmazza magát a program felületét, ez XAML kód segítségével van kialakítva és data Binding-ot használ.

A ViewModel (nézetmodel) az egy összekötő réteg a Model és a View között.

A program egy bútor áruház alkalmazottai számára készült, vásárlások kezeléséhez és dokumentálásához.

Az adatok SQL táblázatban vannak tárolva.

A View Command-okkal ad át adatokat a ViewModell-nek.

A program képes:

- Új vásárló regisztrálása
- Vásárlás hozzáadása és törlése
- Új termék hozzáadása
- Termék adatainak módosítása, termék törlése
- Adatok mentése pdf formátumban

•

Új vásárló regisztrálása

A *Customer's Name* és *Purchase ID* megadását követően a *Save Customer* feliratú gomb lenyomásával Commandon keresztül DataBinding-al adja át a program, az adatokat a **ProductViewModel** **addCustomer()** metódusának.

```
<TextBox
        Text="{Binding Path=currentCustomer.PurchaseID, Mode=TwoWay}"
        Grid.Column="1"
        Grid.Row="2"
        Style="{StaticResource textBoxStyle}"/>
<Button x:Name="bttnCustomerSaver"
        Command="{Binding Path=addCustomerCommand}"
        Content="Save Customer"
        Grid.Column="1"
        Grid.Row="3"
        Foreground="#5DBEBE"

    ></Button>
```

Ezt követően a **ProductViewModel** **addCustomer()** metódusa *try catch* en keresztül megpróbálja meghívni a **CustomerService** osztály **add**() metódusát, a vásárló adatait paraméterként átadva. Majd a **ProductViewModel** **loadCustomers()** metódusa frissíti a programban látható listát.

```
1 reference
public void addCustomer()
{
    try
    {
        customerService.add(currentCustomer);
        loadCustomers();
    }
    catch (Exception ex)
    {

        throw ex;
    }
}
```

A **CustomerService** **add()** metódusa a vásárló adatait paraméterként megkapja, létrehoz egy új *Customers* típusú *customer* változót és ennek

adja át a vásárló adatait, valamint adja meg a *Date*, és *FullPrice* alap értékeit. Ezt követve hozzáadja ezt az elemet az adatbázishoz és elmenti a változásokat.

```
1 reference
public void add(CustomerDTO newCustomer)
{
    var customer = new Customers();
    customer.Name = newCustomer.Name;
    customer.Date = DateTime.Now;
    customer.FullPrice = 0;
    customer.PurchaseID = newCustomer.PurchaseID;

    productDatabaseEntities.Customers.Add(customer);
    productDatabaseEntities.SaveChanges();

}
```

A **ProductViewModel** osztály **loadCustomers()** metódusa egy *Customers* nevű *ObservableeCollection<CustomerDTO>* típusú változónak adja át, a **CustomerService** osztály **getAllCustomer()** metódusának visszatérési értékét.

```
6 references
public void loadCustomers()
{

    Customers = new ObservableCollection<CustomerDTO>(customerService.getAllCustomer());
}
```

A **CustomerService** **getAllCustomer()** metódusa egy *CustomerDTO* típusú listába próbála *try catch*-en keresztül kigyűjteni, az összes vásárlót az adatbázisból egy *foreach()* ciklus használatával és ezt **return** értékként visszaadni.

```
1 reference
public List<CustomerDTO> getAllCustomer()
{
    List<CustomerDTO> allCustomer = new List<CustomerDTO>();

    try
    {
        var customers = from c in productDatabaseEntities.Customers select c;

        foreach (var item in customers)
        {
            allCustomer.Add(new CustomerDTO
            {
                Name = item.Name,
                Date = item.Date,
                PurchaseID = item.PurchaseID,
                FullPrice = item.FullPrice,
            });
        }
    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;
    }
    return allCustomer;
}
```

Vásárlás hozzáadása és törlése

Hozzáadás:

A *Product ID, Customer ID, Price* és *Quantity* megadását követően a *Save Purchase* feliratú gomb lenyomásával Commandon keresztül *DataBinding*-al adja át a program, az adatokat a **ProductViewModel addPurchase()** metódusának.

```
<TextBlock Text="Product ID"
           Style="{StaticResource fontStyle }"
           Grid.Column="0"
           Grid.Row="5"
           HorizontalAlignment="Right"
           />
<TextBlock/>
<TextBox Style="{StaticResource textBoxStyle}"
         HorizontalAlignment="Left"
         Text="{Binding Path=currentPurchase.ProductID, Mode=TwoWay}"
         Grid.Column="1"
         Grid.Row="5"/>
<TextBlock Text="Customer ID"
           Style="{StaticResource fontStyle }"
           Grid.Column="0"
           Grid.Row="6"
           HorizontalAlignment="Right"
           />
<TextBox Style="{StaticResource textBoxStyle}"
         HorizontalAlignment="Left"
         Text="{Binding Path=currentPurchase.CustomerID, Mode=TwoWay}"
         Grid.Column="1"
         Grid.Row="6"
         x:Name="CustID"/>
<TextBlock Text="Price"
           Style="{StaticResource fontStyle }"
           Grid.Column="0"
           Grid.Row="7"
           HorizontalAlignment="Right"
           />
<TextBox Style="{StaticResource textBoxStyle}"
         HorizontalAlignment="Left"
         Text="{Binding Path=currentPurchase.Price, Mode=TwoWay}"
         Grid.Column="1"
         Grid.Row="7"/>
<TextBlock Text="Quantity"
           Style="{StaticResource fontStyle }"
           Grid.Column="0"
           Grid.Row="8"
           HorizontalAlignment="Right"
           />
<TextBox Style="{StaticResource textBoxStyle}"
         HorizontalAlignment="Left"
         Text="{Binding Path=currentPurchase.Quantity, Mode=TwoWay}"
         Grid.Column="1"
         Grid.Row="8"/>
```

```
<Button x:Name="bttnSave"
        Content="Save Purchases"
        Grid.Column="1"
        Grid.Row="10"
        Foreground="#5DBEBE"
        Command="{Binding Path=addPurchaseCommand}"/>
<Button Command="{Binding Path=searchCustomerCommand}"
        Content="Save to Pdf"
        Grid.Column="0"
        Grid.Row="10"
        Foreground="#5DBEBE"/>
```

Ezt követően a **ProductViewModel addPurchase()** metódusa *try catch* en keresztül megpróbálja meghívni a **PurchaseService** osztály **add**() metódusát, a vásárlás adatait paraméterként átadva.

Ha a *purchaseService.success* változó értéke **TRUE**, akkor meghívja a **ProductService editIfPurchaseAdded()** és a **CustomerService editIfPurchaseAdded()** metódusokat. Majd a **ProductViewModel loadPurchase() és loadData()** metódusaival frissíti a programban látható listát. majd **Clear()** metódussal kitörli a *Products ObservableCollection* értékét. és meghívja a **ProductViewModel loadData()** funkcióját.

```
public void addPurchase()
{
    try
    {
        purchaseService.add(currentPurchase);

        if (purchaseService.success)
        {
            productService.editIfPurchaseAdded(currentPurchase.ProductID, currentPurchase.Quantity);
            customerService.editIfPurchaseAdded(currentPurchase);
        }

        loadPurchase();
        loadCustomers();
        Products.Clear();
        loadData();
        infoMessage = purchaseService.settingMessage();

    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

PurchaseService add() funkciója *try catch*-el megpróbálja **linq** kéréssel megkeresi a terméket a *Product ID* alapján majd létrehozza a *Purchases* típusú *purch* változót, amelynek átadja a *newPurchase* változó adatait.

```
1 reference
public void add(PurchaseDTO newPurchase)
{
    try
    {
        var purchashedProduct = new Product();
        var pr = from p in productDatabaseEntities.Product where p.Id == newPurchase.ProductID  select p.Price;

        purchashedProduct.Price = pr.Sum();

        var purch = new Purchases();
        purch.CustomerID = newPurchase.CustomerID;
        purch.ProductID = newPurchase.ProductID;
        purch.Price = purchashedProduct.Price;
        purch.Quantity = newPurchase.Quantity;

        int price = purch.Price;
        int quantity = purch.Quantity;
```

Ezt követően ismét **linq** kéréssel a *Product ID* alapján megkeresi a terméket és annak az *ID*-jét átadja az int típusú *pid* változónak. A *pid* változó segítségével megtalálja a termék adatait az adatbázisban és átadja a *realProduct* változónak.

Ezt követően **if** segítségével megvizsgálja, hogy a megadott mennyiség nagyobb-e mint az adatbázisban szereplő mennyiség az adott termékhez.

Ha nagyobb akkor a *success* változó **FALSE** értéket kap. Viszont ha kisebb, akkor a *success* változó **TRUE** értéket kap és a vásárlás hozzáadásra kerül az adatbázishoz, majd **SaveChanges()** metódussal elmenti a változásokat.

```
var product = from p in productDatabaseEntities.Product where p.Id == purch.ProductID select p.Id;
int pid = product.Sum();

var realProduct = productDatabaseEntities.Product.Find(pid);
if (quantity > realProduct.Quantity)
{
    Message = "You dont even have that many item you bozo";
    success = false;
}
else
{

    success = true;
    productDatabaseEntities.Purchases.Add(purch);
    productDatabaseEntities.SaveChanges();

    Message = "Success";
}
```

ProductService editIfPurchaseAdded() metódus paraméterként megkapja, egy termék id-jét *(id)* és mennyiségét *(quantity).*

Az *id* alapján megtalálja az adott terméket az adatbázisban és ennek tulajdonságait egy új, *product* változónak adja át majd a *product.Quantity* tulajdonságból kivonja a termék mennyiségét *(quantity)* és elmenti a változásokat az adatbázisban.

CustomerService editIfPurchaseAdded() metódus paraméterként megkapja a *currentPurchase* változót amely egy *PurchaseDTO* típusú változó és az éppen hozzáadandó vásárlás adatait tartalmazza.

A metódus megkeresi a *purchase.CustomerID* és a *purchase.ProductID* alapján a megfelelő adattáblákban a megfelelő vásárló és termék adatait, és ezeket átadja a *customer* és *product* változóknak. Ezután a *customer.FullPrice* változóhoz, hozzá adja az új vásárlásban vett áru árát.

```
1 reference
public void editIfPurchaseAdded(PurchaseDTO purchase)
{
    var customer = productDatabaseEntities.Customers.Find(purchase.CustomerID);
    var product = productDatabaseEntities.Product.Find(purchase.ProductID);

    customer.FullPrice += product.Price * purchase.Quantity;
    productDatabaseEntities.SaveChanges();
}
```

Törlés:

A **PurchasesListView** oldalon az *Id* megadása után a *Delete purchase* feliratú gomb *DataBinding* és *Command* segítségével meghívja a **ProductViewModel deletePurchase()** metódusát.

```
<TextBlock Text="Id"
           Grid.Column="0"
           Grid.Row="2"
           Style="{StaticResource ResourceKey=fontStyle}"/>
<TextBox Text="{Binding Path=currentPurchase.Id, Mode=TwoWay}"
         Style="{StaticResource textBoxStyle}"
         Grid.Column="1"
         Grid.Row="2"
         />
<Button Command="{Binding Path=deletePurchaseCommand}"
        Height="20" Width="100"
        HorizontalAlignment="Right"
        Content="Delete purchase"
        Grid.Column="1"
        Grid.Row="3"
        Foreground="#5DBEBE"/>
```

A **deletePurchase()** metódus először meghívja a **ProductViewModel searchPurchase()** metódusát majd pedig meghívja a **PurchaseService delete()** metódusát, annak paraméterként megadva a *currentPurchase Id* tulajdonságát.

Ezután meghívja a **ProductService editIfPurchaseDeleted()** valamint a **CustomerService editIfPurchaseDeleted()** metódusokat. Ezt követően frissíti a megjelenített adatokat a **loadPurchase()**, **loadData()**, **loadCustomers()** metódusokkal.

```
1 reference
public void deletePurchase()
{
    searchPurchase();
    purchaseService.delete(currentPurchase.Id);
    productService.editIfPurchaseDeleted(currentPurchase.ProductID, currentPurchase.Quantity);
    customerService.editIfPurchaseDeleted(currentPurchase);
    loadPurchase();
    loadData();
    loadCustomers();
    infoMessage = purchaseService.settingMessage();
}
```

A **searchPurchase()** metódus meghívja a **PurchaseService** osztály **searchPurchase()** metódusát és annak paraméterül a *currentPurchase Id* tulajdonságát adja. Ennek a **return** értékét átadja egy *PurchaseDTO* típusú *foundPurchase* változónak, amely értékeit ezután átadja a *currentPurchase* változónak

```
1 reference
public void searchPurchase()
{
    PurchaseDTO foundPurchase = purchaseService.searchPurchase(currentPurchase.Id);
    currentPurchase.ProductID = foundPurchase.ProductID;
    currentPurchase.CustomerID = foundPurchase.CustomerID;
    currentPurchase.Price = foundPurchase.Price;
    currentPurchase.Quantity = foundPurchase.Quantity;

}
```

A **PurchaseService** **searchPurchase()** metódusa paraméterként megkapja a **currentPurchase.Id** változó értékét. Ez alapján *try catch*-el megpróbája kikeresi a megfelelő adattáblából azt a vásárlást amely **id** száma egyenlő a megadott *id* változó értékével és ezt átadja a *purchase* változónak.

Miután ezmegtörtént egy if-el megvizsgálja, hogy **NULL** értékű-e a *purchase* változó. Ha nem **NULL** értékű, akkor létrehoz egy *purhaseDTO* nevű *PurchaseDTO* típusú változót és ennek átadja a *purchase* változó értékeit majd **return** értékként visszaadja a *purchaseDTO* változót.

```
1 reference
public PurchaseDTO searchPurchase(int id)
{
    PurchaseDTO purchaseDTO = null;

    try
    {
        var purchase = productDatabaseEntities.Purchases.Find(id);
        if (purchase !=null)
        {
            purchaseDTO = new PurchaseDTO()
            {
                Id = purchase.Id,
                CustomerID = purchase.CustomerID,
                ProductID = purchase.ProductID,
                Price = purchase.Price,
                Quantity = purchase.Quantity
            };
        }
    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;
    }

    return purchaseDTO;

}
```

A **PurchaseService** **delete()** metódusa a *currentPurchase.Id*-ét kapja paraméterül és ezt a megfelelő adattáblában *id* alapján megkeresi, majd eltávolítja az adattáblából. Ezt követően elmenti a változásokat.

```
1 reference
public void delete(int primaryid)
{

    var purchase = productDatabaseEntities.Purchases.Find(primaryid);
    productDatabaseEntities.Purchases.Remove(purchase);

    productDatabaseEntities.SaveChanges();

    Message = "Purchase successfully removed!";
    productDatabaseEntities.SaveChanges();

}
```

A **ProductService** **editIfPurchaseDeleted()** metódusa paraméterként megkapja a *currentPurchase.Id*-ét és a *currentPurchase.Quantity*-t. Az *id* alapján megtalálja a megfelelő adattáblában a terméket és ezt átadja a *product* változónak. Ezután a *quantity*-t hozzáadjuk a *product.Quantity*-hoz, a megadott termék raktárban lévő számának növekedése miatt. Végezetül elmenti a változásokat.

```
public void editIfPurchaseDeleted(int id, int quantity)
{
    var product = ProductDatabaseEntities.Product.Find(id);
    product.Quantity += quantity;
    ProductDatabaseEntities.SaveChanges();
}
```

A **CustomerService** **EditIfPurchaseDeleted()** metódusa paraméterül kapja a *currentPurchase*-t, a megfelelő adattáblából kikeresi *CustomerId* alapján az adott vásárlást majd a vásárlóhoz tartozó *customer.FullPrice*-ból kivonja a vásárlásnál fizetett árat *(purchase.Price*purchase.Quantity).*

```
public void editIfPurchaseDeleted(PurchaseDTO purchase)
{
    var customer = productDatabaseEntities.Customers.Find(purchase.CustomerID);

    customer.FullPrice -= purchase.Price * purchase.Quantity;
}
```

A **loadData(), loadPurchase()** és **loadCustomers()** metódusok rendre egy *ObservableCollection*-nek adják át a megfelelő Service osztályok **getAll** metódusainak **return** értékét.

```
6 references
public void loadData()
{
    Products = new ObservableCollection<ProductDTO>(productService.getAllProduct());
}

3 references
public void loadPurchase()
{
    Purchases = new ObservableCollection<PurchaseDTO>(purchaseService.getAllPurchase());
}

6 references
public void loadCustomers()
{

    Customers = new ObservableCollection<CustomerDTO>(customerService.getAllCustomer());
}
```

A **ProductService** **getAllProduct()** metódusa létrehoz egy *ProductDTO* típusú adatokat tároló listát *productDTOs* néven. Ezt ezután *try catch*-el próbálja **foreach** ciklus használatával feltölteni egy **linq** kérés segítségével ami visszaadja az összes elemet az adattáblából. Végezetül **return** érték ként visszaadja a *productDTOs* listát.

```
public List<ProductDTO> getAllProduct()
{
    List<ProductDTO> productDTOs = new List<ProductDTO>();

    try
    {
        var products = from pr in ProductDatabaseEntities.Product select pr;

        foreach (var pr in products)
        {
            productDTOs.Add
                (new ProductDTO
                {
                    Id = pr.Id,
                    Name = pr.Name,
                    Manufacturer = pr.Manufacturer,
                    Quantity = pr.Quantity,
                    Price = pr.Price,
                    Width = pr.Width,
                    Length = pr.Length,
                    Height = pr.Height
                });
        }
    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;
    }

    return productDTOs;
}
```

A **PurchaseService getAllPurchase()** metódus az előzőhez hasonló módon **linq** kérés segítségével egy *try catch*-en keresztül **foreach** ciklussal listába helyezi az összes elemet a megfelelő adattáblából made ezt **return** értékként visszaadja

```
public List<PurchaseDTO> getAllPurchase()
{
    List<PurchaseDTO> purchaseDTOs = new List<PurchaseDTO>();

    try
    {
        var purchases = from p in productDatabaseEntities.Purchases
                        select p;


        foreach (var p in purchases)
        {
            purchaseDTOs.Add(new PurchaseDTO
            {
                Id = p.Id,
                ProductID = p.ProductID,
                CustomerID = p.CustomerID,
                Quantity = p.Quantity,
                Price = p.Price
            });

        }
    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;

    }
    return purchaseDTOs;
}
```

A **CustomerService getAllCustomers()** metódusa ugyan így működik *CustomerDTO* típusú elemeket tároló listát adva vissza.

```
public List<CustomerDTO> getAllCustomer()
{
    List<CustomerDTO> allCustomer = new List<CustomerDTO>();

    try
    {
        var customers = from c in productDatabaseEntities.Customers select c;

        foreach (var item in customers)
        {
            allCustomer.Add(new CustomerDTO
            {
                Name = item.Name,
                Date = item.Date,
                PurchaseID = item.PurchaseID,
                FullPrice = item.FullPrice,
            });
        }
    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;
    }
    return allCustomer;
}
```

A **ProductViewModel LoadData()** metódusa is ugyan ezen az alapon működik, meghívja a **ProductService getAllProduct()** metódusát.

```
public List<ProductDTO> getAllProduct()
{
    List<ProductDTO> productDTOs = new List<ProductDTO>();

    try
    {
        var products = from pr in ProductDatabaseEntities.Product select pr;

        foreach (var pr in products)
        {
            productDTOs.Add
                (new ProductDTO
                {
                    Id = pr.Id,
                    Name = pr.Name,
                    Manufacturer = pr.Manufacturer,
                    Quantity = pr.Quantity,
                    Price = pr.Price,
                    Width = pr.Width,
                    Length = pr.Length,
                    Height = pr.Height
                }) ;
        }
    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;
    }

    return productDTOs;
}
```

Termék hozzáadása

A **StorageView**-ben az adatok megadása után az *Add Product* feliratú gomb megnyomásával *DataBinding* és *Command* segítségével hívja meg a program, a **ProductViewModel** **addProduct()** metódusát. Ez a metódus *try catch*-el megpróbálja meghívni a **ProductService** **add()** metódusát és annak a *currentProduct* változót átadni paraméterként. ezután a **return** értékét átadja az *isAdded* változónak és a **loadData()** metódussal frissíti a megjelenő adatokat. **If**-segítségével megvizsgálja, hogy az *isAdded* változó értéke **TRUE**-e és ha igen akkor sikeres üzenetet ad az *infoMessage*-be, ha nem akkor *errorMessage* változó tartalmát kapja meg az *infoMessage*.

```
public void addProduct()
{
    try
    {
        bool isAdded = productService.add(currentProduct);
        loadData();
        if (isAdded)
        {
            infoMessage = "Product successfully added to the database!";
        }
        else
        {
            infoMessage = errorMessage;
        }

    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

A **ProductService** **add()** metódusa az új termék adatait tartalmazó változót kapja meg paraméterként *(newProduct)*. Ezt követően egy try catch-el megpróbál létrehozni egy új *Product* típusú változót *(product)* és annak átadni a *newProduct* értékeit. A *product*-ot hozzáadja az adattáblához és elmenti a változásokat. Végezetül az *isAdded* változónak az értékét megadja az *affectedRows* változó segítségével és ezt adja vissza **return** értékként

```
public bool add(ProductDTO newProduct)
{
    bool isAdded = false;
    try
    {
        var product = new Product();
        product.Id = newProduct.Id;
        product.Name = newProduct.Name;
        product.Manufacturer = newProduct.Manufacturer;
        product.Quantity = newProduct.Quantity;
        product.Price = newProduct.Price;
        product.Width = newProduct.Width;
        product.Length = newProduct.Length;
        product.Height = newProduct.Height;

        ProductDatabaseEntities.Product.Add(product);
        var affectedRows = ProductDatabaseEntities.SaveChanges();

        isAdded = affectedRows > 0;


    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;
    }

    return isAdded;
}
```

Termék adatainak módosítása, termék törlése

Módosítás:

A termék adatainak módosításához a **StorageView** ablakban megadjuk az adatokat és az *edit Product* feliratú gomb megnyomásával meghívja a program a **ProductViewModel** **edit()** metódusát.

Az edit() metódus try catch-el megpróbálja meghívni a **ProductService** **edit()** metódusát és annak paraméterként átadni, a *currentProduct* változót, majd a loadData() metódust meghívva frissíti, a megjelenített adatokat.

If használatával megvizsgálja az *isEdited* változó értékét és ha **TRUE** értéket tartalmaz, akkor sikeres üzenetet kap meg az *infoMessage* változó értéknek. Ha **FALSE** értékkel rendelkezik, akkor az *errorMessage* változó értékét kapja meg értéknek az *infoMessage* változó.

```
private void edit()
{
    try
    {
        var isEdited = productService.edit(currentProduct);
        loadData();
        if (isEdited)
        {
            infoMessage = "Product successfully edited!";
        }
        else
        {
            infoMessage = errorMessage;
        }
    }
    catch (Exception ex)
    {
        infoMessage = ex.Message;
    }
}
```

A **ProductService edit()** metódusa a paraméterként kapott *ProductDTO* típusú változó alapján *try catch*-el megpróbálja, kikeresi az adattáblában azt az elemet, amelynek az *Id* értéke megegyezik a keresett *Id*-vel. Ezt követően átadja neki az új értékeket és elmenti a változásokat. Az *isEdited* változó értéket az *affectedRows* változó segítségével adja meg, majd azt adja vissza **return** értékként.

```
public bool edit(ProductDTO productToEdit)
{
    bool isEdited = false;

    try
    {
        var product = ProductDatabaseEntities.Product.Find(productToEdit.Id);
        product.Name = productToEdit.Name;
        product.Manufacturer = productToEdit.Manufacturer;
        product.Quantity = productToEdit.Quantity;
        product.Price = productToEdit.Price;
        product.Width = productToEdit.Width;
        product.Length = productToEdit.Length;
        product.Height = productToEdit.Height;

        var affectedRows = ProductDatabaseEntities.SaveChanges();

        isEdited = affectedRows > 0;

    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;
    }

    return isEdited;
}
```

Törlés:

A **StorageView**-ben megadva a kitörlendő termék *ID*-jét és a *Delete Product* gomb megnyomásával *DataBinding* és *Command* használatával a program meghívja a **ProductViewModel** **delete()** metódusát. A metódus *try catch*-el megpróbálja meghívni a **ProductService** **delete()** metódusát és annak **return** értékét átadja az *isDeleted* változónak. majd **loadData()** metódussal frissíti a megjelenített adatokat. Ezután *If*-et használva megvizsgálja az *isDeleted* változó értékét és attól függően az *infoMessage* vagy egy sikeres üzenetet tartalmazó *string*-et kap értékül, vagy az *errorMessage* változó értékét.

```
private void delete()
{
    try
    {
        var isDeleted = productService.delete(currentProduct.Id);
        loadData();
        if (isDeleted)
        {
            infoMessage = "Product Successfully deleted from the database!";
        }
        else
        {
            infoMessage = errorMessage;
        }
    }
    catch (Exception ex)
    {

        throw ex;
    }
}
```

A **ProductService delete()** metódusa megkapja a törlendő elem *id* értékét és egy *try catch*-el megpróbálja megkeresni ezzel az *id*-vel rendelkező elemet majd kitörli az adattáblából. Ezután elmenti a változásokat és az *affectedRows* változó segítségével megadja az *isDeleted* változó értékét.

```
public bool delete(int idToDelete)
{
    bool isDeleted = false;

    try
    {
        var product = ProductDatabaseEntities.Product.Find(idToDelete);
        ProductDatabaseEntities.Product.Remove(product);

        var affectedRows = ProductDatabaseEntities.SaveChanges();

        isDeleted = affectedRows > 0;
    }
    catch (Exception ex)
    {

        infoMessage = ex.Message;
    }

    return isDeleted;
}
```

Adatok mentése pdf formátumban

A vásárló adatainak pdf fájlba való elmentéséhez a **NewCostumerView** ablakban a *Customer Id* megadását követően a *Save to pdf* feliratú gomb lenyomásával *DataBinding* és *Command* használatával meghívja a program, a **ProductViewModel** **searchCustomer()** metódusát.

A **searchCustomer()** metódus *CustomerDTO* típusú *foundCustomer* változónak adja át a **CustomerService** **search()** metódusának a **return** értékét. az utóbb említett metódus a *currentCustomer.PurchaseID* értékét kapja meg paraméterként. majd **If**-el megvizsgálja, hogy a *foundCustomer* értéke nem **NULL**-e. Ha nem **NULL** az értéke, akkor a *currentCustomer* megkapja a *foundCustomer* adatait és meghívódik a **PdfService** **writeToPdf()** metódusa, amely a *currentCustomer* változót kapja paraméterül.

```
1 reference
public void searchCustomer()
{
    CustomerDTO foundCustomer = customerService.search(currentCustomer.PurchaseID);
    if (foundCustomer != null)
    {
        currentCustomer.Name = foundCustomer.Name;
        currentCustomer.Date = foundCustomer.Date;
        currentCustomer.FullPrice = foundCustomer.FullPrice;
        pdfService.writeToPdf(currentCustomer);
    }


}
```

A PdfService writeToPdf() metódusa paraméterként megkapja az elmentendő customer változó értékeit. Létrehoz egy Customerbill mappát a projekt debug mappájában és ide hoz létre egy pdf fájlt PdfWriter segítségével. Ebbe a pdf fájlba írja ki a vásárló adatait, majd az összes vásárlását.

```
public void writeToPdf(CustomerDTO customer)
{
    string pdfName = customer.Name + ".pdf";
    System.IO.Directory.CreateDirectory(Environment.CurrentDirectory + "\\CustomerBill");
    PdfWriter pdfWriter = new PdfWriter(Environment.CurrentDirectory + "\\CustomerBill" + "\\" + pdfName);
    PdfDocument pdf = new PdfDocument(pdfWriter);
    Document document = new Document(pdf);
    Paragraph header = new Paragraph("*Xy furniture store*").SetTextAlignment(TextAlignment.CENTER).SetFontSize(22);
    document.Add(header);
    LineSeparator ls = new LineSeparator(new SolidLine());
    document.Add(ls);
    Paragraph basicData = new Paragraph("Id number: " + customer.PurchaseID.ToString()
        + "\nPurchase date: " + customer.Date + "\nCustomer Name: " + customer.Name + "\n\n");

    document.Add(basicData);


    var purchases = from p in entities.Purchases where p.CustomerID == customer.PurchaseID  select p;
    int LineDrawn = 0;
foreach (Purchases item in purchases)
{
    int dashedLineNeeded = purchases.Count() - 1;



    var prod = entities.Product.Find(item.ProductID);
    string prodName = prod.Name;

    Paragraph product = new Paragraph("Bought product: " + prodName);
    Paragraph unitprice = new Paragraph("Unit price: " +item.Price.ToString());
    Paragraph quantity = new Paragraph("Number of units bought: " + item.Quantity);
    Paragraph price = new Paragraph("Subtotal: " + item.Quantity * item.Price);
    LineSeparator line = new LineSeparator(new DashedLine());
    document.Add(product);
    document.Add(unitprice);
    document.Add(quantity);
    document.Add(price);

    if (LineDrawn < dashedLineNeeded)
    {
        LineDrawn++;
        document.Add(line);
    }




}


   document.Add(ls);
   Paragraph fullPrice = new Paragraph("Full price: " + customer.FullPrice).SetTextAlignment(TextAlignment.RIGHT).SetFontSize(14).SetBold();
   document.Add(fullPrice);


   document.Close();
}
```