Papa Dario's Pizza Report

FINAL PROJECT

PROG-32356

2020-08-15

Prof.Dario Guiao

Luka Necajev

ID=991475972

TABLE OF CONTENTS

iii. Creating Own Pizza Or Sandwich

- Adding toppings to the toppings List
- Display Cart
- Cp Product
- Calculate Total
- Remove Product Cart
- Add Custom Pizza Method
    i. Update Cart View

WEBPAGE DESIGN MAP

Navigation

The navigation is kind of a mix up that I really enjoy, I used the bigger heading from Pizza nova, and changed the design to suit my needs, keeping everything in the center and ensuring that important things, for instance, the cart and the sign in page are where the user would normally see since they are one of the more important elements for the user interface. The Main navigation aspect of the Navigation bar includes a link to a menu page, this menu page will display all the different menu's (Pizza, Sandwich, Other, Create Own) items that you can order. The Specials page is a direct link to the Specials/Combos page. Next is the History tab where the user would be able to see his Order history followed by the Suggestions page, this page is made for you to have easy access to the suggestions that you could give, initially the idea was to keep it only in the main page on the phone icon, but since I did not have enough time to make a locations page, I just replaced it with Suggestions. The last navigation is the About us page, this page is all about Papa Dario. On the right-hand side there is contact information (phone number) and your usual sign up/sign in as well as the cart.

Main Page/Home Page

The home page is the main hub which will allow you to choose your fate. By this I mean that you will be able to choose almost anywhere you want to go from this page. This page is designed to navigate to all the popular item's pages, as well as a direct link to the combos and Creating a pizza view. It also includes a promotion for you to get 10% off your orders, so when you click on it you will be taken to create an account and login. You also have a suggestions icon in the bottom right which will take you directly to the suggestions page, which you can also get to in the suggestions page. The main page is designed for easy access to Papa Dario's products as everything is centered and simple.

Menu Page

The menu page will display all the different menu's, this includes Popular Pizza's, Popular Sandwich's, Other favourites (wings and sides), and being able to Create Your Own items that you can order. The menu page is an outsource to all other product pages.

Create Your Own Navigation Page

The Create your own navigation page allows you to choose if you want to create your pizza , sandwich, or if you change your mind, allows you to directly go to the Popular Pizza, Popular Sandwiches and Other Favourites pages. It is based on easy to use User interface and basic web design with a bit of a twist.

The Specials Page

Also called the Combo's Page, lists all the special products that the owner is selling to users, the "deals" of Papa Darios. Having two for three and different specialized pizzas that cost more than the regular Pizza's.

Create Own Pizza and Create Own Sandwich Pages

These pages allow you to easily create your own sandwiches and pizzas with specific toppings that allow you to specialize your product for you, they will be stored in the database and in your receipt. You can see your toppings as well as your items in your cart on every page.

Member History Page

This page would allow you to view the members history. I had a lot of trouble with the observable lists, but this page displays the members history based on who is logged in, since you can login on every page by clicking the cart you can either do it that way or sign in and come back to the page. The page also includes a login button on the top left corner to allow you to easily login.

Suggestions

This page is made for you to have easy access to the suggestions that you could give. Initially, I meant to keep it only in the main page on the phone icon, but since I do not have a locations page, I just replaced it with Suggestions. The last navigation is the About us page, this page is all about Papa Dario. This is a simple page allowing interaction between the admins and the users.

About Us Page

The about us page gives background information about Papa Dario, his past and struggles that he had. It shows you a nice picture of him and a video clip that shows how he makes his pizza. The About us page also includes the contact information of the main restaurant.

Sign In Page

The sign in page allows you to Sign in as a user. I did not want to clutter the screen, so I divided sign in and sign up into two different pages. Normally you would sign in as a user using you email and password. I believe that having a username and an email is not appropriate for a pizza page since people have the tendency to forget their usernames and would have to either login with email or get their passwords using forget Password.

Sign Up Page

The sign-up page allows users to sign up and join the Papa Dario Club, this club gets exclusive offers like 10% off every online order. The sign-up page also includes regex that will catch any error the user makes while creating their application. Once they sign up the users are sent to the database.

Admin Page Theory

To login to the admin page you don't use an email, you use a predefined login information like (admin/admin OR Dario2020/abcde). Once you login with these credentials, you will be taken to a new page/application almost like it is separate from the others allowing crud to be preformed.

Admin page (First view is Suggestions)

This will allow the admin access to the Suggestions, allowing him to add, remove, update and delete suggestions that you will be able to see using the Observable list. This page includes buttons which will allow a flyout/ popup to allow easy access on the left side, as well as a different way to edit your items, using combo boxes to edit your information by choosing each item individually.

Admin Page Menu (allows product crud)

When you click menu in the admin page, you will be redirected to a Menu Page where you will be able to create new products, edit your old products and delete them as well. This page allows you to edit the items in the database. Unfortunately, I did not find a good way to responsively populate the added products into the page.

Cart

The cart is a flyout that will allow the user to view the cart seamlessly throughout multiple pages and will allow you to checkout at any time/page. You can also delete products from the cart as well as add new ones from different pages.

Outstanding Features

1. Allows the user to view the cart seamlessly throughout multiple pages and allows you to checkout at any time/page. You can delete products from the cart as well as add new ones from tdifferent pages. This took longer than I anticipated given that I had 2 applications, one with no passing of values and the other that I had to redesign to more efficiently incorporate the navigational features.
2. Having a create pizza and sandwich page, allowing users to create a pizza easily, adding to the price per topping.

3. Being logged in through different pages and having the name show up on each page that shows your Name in the top right corner of the pages.
4. Created Entity First Classes.

Brief Introduction to how to use program

The application is fairly straight forward, the only parts you might get a little stuck on is the cart, you do not have to update the items before clicking checkout, they both do the same thing one just displays the information for you before you sends it while the other just sends the receipt and DataHistory to the db.

Another thing that might confuse you is that when you are logged in as admin and you click on anything but Suggestions and Menu, it will send you back to the original Program. This is a minor issue that I missed on fixing.

Other then that things are pretty straight forwards, I have commented my code so you should be able to understand what each method is doing and what is going on in the method by reading the comments if that is needed.

Also The Database File is in the same folder as this word file, it is in a txt format.

DataBase Access

This Class allows access from the application to the database. It allows CRUD on a non entity framework and uses different techniques to get information from different tables in the database.

Creating Connections to the database

These variables will be used to connect, build, store, and edit the datasets.

```
//connection to the MEMBERS DATABVASE
        private SqlConnection _conn;
        private SqlDataAdapter _adapter;
        private SqlCommandBuilder _cmdBuilder;
        private DataSet _dataSet;
        private DataTable _tblMembers;

        //connection to the MEMBER HISTORY DATABASE
        private SqlDataAdapter _MemberHistoryAdapter;
        private SqlCommandBuilder _MemberHistorycmdBuilder;
        private DataSet _MemberHistorydataSet;
        private DataTable _tblMemberHistory;
```

Setting the Information/Filling Data

This information will create the connection using the connection string, store the query as well as the connection into the adapter, and use the cmd builder to allow the building of the queries using the SQL package provided by Microsoft. The next thing to do would be to fill your data using the method FillDataSet. This dataset is based on Each table, and will be a little different for each table.

```
  string cs = GetConnectionString("DarioPizzaDBLocalDB");

            string query = "Select MemberID, MemberFirstName,  MemberLastName,
MemberEmail,  MemberPassword from PapaDarioMembers";
            _conn = new SqlConnection(cs);
            //adapter for Members Table.
            _adapter = new SqlDataAdapter(query, _conn);
            //commandBuilder for Members Table.
            _cmdBuilder = new SqlCommandBuilder(_adapter);
            //calling method to fetch fresh copy of data from database
            FillDataSet();

            //going into a PRODUCTS database
            string productsQuery = "SELECT ProductID, ProductName, ProductSize,
ProductPrice, ProductCategory from PopularProducts";
            _ProductsAdapter = new SqlDataAdapter(productsQuery, _conn);
            //commandBuilder for Members Table.
            _ProductscmdBuilder = new SqlCommandBuilder(_ProductsAdapter);
            //calling method to fetch fresh copy of data from database
            FillProductDataSet();



  //CREATING THE DATA SET FOR THE MEMBERS DATABASE
        public void FillDataSet()
        {
            //resets the data set

            _dataSet = new DataSet();
            _adapter.Fill(_dataSet);
            _tblMembers = _dataSet.Tables[0];

            //define primary key
            DataColumn[] pk = new DataColumn[1];
            pk[0] = _tblMembers.Columns["MemberID"];
            _tblMembers.PrimaryKey = pk;

        }

        //CREATING THE DATA SET FOR THE PRODUCT DATABASE
        public void FillProductDataSet()
        {
            //resets the data set

            _ProductdataSet = new DataSet();
            _ProductsAdapter.Fill(_ProductdataSet);
            _tblProduct = _ProductdataSet.Tables[0];

            //define primary key
```

```
            DataColumn[] pk = new DataColumn[1];
            pk[0] = _tblProduct.Columns["ProductID"];
            _tblProduct.PrimaryKey = pk;
        }
```

Getting Information based on different Parametrs

Here are some basic methods to get items from a database and returning the intended items.

```
public double GetProductPriceByName(string productName, string size)
        {

            // string express = "ProductID = '" + productName + "' AND ProductSize = '" +
size + "'";
            string exp = "ProductName = '" + productName + "' AND ProductSize = '" + size
+ "'";

            DataRow[] foundRows;
            foundRows = _tblProduct.Select(exp);

            return Convert.ToDouble(foundRows[0][3]);
        }


        //Gets the SIZE of the ITEMS from the database to be displayed in the pizza page.
        public List<string> getProductsSize(string size)
        {
            List<string> stringSize = new List<string>();

            foreach (DataRow item in _tblProduct.Rows)
            {
                stringSize.Add(item["ProductSize"].ToString());

            }
            return stringSize;
        }
        //getting column headers for the Product Tables
        public List<string> getProductTableHeaders()
        {
            List<string> prodTable = new List<string>();


            foreach (DataColumn item in _tblProduct.Columns)
            {
                prodTable.Add(item.ToString());

            }
            return prodTable;
        }


        //will get the IDs of the suggestion table to be shown in the observable list
        public List<string> getProductRowID()
        {
            List<string> prodID = new List<string>();
```

```
            foreach (DataRow item in _tblProduct.Rows)
            {
                prodID.Add(item["ProductID"].ToString());

            }
            return prodID;
        }
```

Authenticating the User

This method will make sure that the user is authenticated and returna Boolean if the item is in the datarows or if it is not

```
public bool AuthenticateUser(string firstName, string password)
        {
            //debug
            // string express = "FirstName = '" + firstName + "'";
            string exp = "MemberEmail = '" + firstName + "' AND MemberPassword = '" +
password + "'";

            DataRow[] foundRows;
            foundRows = _tblMembers.Select(exp);

            for (int i = 0; i < foundRows.Length; i++)
            {
                Console.WriteLine(foundRows[i][0]);
            }

            if (foundRows == null)
            {
                return false;
            }
            else if (foundRows.Length < 1)
            {
                return false;
            }
            else
            {
                return true;
            }
        }
```

Editing and Deleting Members

```
   //edits the suggestion column of the suggestions.
        public void EditSuggestion(int id, string suggestion)
```

```
        {
            //find a row based on its primary key
            DataRow row = _tblSuggestion.Rows.Find(id);
            if (row != null)
            {
                row["SuggestionID"] = id;
                row["Suggestion"] = suggestion;

                _SuggestionAdapter.InsertCommand =
_SuggestioncmdBuilder.GetInsertCommand();
                _SuggestionAdapter.Update(_tblSuggestion); //saving the changes to the
database.

                FillSuggestionDataSet(); //refreshes/updates the dataset

            }
            else { Console.WriteLine("\nInvalid SuggestionID\n"); }
        }


        //this one DELETE
        //deletes a Member by ID from DariosMembers
        public void DeleteMember(int id)
        {
            //find a row based on its primary key
            DataRow row = _tblMembers.Rows.Find(id);
            if (row != null)
            {
                row.Delete();
                _adapter.DeleteCommand = _cmdBuilder.GetDeleteCommand();
                _adapter.Update(_tblMembers); //saving the changes to the database.
                //refreshes dataset
                FillDataSet();

            }
            else { Console.WriteLine("\nInvalid MemberID\n"); }
        }
```

Programming Logic

Programming Logic Design

The programming logic of my program is quite simplistic. For my classes, I incorporated the entity Framework classes, they are designed to start off from the products class. This class will have the base attributes that all the subclasses will inherit. The products class will also include all the main methods that are used throughout the application as a way to save space in my code. The classes are entered in a pyramid hierarchy where products are at the top, and all types of products are inheriting the abstract methods (Pizza, Sandwich, Wings). The members will also have a hierarchy with members at the top and Member history. Receipts is its own entity as it incorporates both the classes items but uses them for a different reason. The main reason for the classes is to store the values in one object, so I can store the objects in a List, to be able to add and remove items from the display of the cart. This makes it easy for me to add and delete items form my cart since all I am doing is removing a specific product.

Class Programming Logic

As Discussed in the Programming Logic section, the Classes are all similar but with different values and depending on if they are being edited by the admin, they have methods for observable lists as well.

Class Orginization

I have organized the classes and xaml files like so

The second letters signifying the subclass.

c_Name=main class

cp_Products= Products Class/Section

db_DataAccess=DatabaseFile

The xaml files I signified with a x at the front followed by the first two letters, and if they can be grouped I did that as it makes it easier to navigate through the pages.

xab_X= about me

xad_name=AdminPages

xlo_name=loginRelated

xmh_name=memberhistory

xpp_name=Popular Products

Detailed Classes Explanations

Admin Page's

Execute

The Execute function executes the users calls methods from the data access class that would preform CRUD operations for the admin, changing the items in the database.

```
private void Execute(object sender, TappedRoutedEventArgs e)
        {
            //getting the ID of item to be edited.
            int id =Convert.ToInt32(RowID.SelectedValue);
            //getting the data if needed
            string data = Data.Text.ToString();

            //THE TYPE OF ITEM BEING EXECUTED
            if (Type.SelectedItem.Equals("Edit"))
            {
                da.EditSuggestion(id, data);

            }else if (Type.SelectedItem.Equals("Add"))
            {
                da.InsertSuggestion(data);
            }
            else
            {
                //delete
                da.DeleteSuggestion(id);
            }

            InventoryList.ItemsSource = s.GetSuggestion("Server = DESKTOP-
AD228UT;Database=PapaDarioPizza;User ID=PapaDario; PASSWORD=12345");

        }
```

Initialize Component

At the loading of the page, I want to show the suggestions that the users have given, so I do that in the Initialize component method. This method is called the getComboboxes method which will get the method.

```
  public xad_AdminMainPage()
        {
            this.InitializeComponent();
            c_Suggestions s = new c_Suggestions();
            InventoryList.ItemsSource = s.GetSuggestion("Server = DESKTOP-
AD228UT;Database=PapaDarioPizza;User ID=PapaDario; PASSWORD=12345");
```

```
            getComboboxes();
            //ADS THE PRODUCTS TO THE CART
            //  cp_Product.DisplayCart(myCart, MainCart, RemoveButton, toppings);
            //if the user is logged in the members first and last names are displayed in
the top corner.

            //SETTING THE SUGGESTION COMBO BOXES TO LINK TO DATABASE
            //LIST FOR SIZE


        }


        //creating pizza for instance
        public static xpco_CreatePizza cp = new xpco_CreatePizza();
        //public List<cp_Pizza> myCart = cp.GlobalCart();

        // private List<cp_Pizza> myCart = new List<cp_Pizza>();
        private List<c_Toppings> toppings = new List<c_Toppings>();
        public xlo_LoginPage login = new xlo_LoginPage();
        public db_DataAccess da = new db_DataAccess();

        //boolean to hold if the user will get the discount, should be global, so it
would be kept, but I was lazy.


        public bool UserAuthenticated = xlo_LoginPage.IsMemberLoggedIn();
        public int id = xlo_LoginPage.MemberId();
```

GetComboBoxes

At the loading of the page, I want to show the suggestions that the users have given, so I do that in the Initialize component method. This method will call the getComboboxes method which will populate all the combo boxes dependant on the table being called for.

```
        private void getComboboxes()
        {
            //List<AddToCart> choppedNames = new List<AddToCart>;
            List<string> TableHeaders = da.getSuggestionTableHeaders();

            List<string> HeaderList = new List<string>();

            // string list = "";

            for (int i = 0; i < TableHeaders.Count; i++)
            {
                if (!HeaderList.Contains(TableHeaders[i]))
                {
                    HeaderList.Add(TableHeaders[i]);

                }
            }

            //adding the headers to the Binded Headers Combobox
            ColumnName.ItemsSource = HeaderList;
```

```
    List<string> StringIDs = da.getSuggestionRowID();

    //adding ID


    List<string> IDList = new List<string>();

    // string list = "";

    for (int i = 0; i < StringIDs.Count; i++)
    {
        if (!IDList.Contains(StringIDs[i]))
        {
            IDList.Add(StringIDs[i]);

        }
    }


    List<string> stringColumn = da.getProductTableHeaders();
    //  cookie.Text = stringColumn[0].ToString();

    //adding the headers to the Binded Size
    RowID.ItemsSource = IDList;

    //ADDING the TYPES of data editing you can do

    List<string> Types = new List<string>();

    Types.Add("Edit");
    Types.Add("Remove");
    Types.Add("Add");

    Type.ItemsSource = Types;



}
```

MAIN PAGE AND NAVIGATION PAGES

Initialize Component

At the loading of the page, I want to display the cart from the last page, I want to check if the member is logged in to display the first name and last name on the page

```
public MainPage()
        {
            this.InitializeComponent();
            //ADS THE PRODUCTS TO THE CART
            cp_Product.DisplayCart(myCart, MainCart, RemoveButton, toppings);
            //if the user is logged in the members first and last names are displayed in
the top corner.
            if (xlo_LoginPage.IsMemberLoggedIn() == true)
            {
                string names = da.GetMemberFirstAndLastByID(xlo_LoginPage.MemberId());
                TextBlock memberName = new TextBlock();
                memberName.Text = "Member Logged in:" + names;
                memberName.FontSize = 20;

                MemberName.Children.Add(memberName);
            }
            //creating the pizza list at the instansiation of the file
            //passing the cart from page to page.
            //       CreatePizza cp = new CreatePizza();
            //      List<AddToCart> myCart = cp.GlobalCart();
            xpco_CreatePizza cp = new xpco_CreatePizza();
             myCart = cp.GlobalCart();
```

Special Pictures (on hover/off hover)

Allows the on hover and off hover effect on the images in my main page.

```
private void InsidePictureSpecial(object sender, PointerRoutedEventArgs e)
        {
            this.SpecialPizzaImg.Source = new BitmapImage(new Uri(this.BaseUri,
"Images/MainPageImg/NewPics/SpecialPizzasBefore.png"));
        }

        private void OutSidePictureSpecial(object sender, PointerRoutedEventArgs e)
        {
            this.SpecialPizzaImg.Source = new BitmapImage(new Uri(this.BaseUri,
"Images/MainPageImg/NewPics/SpecialPizzasHover.png"));
        }
```

In EVERY PAGE we have

- A method to remove item
- A method to calculate total when Cart is clicked

- A method to Checkout the product

- A method to Update the text boxes.

```
  //calls the addtocart method that will remove the item from the list.
        private void RemoveProductCart(object sender, RoutedEventArgs e)
        {
            cp_Product.RemoveProductCart(myCart, MainCart, RemoveButton);
        }


        //Calculates the total when you click on the cart icon.
        private void CalculateTotal(object sender, TappedRoutedEventArgs e)
        {


            if (xlo_LoginPage.authenticateMember == true)
            {
                HiddenDiv.Visibility = Visibility.Collapsed;
            }
            else
            {
                HiddenDiv.Visibility = Visibility.Visible;
            }

            double total = cp_Product.CalculateTotal(myCart);
            Total.Text = "Total:  " + total.ToString();
        }


        private void Checkout(object sender, TappedRoutedEventArgs e)
        {

            //FINAL CHECKOUT SENDS TO RECIEPT
            cp_Product.CheckoutCart(UserAuthenticated, EmailTextBox, ErrorMessage,
UsernameTextBox, PasswordTextBox, RecieptMemberID, RecieptMemberName, myCart);

        }


        private void UpdateTextBoxes(object sender, TappedRoutedEventArgs e)
        {
           cp_Product.UpdateCartView(UserAuthenticated, EmailTextBox, ErrorMessage,
UsernameTextBox, PasswordTextBox, RecieptMemberID, RecieptEmail, RecieptMemberName,
RecieptPrice, RecieptDiscount, RecieptTotal, RecieptOrder, myCart);
        }
```

Sign Up Page

Passing Members to Other Pages

This part of the code is to pass methods from one page to the other, the theory is that we can pass the items using getters and setters, but since they are not seen from one application to another, you have to make them public and static as the cart can be accessed and changed by any user. The log in and id only have getters.

```
 //used to be AuthorizedMem
        public static bool IsMemberLoggedIn()
        {
            return authenticateMember;
        }

        //used to pass the the Member's ID from page to page
        public static int MemberId()
        {
            return Convert.ToInt32(id);
        }

        public void SetMemberLogin(string username, string password)
        {
            authenticateMember = true;
            id = da.GetMembersIDByLogin(username, password);
        }
```

Logging In And Checking Database

This method ToSignedInPage, will redirect you if the username and password match something in the database, if the username and password are specific then you will be sent to the admin page.

```
/this is where the user will be directed if their sign in is validated
        private void ToSignedInPage(object sender, TappedRoutedEventArgs e)
        {
            //flags for if both of them are true. then go through



            //getting the information from the text boxes
            string username = UsernameTextBox.Text;
            string password = PasswordTextBox.Password;


            //giving admins Access.
            if (username == "admin" && password == "admin" || username == "Dario20" &&
password == "abcde")
            {
                //< Button x: Name = "ToAdminPage" Tapped = "toAdmin" Content = "ToAdmin"
Background = "Orange" Foreground = "White" Width = "80" HorizontalAlignment = "Center"
Margin = "0,20" />

                authenticateMember = true;
                //id = da.GetMembersIDByLogin(username, password);
```

```
                xad_AdminMainPage adminPage = new xad_AdminMainPage();
                this.Content = adminPage;


            }
            //checking if the username is in the db
            //if the password matches the given username.
            if (da.AuthenticateUser(username, password) == true)
            {
                authenticateMember = true;
                id = da.GetMembersIDByLogin(username, password);
                //UserLoggedInPage userLogged = new UserLoggedInPage();
                xpco_CreatePizza userLogged = new xpco_CreatePizza();
                this.Content = userLogged;
            }
            else
            {
                authenticateMember = false;
                ErrorMessage.Text = "Your Credentials Dont match anything in the
database";
            }

        }//end of tologgedIN
```

Sign Up Page

The Sign up page allows users to create an account, this will check using regex to make sure all fields are filled and information is accurate

```
private void ToSuccessSignUp(object sender, TappedRoutedEventArgs e)
        {
            //SETTING up the flags
            bool firstNameFlag = false;
            bool lastNameFlag = false;
            bool emailFlag = false;
            bool PasswordMatch = false;
            //ERROR message


            //getting all of the values
            string firstName = FirstNameTextBox.Text;
            string lastName = LastNameTextBox.Text;
            string email = EmailTextBox.Text;
            string password = PasswordTextBox2.Password;




            // //////////////     Validation    //////////////
            ErrorMessage.Text = " ";
            //checking for the first name if there are any letters
            if (Regex.IsMatch(firstName, @"^[a-zA-Z]+$") == false || firstName == " ")
            {
```

```
                ErrorMessage.Text = "Your First Name must be only Letters!";
            }
            else
            {

                firstNameFlag = true;
            }
            //checking for the Last Name
            if (Regex.IsMatch(lastName, @"^[a-zA-Z]+$") == false || lastName == " ")
            {
                ErrorMessage.Text = "Your Last Name must be only Letters!";
            }
            else
            {
                lastNameFlag = true;
            }
            //checking for the Email
            if (Regex.IsMatch(email, @"^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$") == false ||
email == " ")
            {
                //if the emial is written wrong
                ErrorMessage.Text = "Your Email must be formatted like xyz@xyz.com ";
            }
            else
            {
                emailFlag = true;
            }
            //checking for thePassword

            //checking if the password matches
            if (PasswordTextBox.Password == " ")
            {
                ErrorMessage.Text = "Your Passwords cannot be empty";
            }
            else
            {
                if (PasswordTextBox.Password.Equals(PasswordTextBox2.Password))
                {
                    ErrorMessage.Text = "Your Passwords do Not Match";
                    //if they password match.
                    PasswordMatch = true;
                }
                else
                {
                    ErrorMessage.Text = PasswordTextBox.Password + " " +
PasswordTextBox2.Password;
                }
            }

            //if all are true add items into the database, and redirect to other page
            if (firstNameFlag == true && lastNameFlag == true && emailFlag == true &&
PasswordMatch == true)
            {
                ErrorMessage.Text = "Everything went through";
                //create new instance and add all items into the database

                da.InsertMember(firstName, lastName, email, password);
```

```
            //redirect them to new page.

            xlo_RedirectedSignedUp worked = new xlo_RedirectedSignedUp();
            this.Content = worked;


        }
        else
        {

            ErrorMessage.Text = "One of the Items in this page was not met!";
        }



    }//end of signuptag
```

Navigation From Page to Page

Almost All Navigation Methods

```
// ////////////////////////// NAVIGATION STUFF START
//////////////////////////////////////// //


    private void ToAboutUs(object sender, TappedRoutedEventArgs e)
    {
        xab_AboutUs about = new xab_AboutUs();
        this.Content = about;
    }

    private void ToHomePage(object sender, TappedRoutedEventArgs e)
    {
        MainPage main = new MainPage();
        this.Content = main;
    }

    /*        private void ToPopularPizza(object sender, TappedRoutedEventArgs e)
            {
                PizzaPage pizza = new PizzaPage();
                this.Content = pizza;
            }*/



    //create your own main page
    private void ToCreateOwn(object sender, TappedRoutedEventArgs e)
    {
        xpco_CreateOwnProduct createMain = new xpco_CreateOwnProduct();
        this.Content = createMain;
    }
    //to the other foods we give
    private void ToWingsFries(object sender, TappedRoutedEventArgs e)
    {
        xpp_PopularOther otherFood = new xpp_PopularOther();
```

```
    this.Content = otherFood;
}


//to the

private void ToLogin(object sender, TappedRoutedEventArgs e)
{
    xlo_LoginPage login = new xlo_LoginPage();
    this.Content = login;
}

private void ToCombo(object sender, TappedRoutedEventArgs e)
{
    xco_Combos combo = new xco_Combos();
    this.Content = combo;
}



private void ToPopularPizzas(object sender, TappedRoutedEventArgs e)
{
    xpp_PopularPizza cookie = new xpp_PopularPizza();
    this.Content = cookie;
}

private void ToPopularSandwiches(object sender, TappedRoutedEventArgs e)
{
    xpp_PopularSandwiches sandwich = new xpp_PopularSandwiches();
    this.Content = sandwich;
}

private void ToMenu(object sender, TappedRoutedEventArgs e)
{
    xpme_MenuNav menu = new xpme_MenuNav();
    this.Content = menu;
}

private void ToSignUpPage(object sender, TappedRoutedEventArgs e)
{
    xlo_SignUpPage signUp = new xlo_SignUpPage();
    this.Content = signUp;
}

private void ToSuggestions(object sender, TappedRoutedEventArgs e)
{
    xsu_Suggestions support = new xsu_Suggestions();
    this.Content = support;
}
//gets the members history.
private void ToMemberHistory(object sender, TappedRoutedEventArgs e)
{
    xmh_MemberHistory mem = new xmh_MemberHistory();
    this.Content = mem;
}

private void ToCreatePizza(object sender, TappedRoutedEventArgs e)
{
```

```
            xpco_CreatePizza mem = new xpco_CreatePizza();
            this.Content = mem;
        }

        private void ToCreateSandwich(object sender, TappedRoutedEventArgs e)
        {
            xpco_CreateSandwich mem = new xpco_CreateSandwich();
            this.Content = mem;
        }

        private void ToCart(object sender, TappedRoutedEventArgs e)
        {

        }
        private void Check(object sender, TappedRoutedEventArgs e)
        {

        }
        private void CheckMember(object sender, TappedRoutedEventArgs e)
        {

        }

        // /////////////////////////// NAVIGATION STUFF END
/////////////////////////////////////// //
```

Creating Own Pizza Or Sandwich

Add Custom Pizza Method

This method adds all the items that the user selects into an object depending on if it is a pizza, wings, sandwich, and adds it to a list called Cart, this cart is then updated and eventually sent to the receipts.

```
private void AddPizza1(object sender, TappedRoutedEventArgs e)
        {

            //creating product object to store for later use
            cp_Pizza prod = new cp_Pizza();

            //product name
            prod.ProductName = "Custom Pizza";
            //getting the size from the combobox
            if (Size1.SelectedItem == null)
            {
                prod.ProductSize = "medium";
            }
            else
            {
                prod.ProductSize = Size1.SelectedItem.ToString();
            }

            // prod.ProductSize = Size1.SelectedItem.ToString();
            //getting the price based on the name and size
            prod.ItemID = myCart.Count + 1;

            //getting the price of the pizza and adding it to the cart.
```

```
            double total = da.GetProductNameAndSize(prod.ProductName, prod.ProductSize);

            for (int i = 0; i < toppings.Count; i++)
            {
                total = total + toppings[i].ToppingPrice;
            }

            //adding the product price
            prod.ProductPrice = total;


            //adding the product toppings
            prod.ProductToppings = toppings;



            //adding the object to the list

            prod.ItemID = myCart.Count + 1;
            //adding the remove item button
            myCart.Add(prod);
              //unchecking all checkboxes
               check1.IsChecked = false;
            check2.IsChecked = false;
            check3.IsChecked = false;
            check4.IsChecked = false;
              //CLEARS THE CHILDREN SO THEY ARE NOT REPEATED
            check5.IsCheck  toppings.Clear();
           //CLEARS THE CHILDREN SO THEY ARE NOT REPEATED

            MainCart.Children.Clear();
            DisplayCart(myCart, MainCart, RemoveButton, toppings, RemoveProductCart);ed =
false;
```

Adding toppings to the toppings List

This method adds all the items that the user selects into an object depending on if it is a pizza, wings, sandwich, and adds it to a list called Cart, this cart is then updated and eventually sent to the receipts.

```
//this will add the toppings selected into the topping object and add it to the global
list.
        //MAKE INTO FUNCTIONIN product
        private void ToppingAdd(object sender, RoutedEventArgs e)
        {

            c_Toppings top = new c_Toppings();
            top.ToppingID = toppings.Count + 1;
            //adding the items name to the toppings array.
            if (check1.IsChecked == true)
            {
                top.ToppingName = check1.Tag.ToString();
            }
            if (check2.IsChecked == true)
            {
                top.ToppingName = check2.Tag.ToString();
            }
            if (check3.IsChecked == true)
            {
                top.ToppingName = check3.Tag.ToString();
            }
            if (check4.IsChecked == true)
            {
                top.ToppingName = check4.Tag.ToString();
            }
            if (check5.IsChecked == true)
            {
                top.ToppingName = check5.Tag.ToString();
            }

            top.ToppingPrice = 1;
            toppings.Add(top);

        }
```

Display Cart

This method will create the buttons and load the data from the different objects/lists created and will add them all together creating buttons and changing colors and fonts for each one to make them more appealing.

```
 public static void DisplayCart(List<cp_Pizza> myCart, StackPanel MainCart, StackPanel
RemoveButton, List<c_Toppings> toppings, RoutedEventHandler RemoveProductCart)
        {
            foreach (cp_Pizza prod in myCart)
            {
                //adding the color so I can change the color of the text
                SolidColorBrush orangeBrush = new
SolidColorBrush(Windows.UI.Colors.Orange);
                SolidColorBrush redBrush = new SolidColorBrush(Windows.UI.Colors.Red);

                Button button = new Button();
                button.Content = "X";
                button.Foreground = redBrush;
                RemoveButton.Margin = new Windows.UI.Xaml.Thickness(5, 10, 5, 0);
                button.Tag = "" + myCart.Count + 1;
                //RoutedEventHandler RemoveProductCart
                button.Click += new RoutedEventHandler(RemoveProductCart);
                RemoveButton.Children.Add(button);
                TextBlock emptyTextBlock = new TextBlock();
                emptyTextBlock.Text = " ";
                emptyTextBlock.FontSize = 18;
                RemoveButton.Children.Add(emptyTextBlock);


                //creating a textblock and addin the information into the flyout.
                TextBlock textBlock = new TextBlock();
                textBlock.Text = prod.ItemID + "    " + prod.ProductName + "   " +
prod.ProductPrice.ToString() + "   " + prod.ProductSize + "   ";
                textBlock.Margin = new Thickness(5, 10, 5, 0);
                textBlock.Foreground = orangeBrush;
                //addin a label to the cart
                MainCart.Children.Add(textBlock);

                //creating a textblock to show the toppings you chose.
                TextBlock toppingsText = new TextBlock();
                string topText = " ";
                //adding the toppings into a string
                foreach (c_Toppings top in toppings)
                {
                    topText = topText + " , " + top.ToppingName;
                }
                //setting the toppingText field in xaml to the toppings.
                toppingsText.Text = "Toppings : " + topText;
                toppingsText.Margin = new Thickness(5, 0, 5, 10);


                //adding the textbox into the cart
```

```
                MainCart.Children.Add(toppingsText);
            }

        }
```

Cp_Product

The main outline for each product, each product will inherit the abstract methods in this one and will use the other methods in every class.

```
public abstract class cp_Product
    {
        public abstract int ItemID { get; set; }
        public abstract string ProductName { get; set; }
        public abstract string ProductSize { get; set; }
        public abstract double ProductPrice { get; set; }

        public List<c_Toppings> ProductToppings = new List<c_Toppings>();

        public static db_DataAccess da = new db_DataAccess();


        public static List<cp_Pizza> myCart = new List<cp_Pizza>();
```

Calculate Total

This method adds all the items that the user selects into an object depending on if it is a pizza, wings, sandwich, and adds it to a list called Cart, this cart is then updated and eventually sent to the receipts.

```
  public static double CalculateTotal(List<cp_Pizza> myCart)
        {

            double total = 0;
            foreach (cp_Pizza items in myCart)
            {
                total = total + items.ProductPrice;

                //textBlock.Text = i.ProductName + "   " + i.ProductPrice.ToString() + "
" + i.ProductSize + "   ";
            }
            return total;
        }
```

RemoveProductCart

This method will simply remove the items from the cart.

```
//this method will add all the items in the cart and add them together to create the
Price for each product
```

```
        public static double CalculateTotal(List<cp_Pizza> myCart)
        {

            double total = 0;
            foreach (cp_Pizza items in myCart)
            {
                total = total + items.ProductPrice;

                //textBlock.Text = i.ProductName + "  " + i.ProductPrice.ToString() + "
" + i.ProductSize + "  ";
            }
            return total;
        }
```

Add Custom Pizza Method

This method will check if the user's authentication is correct, then it will make the global variable for authentication = true, as well as set the member ID. This function will also create a receipt object which will create a receipt from the data that the user writes and the cart. If the user is not logged in, the method will unhide the login in the cart so you can also login while you are browsing the cart.

```
public static void CheckoutCart(bool UserAuthenticated, TextBox EmailTextBox, TextBlock
ErrorMessage, TextBox UsernameTextBox, PasswordBox PasswordTextBox,
           TextBlock RecieptMemberID, TextBlock RecieptMemberName, List<cp_Pizza> myCart)
        {
            //FINAL CHECKOUT SENDS TO RECIEPT
            //checks if the user's authentication is correct, if its not it makes the
Error message say that the box is empty
            if (UserAuthenticated == false && (EmailTextBox.Text == null ||
EmailTextBox.Text == " "))
            {
                ErrorMessage.Text = "The Email Box Is not correct";
            }
            else
            {

                //getting the information from the text boxes
                string username = UsernameTextBox.Text;
                string password = PasswordTextBox.Password;

                //creating the db instance

                //creating the reciept Object
                c_Reciept reciept = new c_Reciept();
                //checking if the username is in the db
                //if the password matches the given username.
                if (da.AuthenticateUser(username, password) == true)
                {

                    if (UserAuthenticated == true)
                    {
                        reciept.Email = da.GetMemberEmailByID(xlo_LoginPage.MemberId());
                    }
```

```
                    reciept.MemberName =
da.GetMemberFirstAndLastByID(xlo_LoginPage.MemberId());

                    //if they are a member add the first and last name as well as the id
to the box
                    reciept.MemberID = xlo_LoginPage.MemberId();
                    RecieptMemberID.Opacity = 1;
                    RecieptMemberName.Opacity = 1;


                    RecieptMemberID.Text = "MemberID: " + reciept.MemberID.ToString();
                    RecieptMemberName.Text = "MemberName: " +
reciept.MemberName.ToString();
                    ErrorMessage.Text = "Valid Member, Discount Applied";
                    UserAuthenticated = true;


                    //adding the email to the Reciept Object
                    reciept.Email = UsernameTextBox.Text.ToString();
                    //getting the total price of the cart
                    reciept.Price = cp_Product.CalculateTotal(myCart);
                    //getting how much is being taken off due to the discount
                    reciept.DiscountPrice = cp_Product.CalculateTotal(myCart) * .10;
                    //getting the Total price
                    reciept.TotalPrice = reciept.Price - reciept.DiscountPrice;

                }
                else
                {
                    //using the Email TextBox if they are not a member.
                    if (EmailTextBox.Text == null)
                    {
                        ErrorMessage.Text = "Please write your email in the bottom Email
TextBox";
                    }
                    else
                    {
                        RecieptMemberID.Opacity = 0;
                        RecieptMemberName.Opacity = 0;

                        reciept.Email = EmailTextBox.Text.ToString();
                        reciept.Price = cp_Product.CalculateTotal(myCart);
                        reciept.DiscountPrice = 0;
                        reciept.TotalPrice = cp_Product.CalculateTotal(myCart);
                    }
                }
                string order = "";
                string toppings = "";
                foreach (cp_Pizza item in myCart)
                {
                    order = order + item.ProductName;


                }



                foreach (cp_Pizza item in myCart)
```

```
                {
                    System.Collections.IList list = item.ProductToppings;
                    for (int i = 0; i < list.Count; i++)
                    {
                        toppings = toppings + item.ProductToppings[i].ToppingName;
                    }
                }


                reciept.Order = order;
                reciept.Toppings = toppings;


                da.InsertReciept(reciept.MemberID, reciept.Email, reciept.Order,
reciept.Price, reciept.DiscountPrice, reciept.TotalPrice, reciept.Toppings);
                da.InsertMemberHistory(reciept.MemberID, reciept.Order,
reciept.TotalPrice, toppings);

            }
        }
```

UpdateCartView

This is how the user will update the cart view. It is very similar to the Checkout Method, but I wanted to have two methods to confirm what the user wants is what is shown.