UNIVERSITY COLLEGE OF NORTHERN DENMARK

Programming and Technology Report

Computer Science AP Degree DMAI2019 Group 6

3rd semester project, 2019

Authors

Jan Kalasnikov
Valentin Yordanov
Denis Kovacek

Teachers

Nadeem Iftikhar
Michael Holm Andersen

Table of Contents

Introduction

Nowadays around the world, each day there are more natural disasters where many people, animals, and nature suffer. Nature often receives huge damages that are often irreversible. On the other hand, if the damage is in one way or another recoverable, it takes decades or even centuries to regain its normal appearance. Most of these disasters are caused directly or indirectly by human negligence. There are many people who want to help, whether it's volunteering or with money support in order to help the affected people. The technology nowadays is one of the tools that can be used to help nature.

Problem Statement

The main purpose of this project is to create a WCF web service where all the functionality of the backend will be attached. This service in our case will be in touch with our dedicated client and web client, but also it can communicate with all different clients who are able to use it. The web service gives the opportunity to all clients to be in touch with the same backend but from different platforms. Our service will enable the user to donate to a specific disaster(s) or to donate to all of the disasters in our system with the online shop option. Other ideas for the system are to be able to allow hold two different types of users (admin and regular user).

Problem Area

The project idea is a system where the user will be able to see a list of disasters from around the world in one place alongside some information about them, statistics or the status of the current situation. An option for the user to donate money for a disaster of his choice is the main feature. The visitor has the option of creating a profile where he will be able to enter his personal information such as name, address, email and bank account information and become a user. The core functionality in the system "SaveTheWorld" is a donation by the user for a particular disaster. When the user handles his donating, he will be able to choose to donate to a specific disaster and to insert the amount of money that he wants to donate. The system will run on both web client (ASP.NET MVC) and desktop client (WPF) connected together with a service (WCF).

Another option will be that the person who wants to donate money each month through the desired period of time, will be able to use the option to subscribe. The chosen amount of money will be withdrawn from their bank account and the money will be distributed equally to each of the world's disasters in the system.

The next functionality in the system will be an online shop where the customer will be able to make an order to purchase merchandise. The user will be able to choose attributes, such as color and the quantity of each product. The amount of money he will pay will be equally distributed to each disaster in the system.

The system could take part in education becoming a place that everyone can visit and raise awareness about all the disasters around the world.

Permission access will be granted to two types of users. The regular user, who will have permission to shop, donate to disasters, subscribe and manage his profile. The administrative user, who will have the same rights as the regular user. However, he's privileged with manage options, where he can apply CRUD operation for all users, products or disasters.

Method and Theory

After discussing the problem with other team members, we reached a consensus on the system we will build. In order to manage the workflow in our team, we were presented to two different approaches, Agile and Plan-driven approach. Our learning goal is to get familiar with the Agile approach since it's being used by many companies nowadays. The fact that it is much more flexible and scalable to work with, we decided to pick SCRUM methodology as it is one of the most common agile methodologies. The other options were XP and Kanban. We concluded that XP has a too strict priority order for our needs and Kanban was less suitable for scaling. In a nutshell, we chose SCRUM because it provided us with more freedom in constructing the solution and dividing work into smaller cycles during sprints. That is why we decided to use SCRUM combined with a few artifacts from the Plan-driven approach which will help us to get a much clearer idea of the whole system. In terms of database, we chose SQL Management Studio (SQL Scripts). We decided so because of scalability, security, and reliability in which we have assured ourselves during the previous projects. However, we considered another option which was MongoDB. After giving it a study we figured there is no default transaction support and joins were not supported so we concluded that it is not the right database for our application. We also used recommended literature by our teachers from Sommerville, Troelsen, and Miles to expand our current knowledge and create a work that would correspond with a university level.

Architecture and design

When it comes to building a system, architecture is without a doubt one of the most fundamental things about it. We have looked for a type of architecture that is scalable and complex to the point where every team member fully comprehends its purpose.

4-Layer architecture

In this project, we decided to design a 4-layer, system architecture. The reason we made such a decision was that it offers low coupling and high cohesion. In addition, refactoring is also much easier with layers since they are more manipulative. Another reason for going with this type of high-level architecture is maintainability. In our case, each layer communicates with only the layer below it. That allowed us to make changes in one layer without affecting the other layers, making them more independent. The architecture below is representing all layers in our system with their respective names.

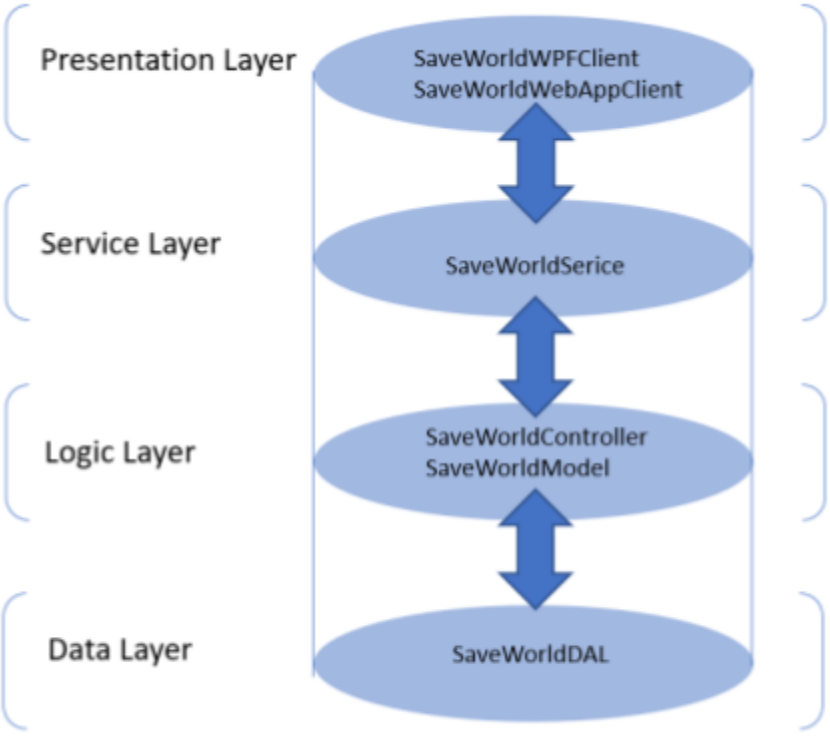

Figure 1 - Architecture

Presentation Layer

Presentation Layer is intended to hold our clients or client projects to be more precise. Both dedicated and web clients will be held in this layer. We have used WPF for a dedicated client and ASP.Net MVC for web clients. The reason for choosing these frameworks will be further explained in the Clients section in the report. The whole point of the Presentation Layer is to make an interface for a user. It is also a place where all of the program's functionality will be graphically presented in a user-friendly way. Our type of system architecture allowed us to make changes in the Presentation Layer without changing the rest of the layers since it only communicates with the Service Layer.

Service Layer

We have been working in Microsoft's Ecosystem, therefore the choice of Windows Communication Foundation has been the logical step for us to take. We used this layer to allow the clients to the business logic for both Desktop and Web clients. The Service Layer only communicates with the Logic Layer. In our service, we are implementing all the necessary service operations from our controller or Logic Layer. For example, the user in the client will be checked first with the service operation "CheckLogin". What it does is checks if the user already exists. In our services, we relate all the data that the user request, with the Logic Layer where we actually manage and control it.

Logic Layer

Adding the Logic Layer gave us a better understanding of how quality architecture should look like. In this layer, we control and handle the information coming from the Service Layer and send it to the Data Layer, where the whole process with handling data is happening. The Logic Layer in our case helps us control all the methods for the system. For example, in the user controller, we have a method "GetUser". The user controller is sending an id as the user's parameter and returns the object which is found or defined in the user data class where all operations are happening.

Data Layer

One of the layers in our architecture is the Data Layer, where the whole processing part with the different CRUD operations is happening. For this particular project, we are using Entity Framework to interact with the database. This layer in our project is the most critical part of the whole system. The reason that we chose to implement all the manipulations with the database here, is that this is the layer that communicates with our database. Another reason is that all the members in the group have agreed that the controller or the Logic Layer is being used just to navigate the data from the user and to the user, so none of the data interactions really happen there.

Database

To store the user's data, we needed to choose a tool that would allow us to handle this requirement. We decided to go with a standard relational database as it has data normalization and ACID support. In terms of software, we picked MSSQL mainly because it is a part of the Microsoft ecosystem and based on the fact that we had experience with it, from the previous semester. We were aware and considering using NoSQL databases such as MongoDB since it supports automatic scaling, high performance or uses JSON data formatting. As a pattern for designing the Database, we used Relational Model as it gives us an overview of the Database layout.

ADO.NET vs Entity Framework

We're using the database in order to store the users, items for the shop, or disasters. In .NET we have 2 choices when it comes to Databases, we can choose either from ADO.NET or Entity Framework. .NET framework provides rich support for working with databases. We can use a part of .NET known as ADO.NET to work with data in relational databases. For years it was used by developers, but its major drawback is that it requires spending time writing repetitive code - queries. Additionally SQL queries are written in string syntax, which prevents the Visual Studio to assist us with detecting any potential syntax errors. That's where Entity Framework stepped into the board, where it makes the data more accessible for use. Simply when using EF, we could interact with a set of models/entities and a DB context class, instead of interacting directly with the database, EF works as a "mediator" - middleware overseeing the data movement from DB to the entity classes and back again. Entity Framework scripts are written using C# Language Integrated Query - LINQ. Entity Framework is based on top of the ADO.NET which means that it is newer than ADO.NET. The other positive fact is that as it was also a new feature for us and we thought that we can benefit from this knowledge in the future.

Decision

For this particular project, we decided to use Entity Framework to persist the user's data and connect it with the business logic layer. We had the opportunity of choosing EF or ADO.NET. Nevertheless, we understand the power of both technologies and how easy it is to manipulate the data through them both. But the reason why we picked EF, in the end, was that it allows us to build a solid foundation for the data access layer, provides auto-generated code, reducing development time and cost, and a developer can visually see the model as well as the database mapping. Another reason to use EF was the fact that it allowed us to choose whether we want to do Model first or Database first approach.

Model-first or Database-first

When choosing Entity Framework we have two options when it comes to mapping data. Those two options are Database first or Model first. The model first uses the model class and designs the tables based on those properties. On the other hand, the Database first approach will allow you to design your database layout with boxes and lines, which is also the reason why we used Relational Diagram to create a layout of the future database. The reason why we decided not to go with the Model first approach is that dealing with diagrams can be tricky, especially when we want to have control over our Model classes and it's really hard to make manual changes along the way. When it comes to building large scaling and complex systems, then the database first approach was better for us as we can make any changes manually on the way. Another reason that it can be better to use Database first is that we can have attributes (PK, FK) under better control.

Decision

The group decided that we will write the database ourselves which meant taking the Database-first approach. The primary and foreign keys are created automatically in the Model-first approach and we wanted to have control over them. One of the main reasons for going with the Database-first approach was that we were anticipating some changes in the database as the project was ongoing. Having that mindset we knew that with the Database-first approach these changes can be easily done and updated with a single click on the application end, unlike in the Code-first approach. The latter would not reflect any modifications done in the database, at least on the application end. It led to a decision of creating our database and to make sure that all the necessary properties that we need will be written with the correct keys and in the correct format.

Database Model

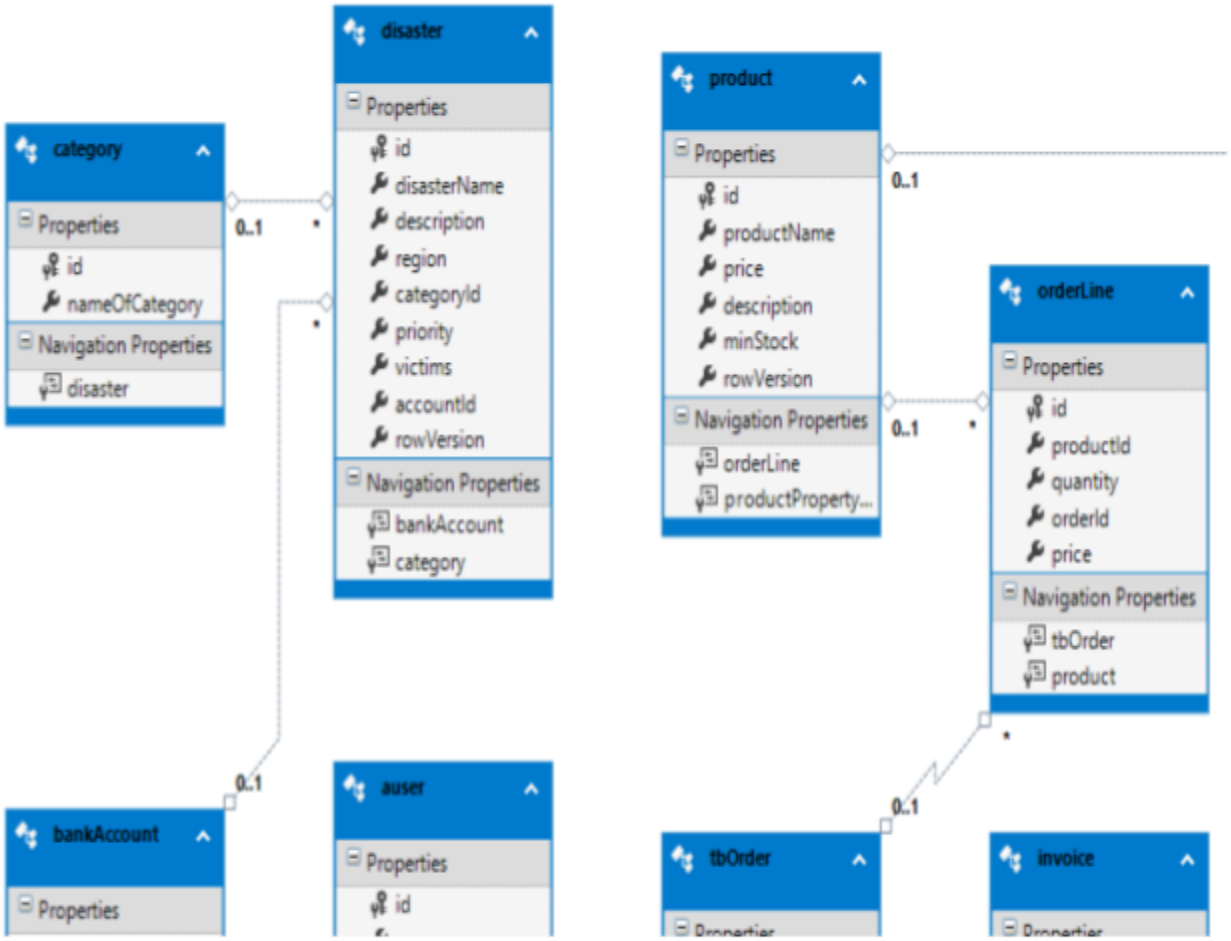

Figure 2 - Database Model

This is a part of our Database Model. The whole Model can be found in the appendix, in code, in the Data Layer with the name “SaveWorldModel.edmx”. When choosing the Database first approach we were forced to design a database model as a part of EF. DB context classes are automatically generated. This way of designing the database was much more convenient for us as we had a great overview of all the things we had there. The layout also corresponds with our Domain Model and Relational Diagram. We considered it to be a critical point as we were able to see the layout. As we mentioned the DB model was automatically generated from the database through Entity Framework. During the implementation of the system, we would often realize that we need to change a field or add a new one in the database. Because of the fact we were using Entity Framework, we had to update the DB model each time changes were being made in the database. But overall, everything worked flawlessly and we were satisfied with how this Model update was happening.

Service

In our system, we want to allow the same functionality of the system for both Web client and Desktop client. That's where various frameworks come in hand, such as WCF, Service Stack or Web API.

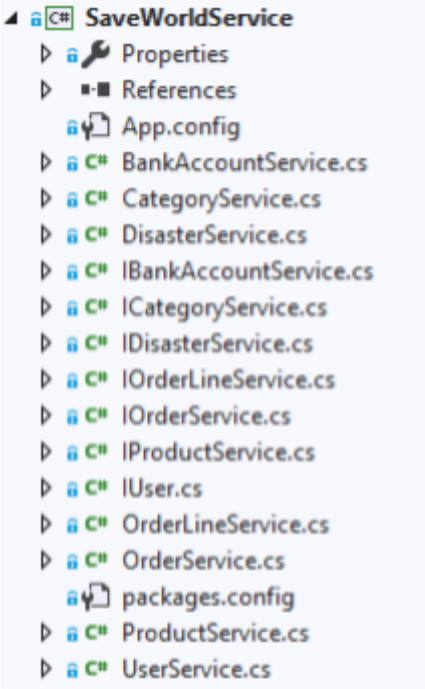

Figure 3 - Service classes

Services expose data through API using common approaches or standards. Decoupling our front-end from our back-end makes it easier for our services to support multiple client apps. Services can also allow other companies to consume and use our services.

In the picture to the side, we can see all of our service classes. As it is visible there is an interface to each service, where the methods are being exposed. This way we can maintain all the changes that are made to the code.

REST and SOAP

We explored both of the options while trying to figure out whether REST or SOAP methodology would be more suitable for our implementation. Overlooking the fact that REST is being preferred over SOAP, WCF is by default a SOAP-based technology. [1]There was also an alternative with Service Stack as it gained popularity in the past few years, unfortunately, serialization/deserialization of request/response back and forth to JSON or XML format performed by ServiceStack is bound to come at the cost of performance compared to pure C#

[1] "Web Service vs WCF Service - Stack Overflow." 17 Dec. 2012, https://stackoverflow.com/questions/351334/web-service-vs-wcf-service. Accessed 15 Dec. 2019.

objects that MVC controllers deal with[2], so we stuck with WCF. When using WCF the developer needs to set-up endpoints from which the data move between the service and clients. Another requirement when specifying an endpoint, we need to set up the so-called ABC.

SOAP

SOAP is a protocol or in other words, is a definition of how web services talk to each other or talk to client applications that invoke them.[3] SOAP is being used by WCF, which is the main reason why we use it in our project. Nevertheless, there were also many more reasons for choosing SOAP technology, that with SOAP, for example, our messages can be sent to the service using any transport protocol so we aren't tied to HTTP. They are encoded in an XML. There are lots of web standards around SOAP - standards for putting security, sessions and other features into the header of the message, for example. SOAP service supports ACID compliance and it's much more powerful in terms of security. For us, security was the primary concern as we are manipulating with any potential donations - real money. For this reason, we have decided to select SOAP service as it supports a higher level of security such as mentioned ACID or SSL certificates. ACID is more conservative than other data consistency models, which is why it's typically favored when handling financial or otherwise sensitive transactions[4]. Therefore, that was a logical step to take.

WCF Framework

WCF supports multiple transport protocols, such as HTTP, TCP, it can have simple or duplex bindings and its services are SOAP-based. SOAP is a protocol and it stands for Simple Object Access Protocol. The reason we chose is that WCF conforms ABC, where A is the address of the service that you want to communicate with, B stands for the binding and C stands for the contract. This is important because it is possible to change the binding without necessarily changing the code. The contract is much more powerful because it forces the separation of the contract from the implementation. This means that the contract is defined in an interface, and there is a concrete implementation that is bound to by the consumer using the same idea of the contract.[5]

Address-Binding-Contract

In WCF we have "endpoints" we need to specify them in order the data could be transferred between the client and service. When we specify the endpoint in WCF we must enter Address,

[2] "Advantages/disadvantages of using ServiceStack services vs ASP.NET ...." 8 Jul. 2013, https://stackoverflow.com/questions/17530541/advantages-disadvantages-of-using-servicestack-services-vs-asp-net-mvc-controlle.
[3] "SOAP Web Services Tutorial: Simple Object Access ... - Guru99." 18 Nov. 2019, https://www.guru99.com/soap-simple-object-access-protocol.html.
[4] "SOAP vs. REST: A Look at Two Different API Styles - Upwork." 19 Apr. 2017, https://www.upwork.com/hiring/development/soap-vs-rest-comparing-two-apis/.
[5] "What Is WCF And Why Should We Use It? - C# Corner." 3 Dec. 2018, https://www.c-sharpcorner.com/article/what-is-wcf-why-should-we-use-wcf/. Accessed 15 Dec. 2019.

which shows where the service is hosted. In our case, we don't have a specific address. Binding responsible for transporting messages from the client to the server using transport protocols like HTTP, HTTPS. The binding in our services is “basicHttpBinding” since is the default one in WCF. Finally, the Contract is an agreement between the client and the server about the structure and content of the messages being exchanged. The contract that we use is IMetadataExchange and the reason behind that is because we need our service to be able to connect the dedicated client and web client with it. This contract will show the methods and their data types. Together it gives us “ABC” (Address, Binding, Contract). All the endpoints can be found in the app.config file. [6]

Configuration

Using the “App. Config” file we have control over the way the services’ endpoints ABCs are defined and we can change them according to our needs. This file needs to be modified in the case that we deploy the service to an online host, therefore the address needing to be changed and also the binding. The endpoints in our case are created automatically.
To set the endpoints there are two different types. The first one is through the code and the second is to set them in the configuration file. We decided to leave the service to generate them automatically when our system runs the service. Another reason was because of the fact that to implement the endpoints in the code is also not good to practice in the real world because the bindings and addresses for the service are typically different from those used while the service is being developed.[7]

Serialization

In the world of object-oriented programming, everything is represented in objects. Those objects are represented by specific entities with specific attributes. When we are sending the objects to different systems, applications or different transfer protocols, a problem can occur because each of them is using different data types. Here is where serialization takes part. To solve a problem which can occur because of sending different object data type we need to serialize the current object. There are a few types of serialization li binary, XML and SOAP serialization, and JSON serialization.[8]
In our system to prevent problems with transferring data types, we decide that, since we use the .NET Framework to use the default one which serializes all the properties and the entities.

---

[6] "ABCs of WCF - C# Corner." 21 Jul. 2014, https://www.c-sharpcorner.com/UploadFile/rkartikcsharp/abc-of-wcf/.
[7] "Endpoints: Addresses, Bindings, and Contracts - Microsoft Docs." 29 Mar. 2017, https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/endpoints-addresses-bindings-and-contracts. Accessed 15 Dec. 2019.
[8] "Serialization (C#) - Microsoft Docs." 25 Apr. 2018, https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/.

To serialize our model classes we are using Data Contract which converts the data to and from XML. Also for each property that we need to serialize we use Data Member. An example of how we serialize our data types is in the figure below.

```
[DataContract]
— references
public class DisasterB
{
    [DataMember]
    4 references
    public int DisasterId { get; set; }
    [DataMember]
    5 references
    public string Name { get; set; }
    [DataMember]
    4 references
    public string Description { get; set; }
    [DataMember]
```

Figure 4 - Model class

Clients

We have developed two clients built on different platforms, one dedicated using WPF and the other one, web client using the MVC framework. Both clients are connected through our service which allowed us to show and control that our implementation works the same way on both clients.

Dedicated client

To meet the requirements for our current project we have to implement one dedicated client and one web client. To build a dedicated desktop client we have two options, first is to develop it with WPF (Windows Presentation Foundation) or WinForms. The second option is already quite old as it has limited functions, that's why we decided to use WPF. It's a powerful tool to create applications with rich user experience making it the main reason for our choice. The development interface is much more flexible and customizable compared to WinForms. That's why we prefer to use WPF instead of WinForms.
The whole team has participated in building the UI. In the beginning, it was a bit challenging as we were tempted to search several things on our own to build the desktop application. But in the end, we enjoyed working with it, as it's really smart and intuitive.

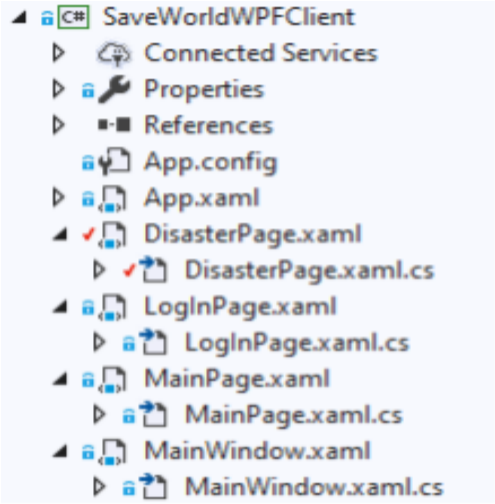

Figure 5 - WPF classes

This is part of our Solution Explorer, where we have our WPF client, which is the dedicated desktop client. We have connected WPF with the WCF service, as you can see in the connected services. That way we can reach out to our controller classes, where is the core logic of the application. Each class has 2 classes, first one is .xaml and second .xaml.cs. The first one is an environment, where we design the layout and the look of the page. We insert lists, buttons and labels, however, we still need to implement the logic behind each button. That's where xaml.cs steps in. In that class, we configure the buttons and attach the logic to them.

Implementation of WPF

These two methods from our system are showing how we connect to the database through the service, through the controller afterward and then the controller communicates with the data access layer where we interact with the data.

```
private void UserList_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (userList.SelectedItem != null)
    {
        userSelect = (string)userList.SelectedItem;
        user = usrClient.GetUserByName(userSelect);
private void Button_Update(object sender, RoutedEventArgs e)
{

    UserService.UserB user = new UserService.UserB();
    user.Password = password;
    user.UserId = userId;
    user.Name = txt_Name.Text;
    user.BankAccountId = accID;
```

Figure 6 - WPF class

The first method shows the action in our list box when the admin chooses the desired user for updating. Then all the data for it is loaded in our textbox.

The second method is the backend of the button “Update”. When the admin finishes his updating information and press it and the desired information will be updated for the user which was selected. But before that, the system checks if the email was changed because in our system each email is unique. Therefore if it's changed the system checks if there is another user who is using it. If there is no user with this email the updating is set. Otherwise, a message box will be displayed saying “This email already exists!”.

Web client

When building the Web Client we had two options on how to build the Web client. It was either with ASP.NET MVC or WebForms. We used MVC since it is much newer and most common when building websites in Visual Studio. Another reason for choosing it is because it offers more control over the HTML and CSS than the WebForms. We didn’t go with WebForms based on the fact that it is more tightly coupled.

ASP.NET MVC is a server-side web framework that utilizes the MVC design pattern. It’s used to process HTTP requests and prepare responses, containing HTML, CSS, JS for rendering the browser. The development environment is much more friendly and customizable in comparison with WinForms.

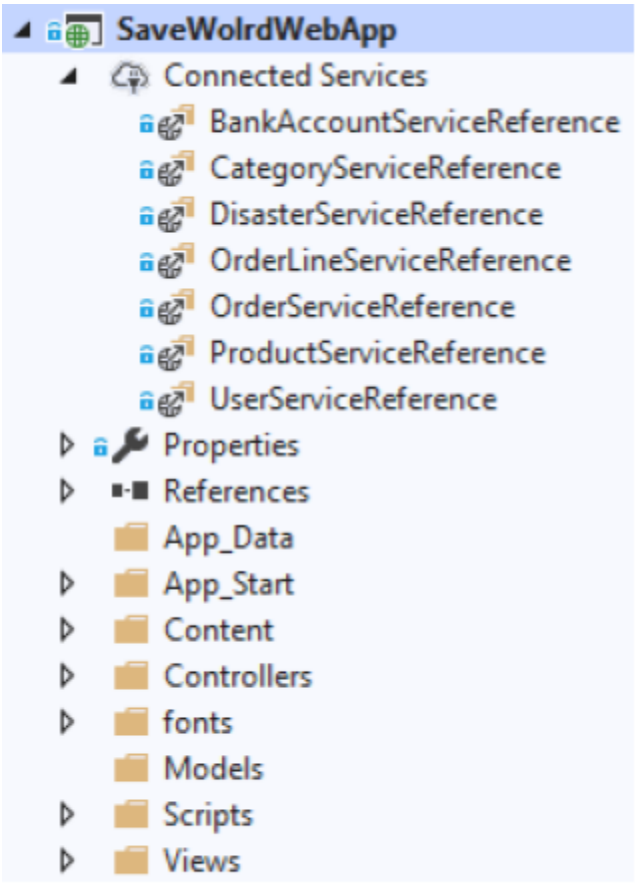

Figure 7 - MVC class structure

As displayed in the picture, we have the same structure. One thing to notice is the model layer. We don't have any classes there, because we’re using the WCF, which allows us to expose the model layer through that service. So it would be inefficient to write again model classes, as it might create conflicts in the future.

Every website or web application has two key participants, the client, and the server. When browsing a website or using a web application, we are using a web browser to communicate with the server. When action is performed on the website, the website formats and sends a request to the server and prepare a response to the user. The format of the request and response follows a specific set of rules known as HTTP or Hypertext Transfer Protocol. When developing a website, there are two places that we can run code. Client and server. On the client, we use HTML, CSS or JS. On the server, we have a lot of options doing server-side back-end development. We can use Java, Python, PHP or even JavaScript using Node.js, but we're using C# - ASP.NET. It comes in several flavors including web forms, web pages or MVC. Out of these options, MVC is the most popular which we used. MVC is used in a wide variety of server-side frameworks including Java Spring Framework or Ruby on Rails for example.

Implementation of MVC

```
public ActionResult Donate(BankAccountServiceReference.BankAccountB bank)
{
    BankAccountServiceReference.BankAccountServiceClient bankClient = new BankAccountServiceReference.BankAccountServiceClient();

    bankClient.donateMoneyToAllDisasters(bank.Amount, 2);

    return View();
}
```

Figure 8 - Controller MVC

We used the MVC pattern, where all the logic for each action is saved in the controller layer. The model class folder was empty as we had it connected with the service (WCF). The view, of course, was necessary to set up the look of the website. For that, we used the Razor syntax. We're calling our service, where we enter values to the attributes. Each action result method stands for any action, that could be called on the website.

Middleware

A perfect example of middleware in our project is Entity Framework. Simply when using EF, we could interact with a set of models/entities and a DB context class, instead of interacting directly with database EF works as a middleware overseeing the data movement from DB to the entity classes and back again.
It lies between an operating system and the applications running on it. Essentially functioning as a hidden translation layer, middleware enables communication and data management for

distributed applications. EF was also a great choice as it supports greatly data encapsulation and security which was critical for our project.

Concurrency

For the system that we will create it is important to update the data as the user wants it because he will mainly operate with his data in the system or with his bank account which means that it is really important that all the updates in this section are handled properly. Our solution was changed during the workflow of the project. The reason for that is explained in detail in the Decision section below.

Optimistic or Pessimistic

Based on our requirements for the project and the options that we have to handle for the concurrency part were mainly two techniques. They come in two flavors, pessimistic and optimistic. With pessimistic concurrency control when the user reads a record, the database is locked. If another user wants to read the same record or to manage data in the same database, he needs to wait until the database is unlocked. Optimistic control allows multiple users to read the same record. When a user initiates an update the data for the record is checked if it has changed since it was initially read. If the data has been changed, the user trying to make the update will receive an error message. The biggest disadvantage in a pessimistic locking database is that the deadlock or race condition can occur. This is often happening because each multi-thread system often accesses the same resource. The optimistic way is preferable as it first checks and then manages.

Decision

Concurrency was one of the main requirements for this project, and we handled it with Entity Framework, which supports optimistic concurrency by default. This was the critical point of the optimistic locking implementation. At the very start of the implementation of the concurrency part, we made a small experiment. The effort that we put it was to implement pessimistic concurrency by using a lock-in our critical area of the specific method. The fact that in a pessimistic concurrency deadlocks can occur and make the system much slower merit to the record locking, we have decided to rationally select optimistic concurrency, even though it meant slightly rewriting a few lines of code. For implementing the optimistic locking we used a well-known timestamp. The synonym of a timestamp is the row version. We chose to add a new column “rowVersion” in our database which stores binary numbers where every time the specific row is updated, the row version is incremented. This was by far the best solution for our case, as it solves the conflicts between our system and users.

Implementation

```
public bool Update(BankAccountB bankAccountBefore)
{
    bool update = true;
    using (var NWEntities = new SaveWorldEntities())
    {
        var bankId = bankAccountBefore.AccountId;
        var accountForSave = (from p in NWEntities.BankAccounts
                              where p.id == bankId
                              select p).FirstOrDefault();

            accountForSave.accountNo = bankAccountBefore.AccountNo;
            accountForSave.amount = bankAccountBefore.Amount;
            accountForSave.ccv = bankAccountBefore.CCV;
            accountForSave.expiryDate = bankAccountBefore.ExpiryDate;
            accountForSave.rowVersion = bankAccountBefore.RowVersion;

            NWEntities.BankAccounts.Attach(accountForSave);
            NWEntities.Entry(accountForSave).State = System.Data.Entity.EntityState.Modified;
        try
        {
             NWEntities.SaveChanges();
        }
        catch
        {
            update = false;
        }

    }
    return update;
}
```

Figure 9 - Code, update method

In our program, we have several controversial places, where data concurrency could be implemented. The core place that all the group members agree that is the most critical part for concurrency being handled, is when we update the user bank account. The money is transferred from the user bank account to the disaster's bank account. If there is a problem here, the whole idea for the system will fail. In this particular method that we are showing we are sending a bank account object. First, we are selecting the object from the database with this object's id. The database is storing the last update information in a binary array. After all the fields are updated we call the Attach method because if we don't, the update will always succeed but without the concurrency control. Then we set the object state to modified, otherwise, it is not going to save the data. Then is time for our critical point. We try to save the changes but if it is not possible the method itself will return false which means that the data was already updated.

Transactions

In our system, the transactions can be used where users interact or take actions with their bank accounts. In case something goes wrong, for example, if the GUI is not responding or another problem appears, the CallBack should run. This means that the data for the bank account

should be saved without any changes. That's why the implementation of transactions in our system could be a good feature in future improvements.

Testing

Testing is one of the most important pieces when it comes to developing an application. We faced an option to do the code first or the test first. We went with the code first approach because we didn't want to possibly lose sight of our goal for the sake of passing a method before it was even written. The main intention of using tests was to check the logic behind the code. The tests were decided to be implemented at the end of the last sprint for what the group decided were the main features of the application. We decided to test tasks in the user stories we considered important and not for every task. Tasks that were tested included user login info check, checking if the disaster already exists (for the admin mode) and creating a new product (also for the admin mode). For this project, we used unit tests in Visual Studio. We had the option of choosing between unit and integration tests. The group decided to use unit testing. The reason for that was that we were looking for a faster, less time-consuming way, but still efficient enough. Knowing that integration tests were more complex and take more time to write and execute the choice was obvious. Using the assert statement within the unit test we were able to check whether the methods that were tested return wanted results. The tests were as previously mentioned done in the end to make sure that the methods work flawlessly. We found unit tests to be extremely helpful and reassuring that the work is on the right track.

```
12
13      [TestClass]
        0 references
14      public class CreatingProductTest
15      {
16          [TestMethod]
            0 references
17          public void CreatingAProduct()
18          {
19              ProductCtrB productCtr = new ProductCtrB();
20              ProductB prod = new ProductB();
21              prod.ProductName = "umbrella";
22              prod.ProductDescription = "a very nice umbrella";
23              prod.Stock = 10;
24              prod.Price = 10.35M;
25              Assert.IsNotNull(productCtr.CreateProduct(prod));
26          }
27      }
```

Figure 10 - Test code

In the picture above it is visible how we conducted one of our tests. We decided to test a method that creates a product. For that purpose, we needed to declare a class as a test class and a method as a test method which can be seen in the square brackets in the picture above. Then we created a new object with its belonging properties. In this case, they were the product name, description of the product, stock and the price. At the end of the method, we have the Assert statement which we used to check if the newly created object exists now that we test it with all the new properties. After we ran the test and it passed we were assured that the method is working as it should. We have conducted several more tests for other methods as stated earlier to check their functionality as well.

Login

In our program, we have implemented an option for the user to create an account. Once he creates the account he can make a donation, use a shop, and he also has an option to update his information. Our program consists of two types of users, admin, and a regular user. You might already understand that admin has more options in the software, he can do the CRUD operations for disasters, user and shop, making it easier for the admin, as he doesn't need to enter the database each time he would like to edit some data. When you enter the application as admin you have an option ‘Manage’ which is hidden to normal users and only accessible by the administrator.

```
public UserB CheckLogin(string userEmail, string password)
{
    UserB userCorrect = null;
    using (var NWEntities = new SaveWorldEntities())
    {
        var user = NWEntities.Ausers.FirstOrDefault(u => u.email == userEmail);
    if (user.password == Hasher(password + user.salt))
    {
        if (user != null)
        {
            userCorrect = new UserB()
            {
                UserId = user.id,
                Name = user.name,
                Password = user.password,
                Salt = user.salt,
                Email = user.email,
                Address = user.address,
                Phone = user.phoneno,
                TypeOfUser = user.typeOfUser,
                BankAccountId = (int)user.accountId,
            };
        }
    }
    }
    return userCorrect;
}
```

Figure 11 Method of Check Login

When the user puts in his credentials, the method checks his email and password. The first thing that we are doing is to check if there is a user with the input email. If the user exists, the password that he inputs will be hashed with salt field which is described in the Security chapter below. If the hashed password from the database is equal to the hashed input password our method returns the user.

Security

To prevent being attacked or get our information stolen by hackers, we should provide our users with a well-secured application. One way of doing so was to protect the password. The best way to secure the user's password is to hash it, when a user creates a password it's being stored in string format, which is not the most secure way.

When a user registers into the system, the password he inputs as his desired password will get ten random characters attached to it and then it will be hashed using the SHA256 algorithm.
Inside the database, the salt (which is those ten random numbers), the hash of the salt and the desired password will be saved for the current new user. The original string which the user typed is lost. When the user wants to log in, the password he inputs will have attached the salt

from the database, it will be hashed and if the hashed value is equal to the hash inside the database, access will be granted.

```
public string SaltGenerator(int size)
{
    string salt = "";
    string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    Random random = new Random();
    for (int i = 0; i < size; i++)
        salt += chars[random.Next(chars.Length)];
    return salt;
}
//Hashes the input using SHA256
3 references
public string Hasher(string input)
{
    StringBuilder hash = new StringBuilder();
    System.Security.Cryptography.SHA256Managed sha = new System.Security.Cryptography.SHA256Managed();
    byte[] crypted = sha.ComputeHash(Encoding.UTF8.GetBytes(input), 0, Encoding.UTF8.GetByteCount(input));
    foreach (byte chunk in crypted)
        hash.Append(chunk.ToString("x2"));
    return hash.ToString();
}
```

Figure 12 - Hash method

Those are the methods that we are using, first to generate the salt string with 10 random characters and second to hash the salt plus the desired password together as a whole variable.
The reason to use these techniques is that C# holds a library System.Security.Cryptography where we can use a couple of hashing functions and it's already implemented.
Since we are using LINQ to Entity Framework and not SQL queries, we are preventing SQL injections to our database. The first reason to use LINQ is that, if in some cases our system is attacked by SQL injection we will keep an eye on them and save our data. The other one was that with SQL queries it could create conflicts in the future and we could be in big trouble if we write wrong queries or have a syntax errors.[9]

[9] "What is Hashing? - Definition from Techopedia." https://www.techopedia.com/definition/14316/hashing.

Conclusion

In this project we found ourselves facing a completely new set of challenges. The semester is only so long as the curriculum sets it and we had to adapt in many ways. The main feature we faced was Visual Studio and C# programming language. Unlike other projects before, we had two clients this time, a dedicated client and a web client.
Using all these new technologies that were mentioned throughout the report, enabled us to create what we believe to be a unique application for the targeted market. Dealing with concurrency, hashing, using services, data access layer, and LINQ together with entity framework helped us a lot behind the scenes. Creating a web client and using MVC enabled us to provide a user with greater experience. We are of a strong opinion that we have reached our goal with this application. It is meant for the people who know what they want without any redundant information and social media posts distracting them from providing to the cause. We created an app that people can use to broaden their knowledge and contribute to the fight in beating the disasters or dealing with their consequences.

We believe that in this report we have explained why we used these technologies, backed up by well-structured arguments. In the future, we would like to work a bit more on the Web client and broaden the specter of features the user could choose from. As part of the improvements for the Web client, we had in mind to create different backgrounds depending on where the user would navigate. We think that it would differentiate us from the competition even more and put our name on the map by just creating a better and more vivid user experience. As for the user features we would like to implement a chat, a subscription option and an option to edit the information about the disasters from the user side. By doing so, we would be able to create a community with the latest relevant information and a proper learning site.

Reference list

1) *"Web Service vs WCF Service - Stack Overflow." 17 Dec. 2012, https://stackoverflow.com/questions/351334/web-service-vs-wcf-service.*

2) "Advantages/disadvantages of using ServiceStack services vs ASP.NET ...." 8 Jul. 2013, https://stackoverflow.com/questions/17530541/advantages-disadvantages-of-using-servicestack-services-vs-asp-net-mvc-controlle.

3) "SOAP Web Services Tutorial: Simple Object Access ... - Guru99." 18 Nov. 2019, https://www.guru99.com/soap-simple-object-access-protocol.html.

4) "SOAP vs. REST: A Look at Two Different API Styles - Upwork." 19 Apr. 2017, https://www.upwork.com/hiring/development/soap-vs-rest-comparing-two-apis/.

5) *"What Is WCF And Why Should We Use It? - C# Corner." 3 Dec. 2018, https://www.c-sharpcorner.com/article/what-is-wcf-why-should-we-use-wcf/.*

6) "ABCs of WCF - C# Corner." 21 Jul. 2014, https://www.c-sharpcorner.com/UploadFile/rkartikcsharp/abc-of-wcf/.

7) “Endpoints” https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/endpoints-addresses-bindings-and-contracts

8) “Serialization” https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/

9) "What is Hashing? - Definition from Techopedia." https://www.techopedia.com/definition/14316/hashing.