3rd Semester Project

Dinnergeddon – Programming & Technology Report

Stefan Nikolaev Borisov
Linda Augustina Carolus Fuchs
Alexander Ignácz
Stefan Jõemägi
Dimitar Bogomilov Pilyakov
Nikola Anastasov Velichkov

UCN – COMPUTER SCIENCES AP DEGREE
DMAI0917 – PROJECTGROUP 7

This page is intentionally left blank.

University College of Northern Denmark

Computer Sciences Academy Profession Degree Programme

Class:
DMAI0917 – Project group 7

Title:
3rd Semester Project – Dinnergeddon
Programming & Technology Report

Project participants:
Stefan Nikolaev Borisov
Linda Augustina Carolus Fuchs
Alexander Ignácz
Stefan Jõemägi
Dimitar Bogomilov Pilyakov
Nikola Anastasov Velichkov

Supervisor:
Dimitrios Kondylis

Abstract

The requirements were to create a client-server architectural product. This was done primarily in Visual Studio and Unity. GitHub was used for version control.

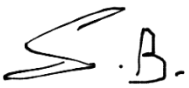
ADO.NET was chosen as the means to connect to the database, because it allows greater control of the system on a base level. WCF was used to expose services to the Internet, while the WPF client and the ASP.NET server consumed those services in order to display information.

Unity was used to create a concept for the game, which served as a proof of concept for the time. A full game could not be integrated due to the limited iterations.

Submission date: 17-12-2018

Number of characters including white spaces, excluding cover, Table of Contents, tables and Appendices:

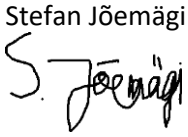
Stefan N. Borisov



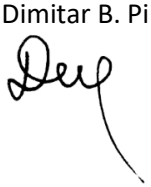
Linda A.C. Fuchs

Alexander Ignácz

Stefan Jõemägi



Dimitar B. Pilyakov



Nikola A. Velichkov

This report has been written as part of the 3rd semester project for the Computer Sciences course at UCN, University College Nordjylland. The main objective of this project according to (University College Nordjylland, 2014, pp. 7-8) *is to "master more sophisticated elements in the computer science profession and realize distributed software systems; and contribute to the selection and use of technology in the context of system development and programming of distributed IT systems as well as give the students thorough knowledge of aspects of technology." and "make new and further developments and integration of distributed IT systems on a systematic basis using situational modern system development methods and techniques" within the timeframe that has been given by the institution.*

Preface

We would like to thank **Dimitrios Kondylis** for his continuing support during this project. The project was challenging, and we set the bar high right from the start. Dimitrios' support and feedback were invaluable to keep the project on track and ensure that we had a healthy view on the reality and achievability of this project.

We would also like to thank **Michael H. Andersen** for his support during the project, giving use some good pointers about what to look out for with our web application and the concurrency issue.

Lastly, we would also like to thank **Simon Schmidt-Jakobsen** for his help with overcoming some of the trickier obstacles we encountered while programming our project.

Thank you,

Stefan B., Linda, Alexander, Stefan J., Dimitar and Nikola.
17-12-2018

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Introduction

This report has been written as part of the Programming and Technology section of the 3rd Semester project for Computer Sciences AP Degree at UCN Aalborg. The goal for this project is to successfully create software in the C# language which demonstrates the student's ability to create client server architecture within a piece of software which also solves a concurrency problem. This report shows the steps taken to build working software from a programming and technology perspective, which also takes security into consideration (University College Nordjylland, 2014).

The rest of this introduction will detail the problem area and -statement in more detail, go into how data is collected and processed, will detail the structure of the rest of the report and finally some logistical information regarding version control and databases used. The team working on this project has agreed to sign a group contract. A copy of this can be found in Appendix 2. This copy is not signed, however by signing this report all group members have agreed to this contract.

Problem Area

In order to get a clear view of the problem, and in order to get some experience dealing with a (fictional) company and (fictional) external product owner, the problem area and problem statement (outlined in the next section) were written from the perspective of Best Design Group (*BDG*), who has been contacted by Worst Production Company (*WPC*) with a request for a retro game. '*WPC'* wants a demo of a game that shows a concept of zombies going rogue in a dinner show so that they can see how their customers would react to it.

'*WPC'* has requested '*BDG'* to create a web application to go with the game so that players could sign-up for a dinner show specifically with their friends. Furthermore, the company can only afford one server.

Due to the '*WPC*'s' limited resources, storage space and server architecture will have to be taken into consideration. As there would be multiple connections to the server, concurrency would have to be accounted for. The company wishes to use a website so that users will be able to download their game, register and reserve spaces for the game online.

This report will delve into the programming and technological decisions made to create software which will satisfy '*WPC's'* requirements and detail how this was done.

Problem Statement

Now that the problem area has been clearly defined it is vital to write a problem statement which covers all aspects of the project. The main question which needs to be answered is as follows:

"How does '*BDG'* create software which is built using C# (.NET framework) and incorporates a client-server architecture with WCF within five Scrum sprints?"

In order to answer this question, 3 additional questions need to be answered first, which go further into the details of the application itself. These questions are:

- "How is a seamless user experience using WPF for a dedicated client achieved?"
- "How is a seamless user experience using HTML, CSS and JavaScript for a web application created?"
- "How are possible concurrency issues which can occur in a multiple user environment solved?"

Please refer to Chapter 3 Conclusion and for a full problem statement; Appendix 3.

Empirical Data Collection

During the construction of this project, external resources, like books, electronic publications and websites were accessed to gather all information necessary to come to a successful fruition of the project. To ensure these all live up to the high-quality standard which is expected from a higher educational institute, only information which has been gathered through empirical research has been used.

Structure of the report

This report starts with an overview of the architectural choices in the project in Chapter 1, elaborating them with the decisions which were made and why. Following this some examples from the project, including a closer look at the controller layer, the concurrency problem and solution, and finally security within the project.

This is followed by Chapter 2 which goes into the technologies used and how they have been applied in the project. This is the largest part of the report and addresses all key aspects of the project, starting with C# and the .NET framework, followed by ADO.NET, WCF and WPF. Next is a segment on SignalR, ASP.NET and the use of Unity in this project.

Closing the report are chapter 3 and 4 which will first discuss the conclusion of the programming process and finally the evaluation which covers the project, as well as group members giving feedback to one another.

Finally, any relevant appendixes which will be referred to in various places the report.

Vision for the Project

To create great software solutions to better everyday life for everyone!

Logistical Information

Database: dinnergeddon.database.windows.net

GitHub repository: https://github.com/best-design-group/Dinnergeddon/commits/master

Version commit number: 390

1. Architecture

The architecture of a system is an abstract model that represents its functionality. It should include minimal design information to maintain a high level of abstraction. (Sommerville, 2016)

1.1. Client-Server Architecture

In client-server architecture the system functions are presented by a set of services, which are each delivered by a server. Users can access these services concurrently, via the Internet. This pattern was used because the purposes of the project required multi-user access from various locations and database storage. The big advantage of this architecture choice is the ability to distribute servers and maintain general functionality. The disadvantages are the services being susceptible to denial-of-service attacks and server failure, unpredictable performance and management issues if there is ownership by different organizations. (Sommerville, 2016)

1.1. Architecture in Dinnergeddon

The architecture of the Dinnergeddon project is simple compared to larger projects out there. The server part of the project has three layers – one that's responsible for accepting requests and giving answers to those requests (see also Chapter 2.3), one that's responsible for the business logic – the controller, and one that's responsible for the data access. The reason this separation exists is to ease the extension or change of the. For example – a change of the way requests should be accepted must be made. The changes that should be made on the project are only on the WCF service layer of the project. Having that separation, makes it very easy to test the whole server as no hard coupling exists.

However, the architecture changed a few times. Namely, during sprint 0, a decision that only a web service and a data layer would be needed. However, that proved to be difficult to work with, because it is not scalable and extensible enough. That's why, during the second sprint the business logic and the WCF service were separated to fulfil the requirements.

1.2. Controller layer and its concurrency issue

The main function of the controller layer is to hold business logic. As it was already mentioned, a separation of concerns is very vital to a project if it must be scalable and extensible. Therefore, in the Dinnergeddon project, all the business logic lies in the controller layer. Of course, there's a reference to the database layer, so that data can be passed around, however it's a reference to an interface. This allows for much easier change and extension of the underlying logic of the data access layer, as shown below in Figure 1.

```
private readonly IAccountRepo accountRepo = new AccountRepo(DbComponents.GetInstance());
```

Figure 1: Interface referencing

The controller is responsible for accounts, lobbies and high-scores. It can create, update, delete and read data while also keeping track of any errors that can occur during those operations. One of the harder tasks for the controller is manipulating lobbies for the game. Since the lobbies can be joined by players around the world and have a limited number of open slots, there can be a concurrency issue with this operation. Let's take an example: two players would like to join a lobby that can have a maximum of 4 players, while 3 players are already inside that lobby. If both players press the join button at the same time, it might happen that both join the lobby, since the check for whether the player would be able to join (to be more elaborate, if there's enough space for the new player to join the lobby) are done before the player joins the lobby. This way, it might happen, due to latency

issues for example, that both the players read that the lobby has three players and they can join. However, that shouldn't be the case. Therefore, in the Dinnergeddon project a locking functionality was introduced to mitigate this problem.

```
private bool JoinLobby(Account account, Lobby lobby)
{
    /* other functionality, removed for clarity */

    // Lock the lobby, prevent dirty reads
    lock (lobby)
    {
        // Check if the lobby can accept another account
        if (lobby.Players.Count >= lobby.Limit)
            return false;

        lobby.Players.Add(account);
    }
    return true;
}
```

Figure 2: Lock demonstration

As you can see in the code snippet above in Figure 2, the lock keyword is used. This locks the object that's referenced in the parenthesis and doesn't allow any other threads to access this object – neither reading, nor writing any data to it. This way, we ensure that the check for the number of players in the lobby is done only one at a time thus removing the concurrency issue that was present before.

1.3. Security and password hashing

Obviously, security is a very big deal nowadays in the technological world. It isn't a good idea to store plain text passwords on the database and this is the reason encryption exists. However, another problem also exists – implementing a custom algorithm for password encryption is difficult as all the security risks must be considered. Therefore, the project uses an already existing implementation that has been proven to be successful so that no data security leaks would be possible.

```
public static string HashPassword(string password)
{
    byte[] salt;
    byte[] buffer;

    if (password == null || password == "")
        throw new ArgumentNullException("Password cannot be null or empty");

    using (Rfc2898DeriveBytes bytes = new Rfc2898DeriveBytes(password, 0x10, 0x3e8))
    {
        salt = bytes.Salt;
        buffer = bytes.GetBytes(0x20);
    }
    byte[] dst = new byte[0x31];
    Buffer.BlockCopy(salt, 0, dst, 1, 0x10);
    Buffer.BlockCopy(buffer, 0, dst, 0x11, 0x20);

    return Convert.ToBase64String(dst);
}
```

Figure 3: Password hashing

As shown above in Figure 3, the Rfc2898DeriveBytes class is being used. It comes from the System.Security.Cryptography namespace implemented by Microsoft themselves, thus assuring that the algorithm is secure and trustworthy.

2. Technologies

This chapter will cover the different technologies used in the project one by one and then give an extensive explanation of how this is used in the project (with code snippets as examples) and what the pros and cons are for using this technology versus any other available technology.

2.1. C# .NET Framework

One of the requirements for the project was to build a system using C# language with the .NET Framework.

.NET is a general-purpose development platform which has key features that enable a wide range of scenarios across multiple platforms. They include support for multiple programming languages, asynchronous and concurrent programming models, native interoperability and others.

.NET implementations apply the Common Language Infrastructure (CLI), which specifies language-independent runtime and language interoperability that lets the user to choose any .NET language to build applications and services.

Microsoft is actively developing and supporting three .NET languages – C#, F# and Visual Basic (VB).

C# is a simple, but powerful object-oriented language that is type-safe. It maintains the expressiveness of C-style languages, thus allowing a smooth learning transition to anyone with knowledge of the C language and similar programming languages.

.NET provides automatic memory managed for programs by employing garbage collection, which helps ensure memory safety.

Delegates are another feature .NET offers. A delegate is represented by a method signature. Any other method that has the same signature can be assigned to the delegate and is executed whenever the delegate is invoked. To see how this project utilizes delegates, please refer to the implementation of the WPF client.

After realizing all these advantages, it became even more clear that implementing the project using .NET Framework and C# language is great idea.

2.2. ADO.NET

As one of the requirements is to have a database connection to the application, a Data Access Layer (DAL) needed to be implemented. For that scenario, there are a lot of options to choose from. One of them is the ADO.NET library provided by Microsoft.

One of the good features of ADO.NET is the fact that it's low level – as close to the database as possible. One must write their own SQL statements, create their own connections, chose when to use a transaction and when to lock that transaction to prevent changing the underlying data while the operation is active, et cetera. This allows for creation of a very robust, easily expandable and fast DAL.

However, there are also drawbacks to using a library so close to the database. One of the most important ones is the connection itself. The programmer creating the DAL must close the connection themselves, even when there's a bug in the system. If that's not done, the connection will be kept open thus polluting the connection pool of the database. Another downside to this approach is the data types. The database connection class doesn't return C# types, instead it returns rows of data which the programmer must bind to the correct datatypes themselves. This creates a couple of problems – awareness of which column represents which datatype is essential for the data access layer. The second, also as important problem, is when the project is being refactored. If there are any changes to the data types of the project, the same type of changes will have to be made on the database and vice versa.

On the other hand, a higher-level library like the Entity Framework, also created and maintained by Microsoft, is type safe. SQL statements don't have to be written to access the database and the connections do not have to be manually closed. However, a higher-level framework lacks customizability to the way the connection is made to the database.

After weighing the pros and cons for using each type of library, it was decided to stick with the low level ADO.NET library which will give them more control over what's going on in the database and keep good speed so that results will be returned to the clients as fast as possible.

The problems described above would have to be mitigated. The way that was chosen to do so is closing the database objects with the "using" statement. It captures an object and opens a block of code that would use that object. After the block of code has finished executing or an exception occurs, the object that was created at the top of the block is immediately disposed either by calling it's Dispose method (if that class implements the IDisposable interface) or by just removing the reference from memory. Since all database classes have implement the IDisposable interface, all of them will be disposed or closed.

```
public Account GetAccountByID(Guid ID)
{
    Account account = null;
    connection.Open();
    using (IDbCommand command = connection.CreateCommand())
    {
        command.CommandText = "select * from Accounts where id=@id";

        // Escape SQL injections
        IDbDataParameter param = command.CreateParameter();
        param.ParameterName = "@id";
        param.Value = ID;
        command.Parameters.Add(param);

        try
        {
            using (IDataReader reader = command.ExecuteReader())
            {
                // Check if we actually have a row from the DB, if not throw an exception
                if (!reader.Read())
                {
                    connection.Close();
                    throw new KeyNotFoundException("An account with this ID was not
found");
                }

                account = Build(reader);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
    connection.Close();
    return account;
}
```

Figure 4: SQL Injection Escape

The code snippet in Figure 4 above shows another important part of our DAL – escaping SQL injection characters. For that IDbDataParameters are used, as they prevent the execution of malicious SQL scripts by assigning their values to a parameter as opposed to executing them.

2.3. Windows Communication Foundation

"Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, one can send data as messages from one service endpoint to another. An endpoint can be a part of a continuously available service, or it can be a service hosted in an application. An endpoint can also be a client of a service that requests data from a service endpoint." (Microsoft, 2017).

WCF is useful because it allows the programmer to easily create a windows service without having to worry about how the data will be sent from the server and received by the client. What the programmer must do instead is to state which services should be exposed to the internet (or intranet) and then continue to program the normal way.

Furthermore, WCF allows for the programmer to create different endpoints. As explained above, endpoints are where the client communicates with the service and vice versa.

What different endpoints mean is that a programmer can define one endpoint to be only accessible on the internet, and another to only be available on the inside of the server. For example, a WCF service can be defined with two endpoints – an HTTP endpoint and a TCP endpoint. The default behaviour of TCP is that it's only accessible within a computer, so if a programmer defines a service to be accessible only through TCP, only applications within the same computer will be able to use that service. On the other hand, HTTP is accessible on the entire internet, which means that everyone who knows the URL to the WCF service can use that service.

However, the most important note to take from this is that a programmer can define more than one endpoint on the service. This allows them to create one endpoint that is to be used by clients (everyone on the world) and one that's to be used by processes on the same server. Figure 5 below depicts this.

```
<protocolMapping>
    <add binding="basicHttpsBinding" scheme="https" />
    <add binding="netTcpBinding" scheme="net.tcp"/>
</protocolMapping>
```

Figure 5: Protocol mapping for two endpoints

A perfect example for this is the Dinnergeddon project. It has two endpoints – an HTTP endpoint and a TCP one. The HTTP endpoint is being consumed by the Windows Presentation Foundation (WPF) client and the TCP endpoint is being consumed by the ASP.NET server, which is explained further below. This means that all users who download the WPF client can access the functionality that's exposed through the HTTP endpoint, while they can't access critical information that's only exposed by the TCP endpoint. On the other hand, the ASP.NET server can access both the TCP endpoint and the HTTP endpoint information and functionality. This essentially allows for an admin functionality without having to worry about normal users being able to access that admin functionality.

What's being exposed on the TCP endpoint is user account information. This way the ASP.NET server can consume that information and serve HTML files to the correct user and proper information.

On the HTTP endpoint, highscores are being exposed. Since highscores are something that users probably wouldn't mind sharing with others, exposing them through the HTTP endpoint makes sense.

2.4. Dedicated client and WPF

To fulfil the client-server architecture requirement a dedicated Windows client needed to be developed using the C# programming language. After a thorough research two candidates were found that seemed to be suitable to be used as a Graphical User Interface framework, WinForms and Windows Presentation Foundation. Generally, a GUI framework makes it possible to create an application with various GUI elements without having to re-create all these elements manually and handle all the user inputs. WinForms is an older technology, which means its tested thoroughly for years, extensive documentation can be found on the internet, but it could be a lot of work to design a custom look and feel in an application, since it's simply just a layer on top of the standard Windows controls. WPF on the other hand is a next-generation presentation system for building Windows client application. It offers more flexibility in the sense of customizing the look of the elements and makes advanced data binding easier. Since no significant disadvantages were found except the fact that WPF is still in evolving phase compared to WinForms, WPF was chosen to be used as the GUI framework for the application.

2.4.1. WPF

Windows Presentation Foundation (WPF) is a UI framework that creates desktop client applications (Warren, 2018). Previously known as Avalon, WPF was initially introduced as part of .NET Framework 3.0.

One of the main features of WPF is that it uses the Extensible Application Markup Language (XAML) to create the user interface for a .NET Framework application. In earlier GUI frameworks both the user interface and behaviour were created in the same language, while WPF allows to create visible UI elements in the XAML markup and implement behaviours in procedural languages such as C#. This separation offers multiple great benefits. One of them is the fact that the development becomes more efficient because designers can implement an application's appearance in parallel with developers who are implementing the application's behaviour. Loose coupling between the appearance and behaviour is another significant advantage.

Another great feature of the Windows Presentation Foundation is the data binding which provides a simple and consistent way to application to present and interact with data. (Wenzel, 2017)
When a binding is established and the data changes, updates on the UI elements are automatic and vice versa.

2.4.2. MVVM

The Model-View-ViewModel (MVVM) pattern helps to cleanly separate the business and presentation logic of an application from its user interface (UI). Maintaining a clean separation between application logic and the UI helps to address numerous development issues and can make an application easier to test, maintain, and evolve. (Britch, 2017)

Model

Model classes in the MVVM pattern hold application data, they can be referred to as the domain objects. Models are usually implemented as simple classes or structs, don't contain any behaviours or services that manipulate the information.

View

Views are responsible for defining the layout and appearance of the UI and displaying visual elements and controls on the screen. They are the only thing the end user really interacts with. Each view is defined in XAML, ideally with a limited code-behind that does not contain business logic. The view manages input which then manipulates properties of the model. If a control supports

commands, the control's Command property can be data-bound to an ICommand property on the view model. When the control's command is invoked, the code in the view model will be executed. (Britch, 2017)

View model

The view model introduces the separation between the model and the view. Model information can be transformed by the view model and then be displayed on a view.

The view model implements and exposes properties and commands to which the view can data bind to. It also manipulates the model and notifies the view of any state changes.

In order for the view model to participate in two-way data binding with the view, its properties must raise the PropertyChanged event. View models satisfy this requirement by implementing the INotifyPropertyChanged interface and raising the PropertyChanged event when a property is changed.

Conclusion

Since applying the MVVM pattern allows true separation between the model and the view, changing the view can easily be done without the model needing to be modified and vice-versa.

There are several advantages for using this pattern. One of them is maintainability. Due to the clean separation it’s possible to make changes without worrying about other parts of the system. That means remaining agile and moving to new releases quickly is easier.

Another great benefit is testability. Since with MVVM each piece of code is more granular it allows for easier unit testing.

There are, however, a few disadvantages as well. For simple user interfaces MVVM can be overkill. Due to the complex data debugging can be difficult, which is a drawback, too.

After comparing the pros and cons, MVVM pattern was chosen to be used for developing the Windows client.

2.4.3. Implementation using the MVVM pattern

Architecture overview

As stated above, the Model-View-ViewModel pattern has been used as the base pattern for developing the dedicated client.

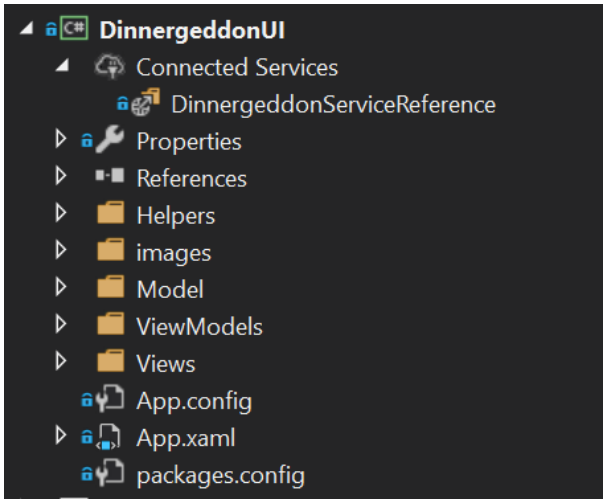


Figure 6: Dedicated client structure

Figure 6 shows the structure of the dedicated client project. To achieve easier navigation through the project, the files have been organized and moved into corresponding folders. The Model, View, ViewModels folders contain all the necessary classes required to implement the MVVM pattern.

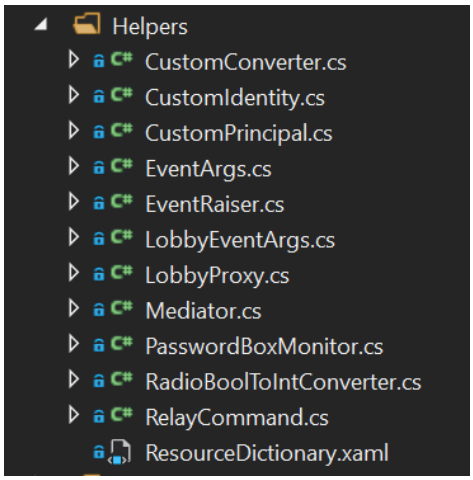


Figure 7: Helpers folder

The Helpers folder as shown in Figure 7 contains all files that were needed to be created in order to achieve various features such as user authentication, SignalR connection and the Mediator pattern.

To correctly implement the pattern, corresponding classes and files needed to be created and connected with each other.

The models

Since the WPF application is connected to a WFC service, the model classes exposed by the service are used as the MVVM model classes.

The views

The main window of the application is the MainWindow.xaml.

```
<DockPanel>
        <Border>
            <StackPanel>
            </StackPanel>
        </Border>
        <ContentControl Content="{Binding CurrentPageViewModel}" />
</DockPanel>
```

Figure 8: The layout of the MainWindow view

As Figure 8 demonstrates, its layout consists of a Border element and a ContentControl inside a DockPanel. The Border contains the menu buttons, that's why this element is visible throughout the whole application, while the ContentControl's Content is being bound to the CurrentPageViewModel property of the view model class associated with the MainWindow view.

The view models

All view model classes extend the *BaseViewModel* base class and implement the *IPageViewModel* interface.

```
class LobbiesViewModel : BaseViewModel, IPageViewModel
    {
```

Figure 9: An example view model class

Figure 9 shows, that the *BaseViewModel* itself implements the *INotifyPropertyChanged* interface which is used to notify client that a property value has changed. By extending the *BaseViewModel* class there's no need to implement the *INotifyPropertyChanged* interface for each view model class, which prevents the need to write duplicate code what would cause higher coupling.

Navigation through the application

Implementing a user-friendly navigation, which is visible in Figure 10, is always crucial when developing an application. As mentioned in the section about the views, each menu button is being bound to a command in the *MainMindowViewModel,* which then sets the corresponding view model to the CurrentPageViewModel property in the *MainWindowViewModel*.

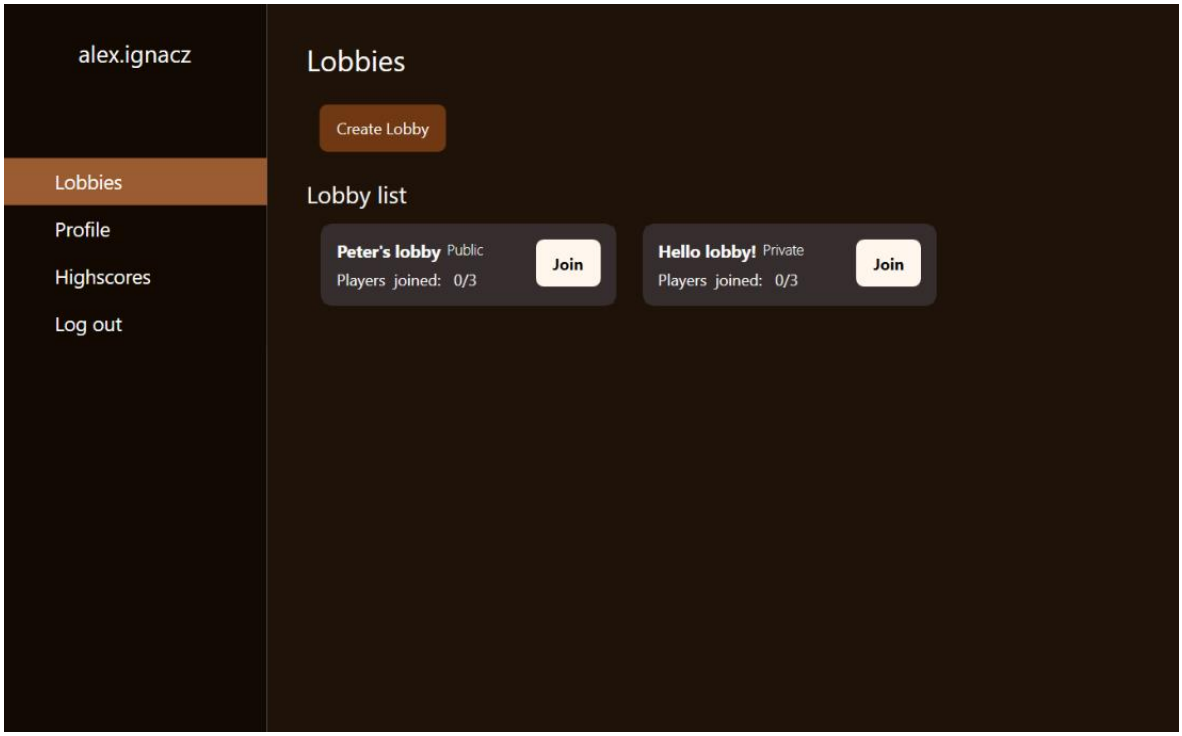


Figure 10: Side panel with menu buttons

Authentication

Since it is not possible to use the application without registering first, authentication needs to be a key part of the system. Keeping security in mind, after clicking the Log in button, the user's credentials are evaluated and verified on the WCF service, not on the client side.

To keep track of the currently logged in user, the System.Security.Principal namespace is used. For this, two custom classes had to be created. One, that implements the IPrincipal and another one that implements the IIdentity interface.

```
public class CustomIdentity : IIdentity
    {
        public CustomIdentity(Guid id, string userName, string email, string[] roles)
        {
            Id = id;
            Name = userName;
            Email = email;
            Roles = roles;
        }

        public string Name { get; private set; }
        public string Email { get; private set; }
        public string[] Roles { get; private set; }
        public Guid Id { get; private set; }

        public string AuthenticationType …
        public bool IsAuthenticated …
    }
```

Figure 11: Implementing the IIdentity interface

Figure 11 shows the implementation of the CustomIdentity class. The identity object contains information about the user being validated. The CustomIdentity class also has a Roles property

which will help to authorize different request made by users who have different roles. Currently the application does not have any admin specific operations but implementing it is a plan for the future.

The IPrincipal interface is implemented by the CustomPrincipal class. The interface defines a property for accessing an associated Identity object which makes it possible to get the currently logged in user throughout the whole application.

```
//Get the current principal object
CustomPrincipal customPrincipal = Thread.CurrentPrincipal as CustomPrincipal;

//Create a CustomIdentity object
customPrincipal.Identity = new CustomIdentity(account.Id, account.Username,
account.Email, new string[] { "" });
```

Figure 12: Accessing the principal object and creating the identity

As Figure 12 shows, after a successful authentication a new CustomIdentity object is created with the account's credentials, which then is accessible by simply referencing the CustomPrincipal object, see Figure 13.

```
CustomPrincipal customPrincipal = Thread.CurrentPrincipal as CustomPrincipal;

Guid userId = customPrincipal.Identity.Id;
```

Figure 13: Accessing the ID property of the currently logged in user

Implementing SignalR

As one of the main functionalities of the dedicated client is to display and allow to join lobbies, a huge concern was to always show the most up to date data without the users having to refresh the screen manually. To solve this issue, the SignalR library was used, which offers real-time functionalities, such as pushing content from the server-side to the connected clients.

All client-side logic for the SignalR was implemented in the LobbyProxy class, including the connection to the Hub, invoking methods from the Hub and setting up listeners the Hub can call.

```
public void CreateLobby(string lobbyName, int playerLimit, string password)
{
	hubProxy.Invoke("CreateLobby", new object[] { lobbyName, playerLimit, password
});
}
```

Figure 14: Invoking a method from the SignalR Hub

Figure 14 demonstrates an implementation of a method in the LobbyProxy, which invokes a method from the Hub.

After invoking the CreateLobby method from the Hub, it calls the lobbyCreated method on all the clients. Then it's the client's job to implement the proper handling of this call back method.

In the application events are used to handle method calls from the Hub, see Figure 15 for an example implementation.

```
public event EventHandler<LobbyEventArgs> LobbyCreated;

protected virtual void OnLobbyCreated(Lobby newLobby)
{
	if (LobbyCreated != null)
	LobbyCreated.Invoke(this, new LobbyEventArgs() { Lobby = newLobby });
}
```

Figure 15: Defining the events in the LobbyProxy class

If a class wants to be notified when a lobby was created, it just simply needs to subscribe to the LobbyCreated event and implement an event handler with a signature that matches the delegate signature for the event as Figure 16 and Figure 17 shows.

```
_proxy.LobbyCreated += OnLobbyCreated;
```

Figure 16: Subscribing to an event in a view model class

```
private void OnLobbyCreated(object sender, LobbyEventArgs args)
{

	DinnergeddonServiceReference.Lobby createdLobby = args.Lobby;
	App.Current.Dispatcher.Invoke((Action)delegate
	{
		_lobbies.Add(createdLobby);
	});
	Lobbies = _lobbies;
	OnPropertyChanged("Lobbies");

}
```

Figure 17: Example implementation of an event handler

Using SignalR with the described implementation makes it possible to develop a more user-friendly application, which was one of the main factors for the dedicated client.

2.5. SignalR

“SignalR is a library for C# developers that simplifies the process of adding real-time web functionality to their applications through the so-called web sockets.” (Microsoft, 2014). This way, the server doesn’t have to wait for the client to make a request so that it can send data. Instead it can send data to clients that are online and ready for accepting data from the server. It is exceptionally useful when creating chat applications for example, as clients are always sending messages to each other.

What SignalR is being used for in the Dinnergeddon project is to help the WPF client adapt to the changes of the lobbies. Since the lobbies can change at any time in any way, the SignalR server takes care of that – it sends messages with the new information to all online WPF clients and the clients themselves will handle those messages displaying the new relevant information.

Due to the lack of curriculum explaining how to implement the SignalR library to the WCF service, a decision was made to create a separate server that would handle all requests regarding lobbies.

The way the SignalR server works in the Dinnergeddon project is by using the so-called Hubs. Those hubs are responsible for a single part on the server. The hub exposes functionality on the web and all clients can connect to it. One of the most important methods in the hub is the CreateLobby method which you can see in Figure 18 below.

```
public void CreateLobby(string lobbyName, int playerLimit, string password)
{
    Lobby lobby = null;

    // If a password doeasn't exist, create a lobby without password protection
    if (password == string.Empty || password == null)
        lobby = lobbyController.CreateLobby(lobbyName, playerLimit);
    else
    // Otherwise, create one with password protection
        lobby = lobbyController.CreateLobby(lobbyName, playerLimit, password);

    // Notify all users of the newly created lobby
    Clients.All.lobbyCreated(lobby);
}
```

Figure 18: CreateLobby method

A very interesting part of how the SignalR process works lives in the dedicated client. A LobbyProxy class was created which wraps around the SignalR.Client functionality that's responsible for communicating with the SignalR server. The reason for a need of a wrapper is because the communication between the client and the server is not strongly typed which makes the development process of the WPF client very prone to type errors.

What the LobbyProxy class does is it instantiates a connection with the server and sets up listeners for messages sent by the server and creates strongly typed events that would fire every time there's a new message from the server. As shown below inFigure 19.

```
private void SetupListeners()
{
    hubProxy.On<Lobby>("lobbyCreated", (lobby) => OnLobbyCreated(lobby));
    hubProxy.On<Lobby>("lobbyUpdated", (lobby) => OnLobbyUpdated(lobby));
    hubProxy.On<Guid>("lobbyDeleted", (lobbyId) => OnLobbyDeleted(lobbyId));
}

// When a message from the server is received, this event is fired
public event EventHandler<LobbyEventArgs> LobbyCreated;

protected virtual void OnLobbyCreated(Lobby newLobby)
{
    if (LobbyCreated != null)
        LobbyCreated.Invoke(this, new LobbyEventArgs() { Lobby = newLobby });
}
```

Figure 19: Lobby listeners and events system

Another functionality that the LobbyProxy class has is to invoke methods that would just ask the server for an answer and not fire any events to the whole WPF application. This is shown below in Figure 20.

```
public Lobby GetLobbyById(Guid lobbyId)
{
    return hubProxy.Invoke<Lobby>("GetLobbyById", new object[] { lobbyId }).Result;
}
```

Figure 20: GetLobbyById method

2.6. ASP.NET

For developing the website ASP.NET was chosen because of vast selection of libraries and Microsoft's online documentation. Bootstrap library was also used for building a responsive website since it is included in ASP.NET by default. Main purpose of the website was to provide information about the game, allow users to download the game and create accounts.

2.6.1. MVC Pattern

The architectural pattern of the web application followed MVC.

MVC stands for Model-View-Controller:

- Model: Classes that represent the data of the application and that use validation logic to enforce business rules for that data.
- View: Template files that your application uses to dynamically generate HTML responses.
- Controller: Classes that handle incoming browser requests, retrieve model data, and then specify view templates that return a response to the browser.

(Anderson, Addie, & Levin, 2013)

The MVC design pattern was chosen because it decouples major components allowing for efficient code reuse and parallel development. Making it possible to work on front- and backend independently.

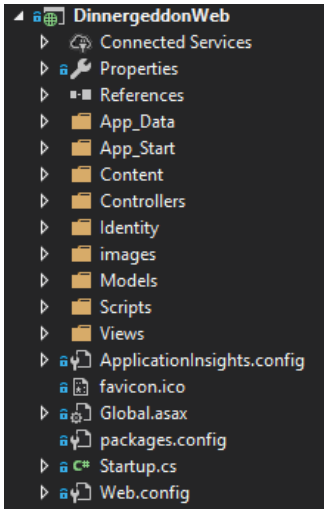


Figure 21: ASP.NET Web application following the MVC pattern

Figure 21 shows the structure for ASP.NET MVC web application.

How ASP.NET MVC works is demonstrated below through setting up a contact form for the website.

```
[HttpPost]
0 references | Alexander Ignácz, 8 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public ActionResult Contact(ContactViewModel vm)
{
    if (ModelState.IsValid)
    {
        try
        {
            string apiKey = ConfigurationManager.AppSettings["ApiKey"];
            var client = new SendGridClient(apiKey);
            var from = new EmailAddress(vm.Email);
            var subject = "contact form";
            var to = new EmailAddress(ConfigurationManager.AppSettings["Receiver"]);
            var message = vm.Message;

            SendGridMessage msg = MailHelper.CreateSingleEmail(from, to, subject, message, message);

            // Send the email.
            if (client != null)
            {
                client.SendEmailAsync(msg);
            }



            ModelState.Clear();
            ViewBag.Message = "Thank you for getting in touch! We will get back to you as soon as possible. ";
        }
        catch (Exception ex)
        {
            ModelState.Clear();
            ViewBag.Message = $" Sorry we are facing a problem here {ex.Message}";
        }
    }

    return View();
}
```

Figure 22: Controller Action method for a contact form

In Figure 22 the controller has a method “Contact” which is called when a form specified with the same name attribute is submitted.

1. The model state is checked for validation (satisfies Models conditions).
2. If it is valid then using the SendGrid API a message is sent.
3. Message text is assigned according to the result.
4. View is returned, and the message displayed to the user through the website. ViewBag is used for transferring temporary data from the controller to the view.

```
public class ContactViewModel
{
    [Required]
    [StringLength(20, MinimumLength = 4)]
    [Display(Name = "Name*")]
    0 references | Alexander Ignácz, 10 days ago | 1 author, 1 change | 0 exceptions
    public string Name { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email*")]
    2 references | Alexander Ignácz, 10 days ago | 1 author, 1 change | 0 exceptions
    public string Email { get; set; }

    [Required]
    [DataType(DataType.MultilineText)]
    [Display(Name = "Message*")]
    2 references | Alexander Ignácz, 10 days ago | 1 author, 1 change | 0 exceptions
    public string Message { get; set; }

}
```

Figure 23: Model for the contact form

In Figure 23 the Model part of MVC is portrayed. The model represents the data of the application. It also specifies which validation logic to enforce when user posts the form. If the requirements are not met, then the model state in the controller will return invalid.

```
using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="col-12">
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            <div class="col-sm-12">
                @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control", placeholder = Html.DisplayNameFor(model => model.Name) } })
                @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-12">
                @Html.EditorFor(model => model.Email, new { htmlAttributes = new { @class = "form-control", placeholder = Html.DisplayNameFor(model => model.Email) } })
                @Html.ValidationMessageFor(model => model.Email, "", new { @class = "text-danger" })
            </div>
        </div>
```

Figure 24: Part of the contact View class.

The View, as seen on Figure 24, is the final layer which will be displayed to the user. It is essentially HTML with C# code in it. Using ASP.NET HTML helpers a form is made according to the Model with an anti-forgery token to prevent Cross-Site Request Forgery. Finally, when the C# code has been ran a full html file is returned to the user and displayed as a web page.

2.6.2. User Accounts

As for everything surrounding user accounts (login, registration, admins), Microsoft's open source ASP.NET Identity library was used. The decision to use the library was made because it is recommended by Microsoft and fits with the project's needs. The library had to be modified though since ADO.NET was used for database access which was not the libraries default. Doing so helped to gain knowledge on how to work with open source software.

2.7. Unity

Unity was chosen due to its simplicity and abundance of learning resources, making it easier to use for development purposes. Because of the time constraints and lack of development experience in this field, it has proven to be the most efficient choice.

2.7.1. Game

Dinnergeddon is a co-operative multiplayer Space Invaders-like game where players work together to defend a front line and receive score based on the time they survive. This score is then displayed for everyone on a public leader board for players to compare and compete.

2.7.2. How we did it (also code snippets)

The game was created using assets (sprites, sounds, etc.) that were created or purchased for its development. Scripts for game behaviour were made using C# and Visual Studio. Examples of these scripts in Figure 25 & Figure 26 below.

```
public float speed;             //Floating point variable to store the player's movement speed.

private Rigidbody2D rb2d;       //Store a reference to the Rigidbody2D component required to use 2D Physics.

// Use this for initialization
0 references
void Start()
{
    //Get and store a reference to the Rigidbody2D component so that we can access it.
    rb2d = GetComponent<Rigidbody2D>();
}

//FixedUpdate is called at a fixed interval and is independent of frame rate. Put physics code here.
0 references
void FixedUpdate()
{
    if (!isLocalPlayer)
    {
        return;
    }

    //Store the current horizontal input in the float moveHorizontal.
    float moveHorizontal = Input.GetAxis("Horizontal");

    // we don't need vertical
    //Store the current vertical input in the float moveVertical.
    float moveVertical = 0; //Input.GetAxis("Vertical");

    //Use the two store floats to create a new Vector2 variable movement.
    Vector2 movement = new Vector2(moveHorizontal, moveVertical);

    //Call the AddForce function of our Rigidbody2D rb2d supplying movement multiplied by speed to move our player.
    rb2d.velocity = movement * speed;
}
```

Figure 25: Player movement script

```
//Instantiate a Random class in order to generate a random number.
Random random = new Random();
//Checks if the rigidbody of the model is colliding with a specific layer.
if (!_rigidbody2D.IsTouchingLayers(collisionObjectsLayer))
{
    //If the body is not colliding it continues to move down.
    //Creates a new vector with the target movement velocity, vector is three-dimensional in order to apply smoothing.
    Vector3 targetVelocity = new Vector2(0f, -movementSpeed * 0.10f);
    //The created vector is applied to the rigidbody along with smoothing.
    _rigidbody2D.velocity = Vector3.SmoothDamp(_rigidbody2D.velocity, targetVelocity, ref velocityForRef, movementSmoothing);
    Debug.Log(String.Format("{0} moving down.", gameObject.name));
    //Sets the "moving around objects" boolean to false.
    movingAround = false;
}

else if (_rigidbody2D.IsTouchingLayers(collisionObjectsLayer))
{
    //Checks to see if the rigidbody is already moving around an obstacle
    if (!movingAround)
    {
        //If not, a new vector is created, three dimensional in order to apply smoothing later.
        Vector3 targetVelocity;

        //A random direction between left and right is chosen from the previously instantiated Random class.
        //The if check equates the result to 0 in the range of 0 and 1.
        if (random.Next(0, 2) == 0)
        {
            //If the randomly generated number is 0, then the rigidbody's velocity is directed right.
            targetVelocity = new Vector2(movementSpeed * 0.10f, 0);
            //The velocity is applied with smoothing.
            _rigidbody2D.velocity = Vector3.SmoothDamp(_rigidbody2D.velocity, targetVelocity, ref velocityForRef, movementSmoothing);
            Debug.Log(String.Format("{0} moving right.", gameObject.name));
            //Moving around boolean is set to true to avoid repeated left right movement when colliding with an obstacle.
            movingAround = true;
            //Flip method to ensure body is facing correct direction.
            Flip(false);
        }
        else if (random.Next(0, 2) == 1)
        {
            //If the randomly generated number is 1, then the rigidbody's velocity is directed left.
            targetVelocity = new Vector2(-movementSpeed * 0.10f, 0);
            //The velocity is applied with smoothing.
            _rigidbody2D.velocity = Vector3.SmoothDamp(_rigidbody2D.velocity, targetVelocity, ref velocityForRef, movementSmoothing);
            Debug.Log(String.Format("{0} moving left.", gameObject.name));
            //Moving around boolean is set to true to avoid repeated left right movement when colliding with an obstacle.
            movingAround = true;
            //Flip method to ensure body is facing correct direction.
            Flip(true);
        }
```

Figure 26: Zombie Movement AI script

3. Conclusion

In order to draw an accurate conclusion, it is vital to first look back at the research question at the beginning of the ***report***, which is as follows;

"How does *'BDG'* create software which is built using C# (.NET framework) and incorporates a client-server architecture with WCF within five Scrum sprints?"

In order to answer this question, 3 additional questions need to be answered first, which go further into the details of the application itself. These questions are:

- "How is a seamless user experience using WPF for a dedicated client achieved?"
- "How is a seamless user experience using HTML, CSS and JavaScript for a web application created?"
- "How are possible concurrency issues which can occur in a multiple user environment solved?"

Discussion

To answer this, let's first have a recap of the project's process. The team started off by creating the database tables and the DAL for the project. This way the data could be manipulated around and used to write all the rest of the project. Since the team consists of six people, at the same time working on the ASP.NET server was also started. By the time the database layer was completed, the WCF project was started, however all the business logic lied in there, which wasn't ideal, so the team decided later to refactor it and split it into two – the controller layer and the WCF service.

While one part of the team was having some struggles with WCF, more requirements had to be fulfilled, so another part of the system was started – lobbies and a dedicated client. The WPF client was started and with it, creating the new parts of the controller for lobbies. While the controller was an easy task for the team, WPF turned out to be a much harder task to tackle.

When the controller layer was completed, the team's workforce was put into the game itself. New technologies had to be learned, so there was no time to waste. At that point the team split in two, due to the limitation of Unity only allowing 3 people to work on the project at the same time, and because the WPF project was still incomplete. By the end of the project, the WPF client was almost complete, the game had been finished and all the services were up and running.

Conclusion

One of the more complex functionalities that were implemented in the WPF client was the automated refreshing of lobbies. It proved to be rather difficult to implement due to time constraints, however it makes it much easier for a user to interact with the user interface. As the MVVM pattern was used as the core of the WPF client, it allows for an instant re-render whenever there's a change in the data structure on the code behind the user interface, which allows for a seamless user experience.

The website had to look appealing and responsive on both mobile and desktop devices. Thus, seamless user experience for the web application was achieved by using a library called Bootstrap which provided consistency and responsive design throughout the whole website for every device. CSS was used for styling the website to make it look good. JavaScript was used to further improve the responsiveness. Bootstrap makes it easier and more robust to build websites but at a cost. Trying to override any Bootstrap styles will most likely turn into a nightmare. The good outweighs the bad, when done right Bootstrap can simplify a lot of tedious work and save from headaches.

The concurrency issues within the project are solved by locking resource usage upon user interaction, in order to ensure only one user will have access to the given resource at a time and no conflicts will occur. A positive of this approach is that the first user that attempts access to the resource will most frequently result in a success, maintaining logical order and preventing the "cutting in front" moment. It is also a frequently-used solution to this issue and is therefore widely expected. However, based on internet connection and all the issues related to it, the exact opposite could be true where a user is overridden by someone with a better connection.

4. Evaluation

In the following chapter we are going to evaluate the project from a technological and programming perspective. It is going to briefly touch on the technologies and methodology used to complete it.

4.1. Project Evaluation

The project was as exciting as it was hard and challenging. Since the very beginning the collective decision was made that pair programming is to be adopted as main work style until completion. Pair programming can have its pros and cons. One of the pros is that two people are working on the same piece of code at the same time. This provides a combination of opinions on how to efficiently and swiftly complete a certain task. It also provides more than one person with knowledge of a certain area of the program in case a problem arises and the person that worked on the problematic code is absent, which could be very detrimental for the work process for an undefined amount of time. Another beneficial outcome of pair programming is that both people engaged in it can learn from each other, fill in informational gaps, learn better and more efficient methods of programming and broaden their knowledge. Not everything was smooth, however not all the blames is to be soaked by the pair-programming method. There were times at which people were idle, waiting for a task to be completed before they can get going with the next task, however this is partly due to having six people in the group and not being able to work on as many tasks.

One of the most challenging aspects had to do with the decision taken to work on a multiplayer game. This required each group member to do a lot of research into topics no one knew anything about, however this only presented the opportunity to learn even more. Having to work on a multiplayer game was very beneficial, as it had great need not only of coding but figuring out how to properly synchronize movement and sound across multiple clients.

4.2. Individual evaluations

This section will go into the individual evaluations of each group member. The feedback given below is feedback from the whole group to the individual person, starting off with a section for the group.

The group

Although working on a very specific technology (separating the tasks to the people that are comfortable with the specific tech) it may backfire on a bigger project, for example when one of the team members is sick, and this member is the only one that has worked on a specific part of the system, that part of the system is no longer being developed. Another thing is that this way, the knowledge of the group is being limited to only one part of the system – the one they're working on.

Pair programming was lacking throughout the project; however, team communication was on point regarding technologies used, contracts created.

Stefan Nikolaev Borisov

Stefan was good at cooperating in programming tasks and researching various methods of work for better efficiency.

<u>Possible points for improvement:</u>

He needs more practice, since his insecurity about his knowledge led him to be reluctant to take on greater responsibilities and initiatives.

Linda Augustina Carolus Fuchs

Linda was very organized and thorough when handling programming tasks. She kept her code clean and understandable. She was also eager to learn new concepts and technologies.

Possible points for improvement:

Linda lacked programming knowledge and that made her reluctant to take tasks on.

Alexander Ignácz

Alex was very proficient in user interface programming which helped the group's understanding of it much easier to grasp and improve.

Possible points for improvement:
Alex should format his code more often. There were a lot of commented out code and code that's not formatted in the correct way.

Stefan Jõemägi

Stefan had a lot of pre-existing knowledge about various technologies, which was very beneficial for the project. He also turned out to be a great partner for pair-programming.

Possible points for improvement:
Stefan can benefit a lot by communicating more with his teammates about problems, possible solutions. Was also not always attentive about his code formatting and cleanliness.

Dimitar Bogomilov Pilyakov

Dimitar's value consisted in taking on tasks that others would avoid or have very little determination in accomplishing. His healthy amount of scepticism helped him, and the group look for alternative methods for completing a solution.

Possible points for improvement:
Dimitar should have more communication with the team, especially about what he's currently working on. He should try to pair program more, as this would give him a lot knowledge about, but for the system that's being created at the time.

Nikola Anastasov Velichkov

Nikola has always been a very active and contributing member of the team. Very knowledgeable and eager to learn about new programming techniques and languages and happy to assist others in learning them as well. Also encouraging other team members to state their opinion. A valued member of the group.

Possible points for improvement:
His enthusiasm and helpfulness are very infectious but sometimes tends to keep him away from his own work and/ or distract others. A possible solution would be to take some extra breaks from time to time to ensure he keeps his enthusiasm and professional work a bit more separated, so he can have time for both without either one getting compromised.

Appendix 1. Literature list

Anderson, R., Addie, S., & Levin, I. (2013, October 17). *Microsoft ASP.NET MVC Getting Started*. Retrieved from Microsofts documentation: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-controller

Andrew Troelsen, P. J. (2017). *Pro C# 7 with .NET and .NET Core* (8 ed.). Minneapolis: Apress.

Britch, D. (2017, 7 8). *The Model-View-ViewModel Pattern*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

Carter, P. (2017, 5 22). *Tour of .NET*. Retrieved from Microsof Docs: https://docs.microsoft.com/en-us/dotnet/standard/tour

Microsoft. (2014, June 10). *Introduction to SignalR*. Retrieved October 2018, from Introduction to SignalR | Microsoft Docs: https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

Microsoft. (2017, March 30). *What Is Windows Communication Foundation*. Retrieved October 2018, from What Is Windows Communication Foundation | Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf

Sommerville, I. (2016). *Software Engineering* (10 ed.). Essex: Pearson.

University College Nordjylland. (2014, September). *Curriculum for the Academy Profession Degree Programme in Computer Science. National Section.* Retrieved November 5th, 2018, from UCN.dk: https://www.ucn.dk/

Warren, G. (2018, April 16). *Get started with WPF*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2017

Wenzel, M. (2017, 3 3). *Data Binding Overview*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview

Appendix 2. 3rd Semester Project Group Contract

Group 7 – Best Design Group

- Deliver a completed and functional product
- Learn to write clean code that adheres to proper code standards and naming conventions, making it easy to follow the three pillars of object-oriented programming and make changes to the program if needed.
- Write down a separate section for these code standards in the programming report, highlighting, among other things, naming in 'Camel Case', C# get and set standards, tab width, and code column size.
- Attractive and usable user interface for both website and the developed application that takes into consideration the 10 heuristics of proper GUI development
- Be present and participate in the stand-up sprint meetings. Be proactive and take initiative.
- Be responsible in group work, in case of absence inform the group and if possible, work from home. The group should be able to show understanding to personal situations.
- Pair programming (2 devs., 1 keyboard) and switching from day to day to learn to write C# and use the .NET framework together.
- Make decisions as a group and keep discussing the pros and cons until we can have a unanimous decision.
- BE ON TIME! this includes being back on time from lunch breaks during class.
- Have some team building exercises during the months.

Appendix 3. Problem Statement

BDG - Dinnergeddon	
Student names	Stefan Nikolaev Borisov Linda Augustina Carolus Fuchs Alexander Ignácz Stefan Jõemägi Dimitar Bogomilov Pilyakov Nikola Anastasov Velichkov
Title (initial)	Dinnergeddon
Subject	Best Design Group (BDG) has been contacted by Worst Production Company (WPC) with a request for a retro game. WPC wants a demo of a game that shows a concept of zombies going rogue in a dinner show so that they can see how their customers would react to it. WPC has requested BDG to create a web application to go with the game so that players could sign-up for a dinner show specifically with their friends. Furthermore, the company can only afford one server.
Problem / Problem area	Due to the company's limited resources, storage space and server architecture will have to be taken into consideration. As there would be multiple connections to the server, concurrency would have to be accounted for. The company wishes to use a website so that users will be able to download their game, register and reserve spaces for the game online.

Problem statement	• How does BDG create software which is built using C# (.NET framework) and incorporates a client-server architecture with WCF within five Scrum sprints?
	  ○ How is a seamless user experience using WPF for a dedicated client achieved?
	  ○ How is a seamless user experience using HTML, CSS and JavaScript for a web application created?
	  ○ How are possible concurrency issues which can occur in a multiple user environment solved?

Method / procedure	During the project, Scrum and XP will be used as the main development methodology. For the back-end, C#'s WCF service has been chosen as a means of connecting to the web application and the game itself. The web application is going to be developed using various web technologies consisting of HTML, CSS and JavaScript primarily. The user interface of the dedicated client will be developed using WPF. The database will be created using Microsoft SQL.

This page is intentionally left blank.