Shipwars Online

Semester Report

December 2015

University of Southern Denmark, 3rd semester, Software Engineering
Projectgroup: 4

Participants:

Klaus Reche Riisom	klrii14@student.sdu.dk	Exam Nr. 65376824
Martin Fabricius	mfabr14@student.sdk.dk	Exam Nr. 64754120
Martin Slusarczyk Hubel	mahub14@student.sdu.dk	Exam Nr. 411321
Niels Bonde Nielsen	nieni14@student.sdu.dk	Exam Nr. 409455
Patrick Blomberg Florczak	paflo14@student.sdu.dk	Exam Nr. 65091350
Rasmus Lassen	ralas14@student.sdu.dk	Exam Nr. 411187

Advisor:

Claudio Giovanni Mattera	cgim@mmmi.sdu.dk

Contents

List of Figures

i Abstract

The overall purpose of this project is to make a distributed system involving a database, a server and a client. In order to specify this purpose into an actual project, it was chosen to digitize the well-known board game Battleship and call it ShipWars Online in order to avoid copyright issues.
This was done by examining the core rules of the game and write a C# application connected to a database and a server, and apply the game-rules and graphical user-interface equivalent to the look of the board game.
The result was a C# application that, when executed, functioned as a digitized version of the board game with features to login and search for matches against other players, if they are logged in.

ii Preface

The following report is the documentation that describes how the software was developed and why the group made the choices it did. It has been written in connection to with the bachelor degree in software engineering on the University of Southern Denmark - 3. Semester. The project has been developed as a part of the course Design of software systems in a global context. The group would like to thank Claudio Giovanni Mattera for the help and advice giving during this project.

This report should be read chronologically to ensure the full understanding of the subject and the choices made by the group.

By signing below here, the group members confirm that they all have been active in and contributed to, the project.

_____ Date: 18/12/2015.

_____ Date: 18/12/2015.

_____ Date: 18/12/2015.

_____ Date: 18/12/2015.

_____ Date: 18/12/2015.

_____ Date: 18/12/2015.

iii Editorial

Chapter	Writer	Contributor	Review
Abstract	Rasmus		Klaus
Preface	Klaus		Martin H
01. Introduction	Patrick		Martin F
02. Background	Rasmus		Niels, Patrick
03. Requirements	Niels		Rasmus
04. Tools	Niels		Rasmus
05. Methods	Niels		Rasmus
06. Market Analysis	Common	Patrick	Martin F
07. Architecture	Klaus		Martin H
08. Design	Martin F		Patrick
09. Implementation	Martin H		Klaus
10. Test	Martin F	Martin H	Niels
11. Discussion	Patrick	Klaus, Martin F	Martin F, Martin H
12. Conclusion	Common	Everyone	Common

4 Project Introduction

This semesters project definition was more open than the last two semester projects. We only had to make a distributed system. So we were free choose a topic, as long as it included a client, a server and a database, which were all connected to make an application.

The project handbook actually gave a case where we could develop a system that would help newly started firms make their way into Chinas market for production. We decided that we wanted to make something else, so we started brainstorming about what each of us wanted to make, then we all came to the agreement of making a game. After that we brainstormed again, this time about what kind of game we wanted to make. It ended up with the classic board game Battleship that we would make into a digital version called Shipwars. The game fit the criteria perfectly for the requirements we needed. It could have a client that connects to a server that also have a connection to a database where user data is stored. The game would also fulfil our courses requirements on this semester project with things like encryption on user data and a market analysis for the cross culture management course. With this game we would eliminate the need for being together in the same room when playing, and instead open the possibility to play against people across the globe.
When we agreed upon the game and its topic, we had to write a project formulation, and decided to write about the background of board games.

4.1 Project Formulation

As board games are rooted in the physical world, board games are limited to the people who can play against each other locally. In order for physical board games to connect a larger player-base that can play anywhere anytime, the Internet may be necessary in order to make it available worldwide. This creates certain problems and here are the ones the group will focus on solving:

- How to digitize a board game?
- How to connect multiple people via the Internet?
- How to differentiate between the players?

5 Background

This section will touch upon the subjects of the varying popularity between board games and video games, and try to explain some of the main reasons why they are varying.

The concept of playing games has existed for a very long time, but its meaning has changed through time. Just a decade or two ago computers were not as normal to possess as they are now, and by playing games people usually referred to some game involving physical activity or playing with physical objects. Chess is a good example of this and it is probably one of the oldest games that are still being played today. Families would usually gather around the table at late evenings and have a good time together before bedtime.

Due to technological advancement, computers now have a fixed spot in the homes of almost any family living in modern societies, and the term games has reached a much broader meaning. Whereas people who grew up in the 60 - 70s would most likely play games in shape of physical board games like chess, play cards or something similar, todays generation has a much wider range of opportunities, such as computers and gaming consoles. And it seems to be the trend to play games through computers and consoles, because as an article from telegraph.co.uk states it with a survey:

> "The survey suggested that while 73 per cent of parents remembered regularly playing board games as children, only 44 per cent of the children polled said they do so now" [1]

The reason for the board games' gradually decreased popularity, besides the technological progress, may also be connected with the games themselves. As board games are games involving physical objects, their value of experience is thereby limited. In order to play board games the participants have to be physically present at the same place as the games take place in the physical world. Video games, on the other hand, takes place in the digital realm and thus they offer another level of experience. But the thing that makes video games truly special, and has been a core factor of video games success in recent years is the concept of online multiplayer.

An online multiplayer means you have the opportunity to play against other players from all around the world. This means that when people play games online on a multiplayer game, they very often play with or against people they have never met or seen before. This concept is often called the separation of space and time and it is one of the major by-products of technological advancement; you basically chat and interact with players, even if they are sitting on the other side of the planet, and the responses are instant. Since board games are physical, they are subject to break sooner or later or at least miss a piece of the game thus making the game impossible to play. Video games does not have this problem since they have no physical shape. All they take to play is an installation and they do not age the same way.

If board games, or just the games, are to survive in a time with video games gaining popularity, the games must be digitized so it will be possible to play on the devices most people use these days: The computer, console or smartphone.

Ultimately this means it is no longer a board game when it has been digitized. But the game is the same - it has the same rules. But the way it is played is just different.

[1] http://www.telegraph.co.uk/men/relationships/fatherhood/11696224/Card-games-and-board-games-are-dying-out-and-its-no-great-loss.html

6 Requirements

This chapter is used to explain the thoughts on the requirements that were made for the project. This covers analyzing what is involved in creating a game. The scope for the project will be covered and finally the use cases for the project will be introduced and explained.

6.1 Game Development Analysis

The following analysis covers games and gaming split into two parts. The first part covers the concept of games. The second part covers the issues with game development.

The concept of a game can be seen as a goal-oriented experience, where a group of various activities is played out by one or more players. These activities are to provide a form of enjoyment or pleasure through competition either with oneself or others. In order to keep the game consistent rules are imposed onto the game. In this way, a game employs interactivity and participation of a user, referred to as a player. The player is put into a scenario with rules and goals, where the scenario could be either a reference to or abstraction of a realistic situation. The player either gets a goal provided by the game, or the player creates their own goal. The goal must be completed to win the game. Getting into a position of inability or failure to complete the goal results in losing the game.

Developing a game can bring certain issues. One of those issues is presentation. If the player cannot effectively interact with or understand what they are seeing or reading, their experience may not lead to enjoyment. Another issue is interaction. Without the proper options to complete the required goals, the player may not be able to fulfill the goal. This can again lead to an undesirable experience. A third issue is efficiency. If the game is unplayable because of inefficiencies, or stability issues, the player may not even want to play the game.

6.2 Scope

The game will contain an online access functionality that utilizes a client-server structure connecting with a database, as the game will make use of user accounts for the client, it will be necessary to encrypt all user account information. The group choose to use .NET as the framework for developing the client-server structure, since C# is used for programming.
The game will be very limited in graphics and aesthetics, since the focus of the development is on the technical aspects. The group have therefore chosen to limit the game to 2D graphics with very simple turn-based functionality, and make our own version of the classic board game Battleship. The group decided to make the game multiplayer to fulfill the distributed system part of the project.

6.2.1 The Board Game

The board game Battleships is build up around a basic turn based game, with a grid size of 10x10. Before the game starts each player chooses the spot for each of their five ships. Each round a player chooses a coordinate to which he attacks from his side to the opponent side. If it is a hit, the opponent places a red pin in the boat that is a hit, and the same does the player who fired. If it is a miss, the firing player must place a white pin in the coordinate to show that the shot did not hit. Each player then have five ships in varying sizes up to five pins long. When a ship has been sunk the opponent must say what kind of ship that is down. When one of the players have lowered all the opponent's ships, they win.

6.2.2 The Computer Game

The computer game will incorporate the board games functionality, and make most of it automated. In the first round each player places their ships location on the boards grid. In the next rounds until a player has won, the players will switch turns when a tile has been chosen. During the current players turn, that player will be shown a larger version of the opponents grid with their ships locations hidden. The player will then choose a tile on the grid to reveal whether or not there is a ship. If there is a ship, the tile will be marked in red, if not the tile will be marked in gray. A player has won when all of the opponent's ships have been fully revealed.

6.3 Use Case Diagram

The use cases made for the project were assigned a category in order to differentiate between them. The categories were assigned after where the use cases functionality were most prevalent. UC means the code made for the client layer, US for server and lastly UD for database.

6.3.1 Actors

Player: Will be interacting with the system though the client.
Client: Software part that is responsible for the GUI and the connection to the Server.
Server: All the software logic will be located here
Database: Here the data will be stored.

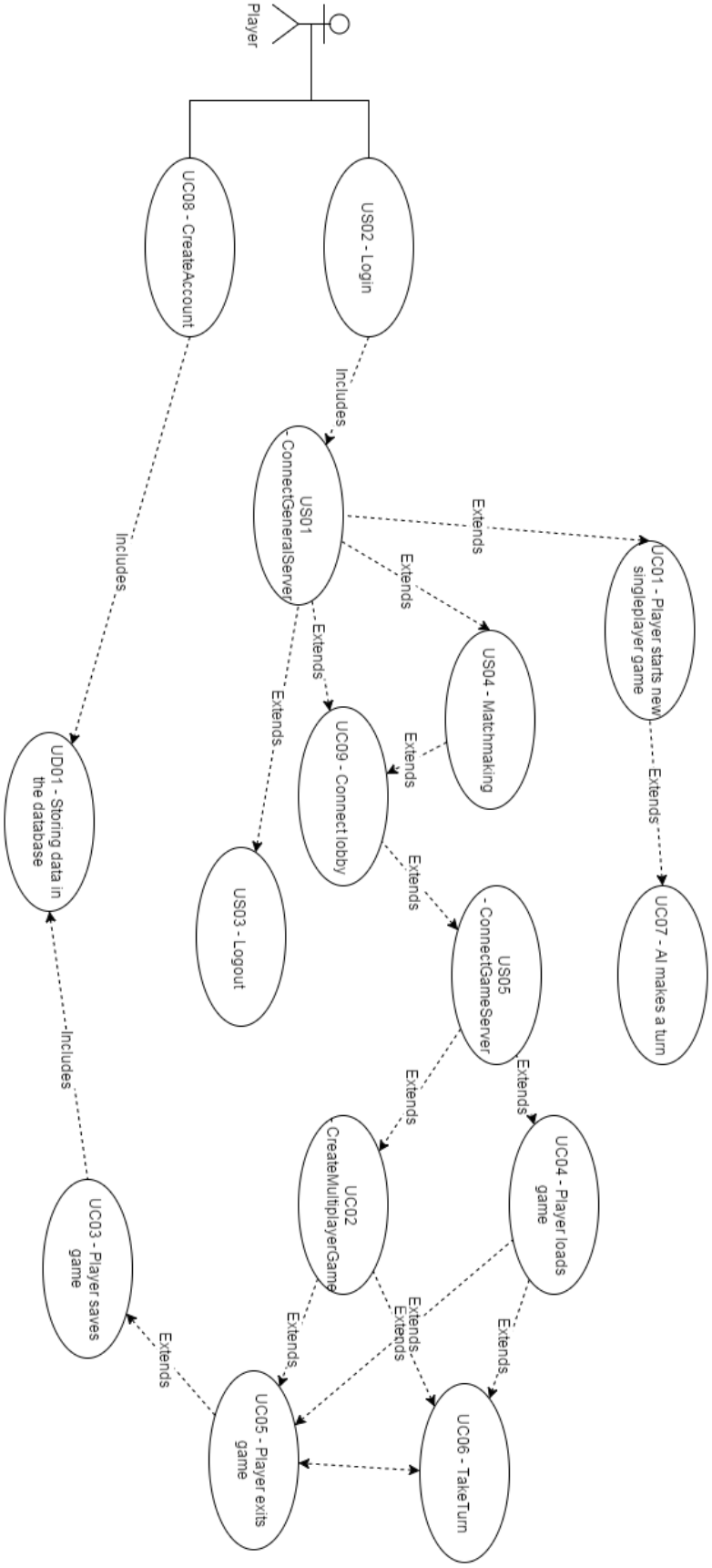


Figure 1: Usecase diagram

6.4 Use Case Priority

In order to prioritize our use cases, the group focused on three factors: First the amount we were going to learn from realizing the functionality involved. Second how important the use case is for fulfilling the requirement of the project. Lastly how important the functionality is for the final product, if some function is required in another use case as well. The more of these criteria, the use case fulfills, the higher it is prioritized.

US02 - Login	Must have
UD01 - Storing data in the database	Must have
US01 - ConnectGeneralServer	Must have
UC02 - CreateMultiplayerGame	Must have
UC06 - TakeTurn	Must have
US05 - ConnectGameServer	Must have
UC08 - CreateAccount	Should have
US03 - Logout	Should have
UC09 - Connect lobby	Should have
US04 - Matchmaking	Could have
UC05 - Player exits game	Would like
UC01 - Player starts new single player game	Would like
UC07 - AI makes a turn	Would like
UC03 - Player saves game	Would like
UC04 - Player loads game	Would like

Figure 2: Usecase priority

6.5 Use Case Description

The report will be focusing on 2 use cases to show an idea of the thoughts behind the process.
First one will be the creation of a game

Use case: UC09 - Connect lobby

Short Description: When a user log in as a player, they will then be put into a lobby with other players.

Primary actors: Player

Secondary actors: System

Stakeholders: The player will be able to easily see a list of other online players.

Overview: When a user wants to play, they login as a player. Then the player is put into a available lobby, in which the player can see other online players.

Precondition: The user has logged in.

Main course of events:

Actor: Player	System
1. A player wants to join a lobby.	
2. The player sends their token ID to verify their login status.	3. The system verifies the token ID.
	4. The system finds an appropriate lobby and adds the player.
	5. The system call back the player to notify a successful connection.

Post condition: The player is put into a lobby.

Alternative course of events:

Cross references: US02 - Login

Sketch of user interface:

Figure 3: Usecase description for UC09

Second one is when the game is started and the player makes a turn.

Use case: UC06 - TakeTurn

Short Description: Allows a player to take a turn in the game

Primary actors: Player

Secondary actors: Client

Stakeholders: Player

Overview: A player decides to take a turn, which allows the opponent to make their turn

Precondition: The player is connected to a game

Main course of events:

Actor: Player	Actor: Client	System
1. A player wants to take a turn	2. The client sends a request to take a turn for the player	3. The server takes the turn in the game
		4. The server sends a response back to both players
	5. The client gets the game data.	

Post condition: The players has taken a turn

Alternative course of events: None

Cross references: US05 - ConnectGameServer

Sketch of user interface: None

Figure 4: Usecase description for UC06

The remaning can be found in the appendix 1.4

7 Tools

This chapter will go through all the software used in the project.

7.1 Development tools

These are the software used directly for developing the software

7.1.1 Git

Git is a widely used version control system, used by project groups to collaborate as a team. When used it creates a repository, which makes a history over the changes to make it possible to rewind to an earlier stage. This was used as the version control system for the project.

7.1.2 Github

Github is a git server hosting service. It makes a server host a git repository available for a team to work on. Most of the hosted projects are made available to the public as an open source project for others to explore and give feedback on. This was used as the host service for our git repository.

7.1.3 Visual Studio

Visual Studio is one of the most used IDEs for programing a broad range of programming languages. It was used in this project to make all the C# code the final product consists of.

7.2 Project Management

These were the software used for managing the task present in the project.

7.2.1 Trello

Trello is a time management website used to create a virtual to-do list. When a board is created for the team all the members will be able to contribute to the board. It was used as the tool to manage our projects development cycles and divide the different use cases equally between the groups.

7.3 Modelling Tool

This where the tools used to create artifacts for the project.

7.3.1 Draw.io

Draw.io is an online diagram tool able to collaborate with different cloud services. The diagrams created here can be shared with the team, allowing multiple people to edit

7.4 Text Editor

This were the tool used for creating the documentation created during the project.

7.4.1 Latex

LaTeX is a document modeling language. It allows the user to design a document from scratch in plain text. When the document is done it compiles the plain text into the finished text file. This is the tool used to create the final report as it provides a high level of control over the fine details like layout, design and formatting.

7.4.2 Google Docs, Google Drive and Microsoft Word

These were used to make and share documents in the group during the development, in a less formal way. Google Docs was our primary text editor, where Word was used for assignments.

8 Methods

In this section the methods used in the project will be explained and why they were used.

8.1 Unified Process

We used UP as a guideline for the project as a whole, which means that Scrumban was our main structure for the project, and during the start of the project the inception phase of UP was used when we created the inception document.

8.2 Scrumban

This is an agile method for creating a product. It is a hybrid between Scrum and Kanban, utilizing the best of both methods. It combines the sprint iteration style from scrum with the bucket list and whiteboard planning style of Kanban. It works by putting all the requirements for the project onto sticky notes. This then becomes the backlog for work, and when a scrum sprint starts the requirement is evaluated by its importance. This helps to prioritize the task for a given sprint. There is also a limit on how many requirements that can be worked on at a given sprint, which will help to make sure that resources are not spend unnecessarily and the tasks get done.

9 Market Analysis

In the course CCM, the group had to make a market analysis which investigates the possibilities of selling the software on the global market. To find out if it is viable and how to go about it. A study on a fictional company, that the group made up, was done and also a study of a chosen markets culture was done for the use to potentially further development of our software.

9.1 Going international or not?

Reasons for going international:

- Bigger Market
  - Thanks to the fact that our product is developed in an international language, it is accessible for a broad audience
- Bigger Audience
  - A simple fact is that there are not many big game development companies in our little country. So if we want to become a part of a bigger company, we will need to get seen by as many as possible

Reason not to go international:

- Difference in cultures may cause conflicts/disagreements
  - Different countries may have different views on what a good game looks like. If we should choose to distribute to a country that dislikes our product, the country may give us a bad image based on their views
- Need for bigger hardware specs
  - If we distribute our game for a bigger market, the hardware specs will need to be higher as there will be more users and longer between the player
- Need for broader company staff across countries

If we would go international theres a high possibility the resources needed to maintain the company will exceed our current staffs limits. Therefore, we would have to hire staff in different countries, which can be both good and bad. The bad thing could be that it would make it more difficult or complicated to maintain an overview of the company and the progress altogether. And eventual conflicts will be difficult to appropriately handle.

9.2 SWOT Analysis

In here we look at our fictional companys goals and strengths, and how our product will fare on the market.

9.2.1 Mission and Goals

Our small game company, Small Dinghy Software, consist of six students and is focused and determined to make a game for the international market, and to make a profit from the sales of the game. We dont plan to make a living of the sales, only to make a profit. The focus will be on the western market, but will still be accessible on a global scale. This will be done using the Steam platform from Valve Corporation, which in turn will be a huge benefit for future customers and handling of taxation.

9.2.2 Eternal Environment

Legal protection and rights
Our game Shipwars Online, which is based on the classic board game Battleships, could not use that name, since it was licensed by the company who made the movie Battleships. This was the reason for naming the game Shipwars Online instead. We could in turn license the name Shipwars as a franchise, to ensure that our game will not be mistaken for another game with the same name.

Trade restrictions
Since we currently only plan to use the Steam platform as our sales platform, we will be limited to the users of Steam as our potential customers, which is not a bad thing, since Steam has a huge user/customer base of approximately 6 million people online on the service every day.

Figure 5: Steam chart of users playing

Shifting Production & Consumption
Competing software products emerging on Steam can change the balance of consumption and sale of Shipwars Online. Clones could emerge that try to take a share of the market. Sales on Steam can push more people to buy the product if wanted by the developer. In terms of production, it is basically nothing, as the product only exists as an intangible virtual product.

Strengths The strengths in our company is the game itself. With an internationally known game that has a similar style as the original board game Battleships, we hope that the memory of the board game will attract the players to our online game. The game will be bought for a one-time fee, and after be free for the user to play. Our company focuses on keeping the game interesting and fun for the gamers around the world. With the skilled employees, we aim to keep the game professional and up to date. The highest cost in our firm will be the server maintenance, that have to withstand the pressure of a high amount of players.

Weaknesses The marketing for our company is low budget. As we are a new player on the market, we need to keep cost down, but also distribute our game out to the public. Therefore, online advertisement on different sites are what we can afford for now. The competitions are the board game itself, and as it stands we have the upper hand, that is mostly because of the growing industry online, where the board games are beginning to lose market share. On the technical front we are not the most developed.

Opportunities The game is online and therefore it fits with the emerging trends of online games. Access to an international market allows a potential large player-base growth if the marketing and choice of platform is just right.

Threats Most of the threats to our product comes from the fact that our company is not big, so we dont have access to a big security network. An example would be that we dont have much claim on our game in terms of copyright and access to lawyers if someone violates it. Also we have to consider the copyrights of the company who first developed the Battleship board game and make sure we have an agreement with them before releasing our online version.

As the companys products focus on software the international market would be the ideal candidate. Software can be deployed across most of the world, as it can be distributed wirelessly and wired through the Internet. Software can be targeted to any specific country if needed, since the code is not set in stone.

9.3 Strategy for going international

Entering an overseas market in relation to Denmark requires looking into physical and virtual platforms situated in these locations. Physical platforms allow software to be resold through export to various brick-and-mortar businesses around the world. Virtual platforms allow software to be sold through either third-party licensing or through direct sales in a home-brewed platform.

In America, the virtual platform Steam is available as a software distributor, mostly focusing on games. Steam allows a software organisation to deploy and sell their software requiring no exclusivity. What Steam gets from this deal is a cut on the sale. Steam only applies to desktop (Windows, OSX, Linux), so other platforms would be needed in order to supplement other directions such as mobile apps (eg. App Store and Google Play Store).

The company chose the licensing strategy. This was done because it would let the company keep control and ownership of the product. It also opens up for the possibility that the company can expand the product to other international markets. Another reason to keep control of the game is so that the company can improve and expand the game.

Licensing also allows a small, if not non-existent, transport cost. Any transport cost would be with network bandwidth. However, with a third-party licensing platform, such as Steam, this cost is shifted onto the platform owner, instead of the developer.

The company chose the Steam platform because Steam offer a lot of help to small companies and individual developers who wish to release games but does not know the intricacies of licensing policy.

We have also tried to check if different cultures would react differently to our game. There was not much information on the subject, other than some countries can ban your game if they find it offensive. Because our game is a remake of the old board game, the likelihood of the game being banned is minimal. Also the age group for our customers, would most likely be in the age regime of nearly all ages. This is again because it is a board game, and can be played with everybody as soon you know the rules to the game, and it will be fairly simple to play.

10 Software Architecture

In this section the group will describe how it used software architecture in the project, the different choices the group made to ensure that the system runs according to the specifications, defined by the group, and how the work was planned, executed and then possibly changed so it would work better. It will also describe the structure of the system and how the system has been distributed.

10.1 Architecture

As told earlier in the report the group has chosen to make an online version of the game Battleship. With this in mind the group generated several requirements in the form of use cases, these use cases are listed in the Requirements chapter and in that section it is also shown how they were prioritized by the group. The requirements were split into groups based on what part of the system they dealt with, and prioritized using MoSCoW as explained in the section.

10.1.1 Client-Server

The main goal of the project was to make a distributed system. This the group accomplished by separating the system into three distinct components, the client, the server and the database. These three components run on different computers, the client and server are two standalone applications.

A distributed system is a collection of independent computers. Each computer has its own processor and memory. This allows each computer to work independently of each other. In order to utilise the resources of each computer, the computers are connected via various communication architectures. Examples of such a communication architecture include buses and wireless connections.[2]

One way to set up the computers is through a client-server architecture. A computer acts as a server, whilst other computers act as clients or other servers, such as a database server. The benefits of this setup are resource sharing, computation partitioning, connection-reliability and communication capabilities.[3]

Since the group decided to make a game, a client-server pattern was the obvious choice, especially when the system had to be distributed, the client was separated so it could run as a stand-alone application. The pattern also makes the server accessible from different places at once, and allows the player to play against each other online.

In relation to games, distributed systems allows multiple player clients to connect to a centralised server, working as an interface for all client requests. Examples of client requests include verifying account information, updating the game state, or communicating with other player clients.

One of the major problems with the client-server architecture is the strain on the server bandwidth. In MMORPGs such as World of Warcraft or EVE Online thousands of concurrent player clients connect to a server. This is possible by utilising multiple servers in their own distributed system. This allows the servers to distribute the computation load to multiple servers.[4]

[2] Silberschatz et al. 2010, p. 719
[3] Silberschatz et al. 2010, pp. 720-721
[4] Winn et al. 2013, pp. 3-4

10.1.2 Layer

The group chose a layered approach to ensure maintainability and to make it so that the game easily could be separated into its components. The group chose to make it a 3-layer architecture pattern where the game was split into 3 parts: (1) Presentation layer, (2) Business layer and (3) Data layer.

Presentation layer: : In this layer the interface is located. It is here that the group developed the GUI and made use of facades. This was to ensure maintainability since it means that the GUI or any other part is relatively easy to change out, as there is no direct connection between the layers. As long as the facades remain the same the system does not care how the GUI looks or how it works.

Business layer: Here is where all the logic of the game is located. This is where the game functionality lies and how it does it is defined. How the connection between the server and the clients worked will be explained in the Service-Oriented sub section.

Data layer: This is where the database is located, again accessible by a facade to make maintainability easy and separate concerns. This is a separate component in this system as the database is is not in the same place as the server or any of the clients. The database is where all the reusable information is stored. Information like user-information, usernames, e-mails, passwords etc. is stored in the database. The data is sent over a secure connection and the information is encrypted to make sure that none of the information given by the users can be stolen.

10.1.3 Service-Oriented

The server follows a service-oriented architecture (SOA), allowing client and server to communicate through a common interface. The service-orientation works by making interfaces that act as facades for logic. As explained earlier this is to ensure that the logic can be maintained easily on the server, and easily on the client by developers without server knowledge.

The advantages of SOA is distributing a reusable, loosely coupled and standardized service, allowing anonymous access from clients, where messages can be either synchronous or asynchronous. It is defined by a contract and a concrete implementation, where the client uses the contract to access the implementation.[5]

10.1.4 Security

In order for a user to be authorized and authenticated in the game, they first need to login. This is done using a single sign-on (SSO) architecture, where a token is created and provided to the client upon successful login.[6] The server tracks the session of the token and controls how it is used. The client now does not need to login again, but simply use the provided token. In this project, this token can be used to connect to the game service in order to start playing against others. This has the advantage of allowing a client to use any service that supports the token after logging in. The disadvantage, though, is that all services supporting the token need a connection to the service verifying the token.

[5] Vogel et al. 2011. pp. 206-207
[6] Vogel et al. 2011. p. 220

10.2 View Model

There are several different architectural view models, but the group decided to work with the 4+1 architectural view model, as this model revolves around the use of use cases, which is a main point of the groups Scrum/Kanban development approach. The model contains six views: (1) Use case view, (2) Logical view, (3) Implementation view, (4) Data view, (5) Process view, (6) Deployment view. Looking through the project, the group has primarily been working with the Logical and Implementation views but have used most of them to various degrees. Most of the project have been spent on the implementation of the system and during that time, the group mostly look at it from the Implementation view. During the planning of the project it was mostly looked at from the Logical view, though many of the requirements were defined as use cases it is clear that the Use case view was used. Moreover since the system is distributed it has also been looked at from the Process view.

Use case view:
This is the basis for the 4+1 view model, it is from this view that all the other views make their architectural decisions. To show all the use cases and how they interact with each other and the user, the group has made a use case diagram, this diagram has already been shown in the Requirements chapter.

Logical view:
This view looks at how the use cases will be implemented. It is used to get an overview of the the code is going to look, how the different classes will work together, and what methods and attributes the classes will contain. The group made a class diagram for each package in the project. They can be seen in the appendix 1.1

Data view:
This view is used to describe the data models and how they interact with the system. Since the group used ADO.NET to make the database, an entity model was created and from this, the database was automatically created. The entity model can be seen and is explained in the chapter Implementation.

Process view:
This view focuses on the behaviour of the system while it is running, and how the different use cases cause different classes to interact with each other and how. To show this the group developed some sequence diagrams, but after the first few sprints it was decided that they were not used enough to warrant the time spend on creating them. But those that have been made can be seen in appendix 1.7.

Deployment view:
This view describes how the static artefacts are deployed in the physical world. This the group has shown by making a deployment diagram, which shows how the system is separated into three parts, and how they connect to each other. Here is the deployment diagram shown:

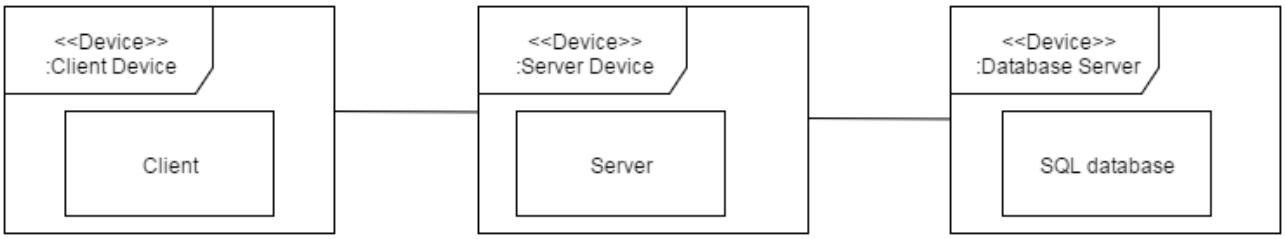


Figure 6: Deployment Diagram

11 Design

This section describes which object-oriented design approach we chose, what they are and which design patterns were used. The design for the Graphical User Interface (GUI) will also be covered.

11.1 Object-Oriented Design

We chose to use both SOLID and GRASP as a guideline for object-oriented design. The reason behind using both, was due to public opinion that using one or the other was not sufficient.[7][8] This way we could follow both guidelines where it was necessary, if SOLID did not help, GRASP just might and vice versa. Another reason was that the patterns of GRASP was used to assign responsibility to objects, whereas SOLID consist of programming principles that would help prevent bad code. SOLID and GRASP are described in the appendix 1.2 and 1.3.

11.2 Design patterns

In this subsection the most frequently used design patterns will be described. A design pattern is a reusable solution to a common problem.

11.2.1 Facades

Facade design patterns help to simplify the communication between multiple classes when used, by providing a collection of methods for common tasks, which also helps to make a complex system easier to understand. This pattern has been used to access the server, database, game mechanics and the GUI for all the windows. The way this design pattern was used, it closely aligns to the Controller pattern in GRASP.

Server
The server makes use of the DomainFacade in GeneralServices, whenever the client needs to access it. It only includes the necessary methods such as login, create account etc. Whereas a lot of the data is handled elsewhere, such as in the SecurityTokenService. This facade also includes the facade to the database, since the client does not need direct access to the database, but will instead go through this DomainFacade.

Database
The database uses DatabaseFacade in GeneralServices, and includes simple methods that holds the database code.

Game Mechanics
The lobby uses DomainFacade in GameServices, and includes only the methods the client would need to navigate the game once logged in, such as connect and disconnect to the lobby, start matchmaking and take turn placing coordinates. The code that happens in between and after, is handled elsewhere.

GUI
The GUI uses GUIFacade in the Client, which holds all the methods to display events on the screen, such as placing a coordinate or changing the page.

11.2.2 Singleton

The singleton design pattern ensures a class has only one instance. This pattern has been used on most of the facades to ensure that all access in the system uses the same instances. An exception is the DatabaseFacade, since the access happens through the DomainFacade, so only a singleton pattern on this facade is necessary. For instance, the DomainFacade in GameServices, uses a singleton pattern to ensure that the same lobby is used by all clients, and the GUIFacade uses it to ensure that only the clients own window is changed. The DomainFacade in GeneralServices uses it to ensure that the same SecurityTokenService access is used.

[7] http://forresterfootnotes.blogspot.dk/2013/03/solid-vs-grasp.html
[8] https://nikic.github.io/2011/12/27/Dont-be-STUPID-GRASP-SOLID.html

11.3 Graphical User Interface

We chose to go with an easy to use graphical user interface, and only with the tools provided for the WPF windows such as buttons, listViews, labels etc. To make the GUI easy to use, the labels will clear when the user wants to input their data by clicking the label, and the buttons will be named appropriately for the action it makes.

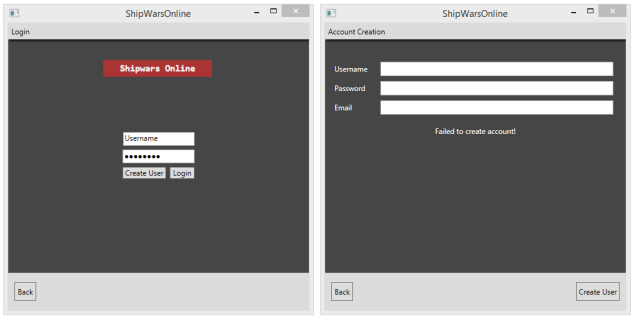


Figure 7: Shows the Login and Account creation page

Login page
The user will also be notified if the data is incorrect or missing, by displaying a label that describes the error when the button Login has been pressed. If the data is correct, the user will be taken to the lobby page. The button Back takes the user back to the previous page, and the button Create User takes the user to a page that allows them to create an account.

Account creation page
When the user clicks the Create User button, an account will be created and a label will be displayed saying that an account has successfully been created. If an error has occurred, the account will not have been created and a label describing the error will be displayed instead.

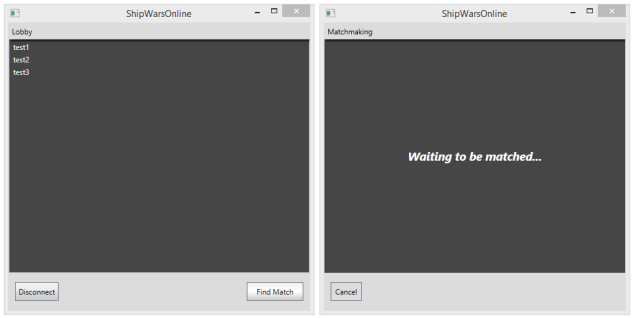


Figure 8: Shows the Login and Account creation page

Lobby page
From here the user becomes a player, and is displayed in a list with all the other players by their accounts usernames. This serves as a function to let the player know that other players are available, and whether or not a friend is online. The button Disconnect serves the same purpose as the back button, with the additional functionality of actually disconnecting the player from the game server. This means that the player will no longer be displayed in the lobby and they become a user again by being taken back to the Login page. The button Find Match takes the player to the matchmaking page.

Matchmaking page
This page has not been made, and the player is instead transferred directly to waiting page. The original plan was to make a matchmaking page, where the player has the option to either search for a random available player, or a specific player by their username. By clicking the Random Player button, the player would be taken to the waiting page. If the player clicked the Specific Player, the system would check if the entered username is available, if true then both players would be taken to the game, if not then a red label would be displayed specifying that the player doesnt exist or that the player is not online.

Since time has been a limit, this page was skipped, because the functionality of a message system would have been needed. The message system would be there to notify the specific player for a match request, this player could then choose to respond with a yes or no, instead of being forced to play.

Waiting page
This page is here to let the player know that the system is trying to find an opponent, even though the page doesnt have any functionality other than cancelling the action by clicking the Cancel button, which then takes the player back to the find match page.

Game page
When another player is available, the matchmaking system takes those two players and creates a game room, where the actual game is played. From here each player takes turns to place a tile on a coordinate, until someone wins. The green field is the player's own, and the red is the opponents field. The player would press the opponent's field to choose a coordinate, and then press the End Turn button. When a hit on a ship is made, it is displayed in orange, a miss is blue, and when the ship is sunk it becomes red.

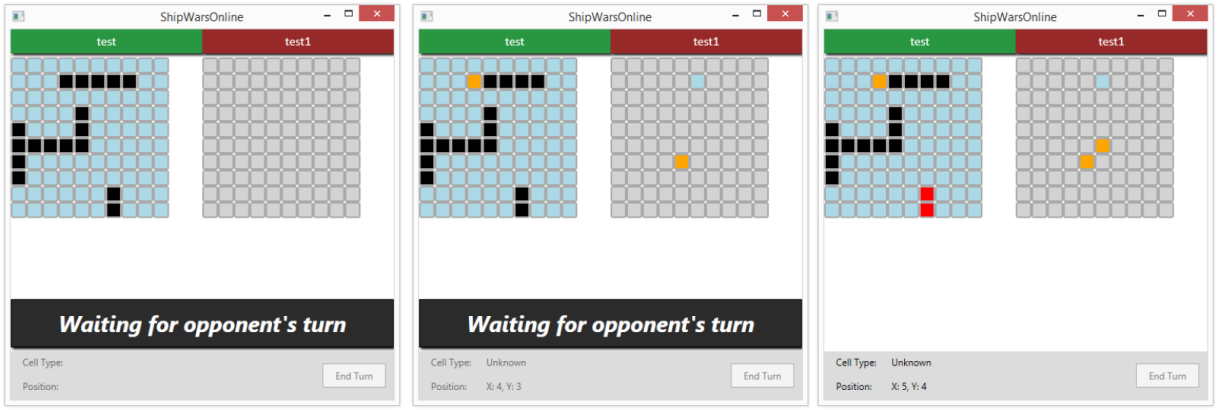


Figure 9: Shows the game GUI

12 Implementation

This section describes the technical implementation of the system. The topic of distribution, parallelism and data handling will be covered. Throughout the subsections, fragments of the implementation of two use cases, ConnectLobby and TakeTurn, will be provided as examples.

12.1 Overview

The overall project solution comprises of a client, server, database and game. The client contains the gui and connections to the server. The server hosts three services: (1) general service, (2) game service and (3) security token service. (1) contains services related to login and account creation. (2) contains the services related to lobby connection, matchmaking and taking turns. (3) include the services that allow verification and creation of tokens to be used with login in (1) and connection in (2).

The database itself lies externally to the implementation, but the SQL code exists in the solution created from an ADO.NET Entity Model. The game includes all the logic and data to create a game locally. This is used both for the server and the client to simulate a game. Client and server have each their wrapper classes to dictate how the local game is controlled and simulated.

12.2 Distribution

In this subsection the topic of distribution with focus on a client-server environment will be described.

12.2.1 Communication Protocol

The communication protocols between server and client are usually either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). TCP being a reliable connection-based protocol, whilst UDP is not.[9] This makes a big difference in terms of how the client communicates with the server. If a game client is supposed to communicate in real-time with the server, then UDP is preferred, as TCP could slow down the messages, or create lag over time. If real-time is not necessary, for instance in a simple turn-based game, then TCP is preferable, as it ensures messages arrive without loss and in the right order. If both are needed, then UDP should be chosen, as it can be customized to act as a TCP when necessary.

12.2.2 Connectivity

Depending on the requirements of the system, the way messages are handled across computers will be different. Some systems require multiple servers to distribute computation and management, whilst others require a simple TCP-connection to a single server.

12.2.3 Networking Technologies

In this project, the .NET framework was utilized to both create the client and server. The database was created externally. .NET offers several technologies to communicate between computers and processes, where a few are presented here: (1) Sockets, (2) Protocol listeners and clients, (3) Remoting and (4) Windows Communication Foundation (WCF).

Sockets allow manual control over inbound and outbound connections, as well as what gets sent and received. Sockets, however, operate with bytes, which would then require creating a message format to understand the message on both the client and server. The protocol listeners and clients act as a layer on top of sockets. They present a common interface for various protocols to work with, without having to work directly with sockets. For instance a TcpListener to create a TCP server.

With Remoting a remote proxy marshals calls to a MarshalByRefObject instance[10] [11]. This is reliant on both client and server having access to the .NET framework. WCF as a framework has the capabilities of all previous technologies. It uses a service-oriented architecture (SOA) where services

[9] Silberschatz et al. 2010. p. 739
[10] Microsoft 1
[11] Microsoft 2

send and receive based on requests. This seems similar to Remoting, but WCF allows interoperability and integration with industry standards, making it possible to distribute to other platforms and framework than .NET.[12] The flexibility of WCF makes it ideal as a choice for networking, which is why we have chosen this framework.

12.2.4 WCF Communication

In the use case ConnectLobby, the service contract for IGeneralService (see 10) is used to define the interface for communication between client and server. The attribute ServiceContract marks the interface as a contract, whilst the OperationContract attribute marks a specific entry point in the interface for a specific request. The FaultContract attribute allows the requests to return exceptions of the type FaultException if needed.

```
[ServiceContract]
public interface IGeneralService
{
    [OperationContract]
    [FaultContract(typeof(FaultException))]
    string Login(string username, string password);

    [OperationContract]
    [FaultContract(typeof(FaultException))]
    void Logout(string tokenId);

    [OperationContract]
    [FaultContract(typeof(FaultException))]
    void CreateAccount(string username, string password, string email);
}
```

Figure 10: IGeneralService service contract

While this settles the client to server messages and return values, it does not solve the issue of inter-client communication. Since this project is based on the client-server model, the client may not directly message each other. Therefore, the server needs to act as a liaison between the clients. The two main solutions for this would be a duplex service contract or creating client services. Creating a client service would allow easy access to all clients, but it would also allow any other person access to this service. Ensuring the integrity of the communication could prove difficult because of this, as the client service would have to verify the messages according to the servers services.

Instead of the client services, we chose the way of a duplex service contract, which allow two-way communication[13]. A duplex service contract requires a specified call-back interface, it is here implemented on the client. This allows the server to then send calls to the client on demand. The game service is a duplex service (see 11), allowing the server Lobby to send calls to players when certain events occur, like when a turn has been taken.

In order for the client to be able to consume the service, the service needs to be configured on both ends via the App.config file in each project. In the host (here server), a service is configured (see 12), whilst in the consumer (here client) a client is configured.

After configuration the client can create proxies to the services (see 13) based on the endpoint specified on the server. After creating a channel (proxy), the services can be accessed through the contract interface.

The solution also uses inter-service communication with the security token service. The security token service uses a net named pipe binding used for on-machine communication, which is used to create and verify tokens in the general service and game service.[14]

[12] Microsoft 3
[13] Microsoft 4
[14] Microsoft 5

```
[ServiceContract(CallbackContract = typeof(ICallback))]
public interface IGameService
{
    [OperationContract]
    [FaultContract(typeof(FaultException))]
    void ConnectLobby(string tokenID);

    [OperationContract]
    [FaultContract(typeof(FaultException))]
    void DisconnectLobby();

    [OperationContract]
    [FaultContract(typeof(FaultException))]
    List<string> GetLobby();
    ...
}
```

Figure 11: IGameService duplex service contract

```
<service name="GameServices.GameService">
    <endpoint address="net.tcp://localhost:8002/GameService"
        binding="netTcpBinding"
        bindingConfiguration=""
        name="GameServiceEndpoint"
        contract="GameServices.IGameService" />
</service>
```

Figure 12: Configuration of IGameService as service

12.2.5 Authentication and tracking

In order to keep track of the various connections to the game service, the IContextChannel - tied to a unique connection - was stored. This would then also be used to verify incoming requests to avoid requests that could change the internal state of the server game and lobby unexpectedly. The channel would also be needed in order to get the matching call-back connection to the client.

12.3 Parallelism

In this subsection the topic of parallelism with focus on threading and concurrency will be described.

12.3.1 Threading

The major concerns in terms of threading is avoiding freezing up the GUI by calling services, which may or may not have a considerable delay with its return. Therefore, all calls to the services are threaded using ThreadPool.QueueUserWorkItem (see 14), which will invoke a delegate in its own thread. This thread is managed in the background, removing it if all foreground threads terminate.[15]

When call-backs are made to the client, the client invokes events. These events are observed by GUIFacade. Here the GUIFacade is supposed to change the GUI based on the type of event. To ensure being able to access the UI thread in WPF, a Dispatcher is used. The Dispatcher is a service manager for providing work to a single thread. For every event the Dispatcher is invoked (see 15), which allows the events invocations to continue without waiting for the GUI update.[16]

12.3.2 Concurrency

As any number of clients could potentially connect to the services, the service is required to be thread-safe to ensure a stable internal state. WCF by default marks services as having a concurrency mode of Single, automatically making the service instances thread-safe. This is done by only allowing one call to enter at a time, each waiting to acquire the synchronization lock. If the calls are unable to get the lock, they will wait in an ordered queue.[17]

[15] Microsoft 6
[16] Microsoft 7
[17] Microsoft 8

```
var generalFactory = new ChannelFactory<IGeneralService>("GeneralServiceEndpoint");
generalService = generalFactory.CreateChannel();

var gameFactory = new DuplexChannelFactory<IGameService>(callback,
"GameServiceEndpoint");
gameService = gameFactory.CreateChannel();
```

Figure 13: Creating proxies for service access

```
public void LoginAndConnectLobby(string username, string password)
{
    ThreadPool.QueueUserWorkItem(delegate
    {
        try
        {
            serviceFacade.Login(username, password);
            serviceFacade.ConnectLobby();
        }
        catch (FaultException)
        {
            Console.WriteLine("Failed to login and connect to lobby!");
            OnPlayerFailedConnecting();
        }
    });
}
```

Figure 14: Queuing the ThreadPool

This heavily limits the service in terms of requests handled over time. For the current state of the project, this is not a problem, as only a few clients are connected, and they send out requests infrequently. If the project needed to be up scaled, however, the concurrency mode of the service would need to be set to Multiple, effectively removing the automatic thread-safety. The concurrently accessed data would then need to be synchronized manually.[18]

The last concurrency mode is Reentrant. It is a variant of the Single mode, also thread-safe, allowing call reentrancy. This means that a call is allowed to be called again in the same context used to enter the first time. This avoids potential deadlocks for something like call-backs, where a call-back method may call the same service call that caused the call-back method to be called.[19] This mode is set on the game duplex call-back in the client to avoid problems with calls reentering. Currently, though, the effects of this is not noticed, as the GUI threads the called events.

While the service itself is thread-safe, the resources it accesses are not by default. Although, in our implementation only one service accesses the resources supplied in its own project. Since only one call is allowed at a time per service, the resources inside should be safe. The only resource in the client that would need synchronization is the token ID string, but since strings are immutable it should already be thread-safe, as the internal state doesnt change.

12.4 Data Handling

In this subsection the topic of data handling with focus on serialized objects, data protection and the database will be described.

12.4.1 Serialized Objects

All primitive and most common types are automatically supported for serialization in WCF when they are used for return values or arguments[20]. In order to serialize custom data structures, however, a data contract must be defined, so that both the host (service) and consumer (client) understands the interface language.[21]

[18] Microsoft 8
[19] Microsoft 8
[20] Microsoft 9
[21] Microsoft 10

```
private void OnPlayerConnected()
{
    uiDispatcher.BeginInvoke(new Action(() =>
    {
        Console.WriteLine("Player connected to the game server!");

        if (currentWindowType != GUIWindowType.Login)
        {
            return;
        }

        WindowContainer.GetLoginControl().OnPlayerConnected();
        GotoWindow(GUIWindowType.Lobby);
    }));
}
```

Figure 15: Invoking the UI dispatcher

Data Transfer Object (DTO) The data contracts (see 16) act as data transfer objects, serving the purpose of sending data without behaviour or logic. They are defined by the DataContract attribute and DataMember attribute.

```
[DataContract]
public class GameCellImpactDTO
{
    [DataMember]
    public int AffectedPosX { get; set; }

    [DataMember]
    public int AffectedPosY { get; set; }

    [DataMember]
    public int PlayerIndex { get; set; }

    [DataMember]
    public CellType Type { get; set; }
}
```

Figure 16: DataContract for cell impact callbacks

12.4.2 Data Protection

Protecting data comes in various degrees, where some are more vulnerable than others if accessed by the wrong people or processes. In order to ensure secure transmission of data, two major areas need to be considered:

1. Communication channel
2. Data storage

Sensitive Data Sensitive data, such as credentials, send through the communication channel between client and server needs to be handled sensitively. Sending credentials as plain text will allow anyone listening to the channel to fetch the data. This would allow the listener to spoof or imitate a user, or in the worst case allow access to other sensitive user information.

Securing data transfer in WCF can be done by using transport security or message security. Transport security is ideal for intranets, whilst message security is ideal for internets. This is an area in which the group did not spent a lot of time in. This also means that credentials arent protected currently, but they would be a priority for future iterations.[22]

Using a technology such as SSL or TLS will provide message integrity and message confidentiality.

[22] Microsoft 11

Message integrity ensures that the messages are not modified while transferring. Message confidentiality ensures that the messages stay privately known while transferring.[23]

Tokens Once a secure channel has been created, the user can send their credentials to the target server. The login server then responds by providing the user with a token from the security token service. The server can use token-based authentication for all future requests for that specific user. This is done by creating a security token that is returned to the user. The user then sends this token to the server for initial requests to services using this login system to verify the user and lock the token in place.

Database The database was created using ADO.NET Entity Model, which can create a ready-to-go database based on the entity model. Using the entity model makes it easy to add new data in the database or extract information (See 17).

```
using (DataModelContainer db = new DataModelContainer())
{
    var accounts = from a in db.AccountSet
                   where a.Username.Equals(username)
                   && a.Password.Equals(passwordHash)
                   select a;

    if(accounts.Count() == 0)
    {
        return false;
    }
}
```

Figure 17: Using entity model container

As of now only user accounts are stored in the database (see 18), since multiplayer games need to be finished in one session. New accounts are created through the general service, contacting the entity model.

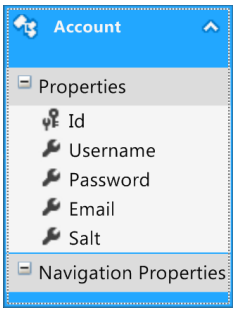


Figure 18: Account entity

Hashing and Salting Storing sensitive data as plain text in the database would allow anyone with access to the database to read and potentially exploit each entry. Therefore all passwords get hashed with a salt (see 19), where both the hashed password with salt and the salt itself is stored in the database. The username and email are not hashed for the database, as they are needed for quick look-up in the database, so that the salt can be retrieved to check the password. The username is however hashed with a salt for the token that needs to be returned.

[23] Microsoft 12

```
SHA256 sha256Encryption = SHA256.Create();
string saltString = databaseFacade.GetSalt(username);

string usernameHash = SecurityTokenManager.GetHashedString(sha256Encryption, username
+ saltString);
string passwordHash = SecurityTokenManager.GetHashedString(sha256Encryption, password
+ saltString);

if (!Verify(username, passwordHash))
{
    throw new ArgumentException("Could not login!");
}

return tokenService.GenerateToken(username,usernameHash, passwordHash);
```

Figure 19: Hashed salt of password

13 Test

In this section all test done on the project will be described. Due to time constraints, we were not able to complete any real testing. Instead we have described how we plan to test the elements of the software.

13.1 Test - Code

Normally we should have done unit testing to make sure our code is working and its behavior is expected, but because we learned about Test-driven development very late in the semester, we did not have the time to implement it. Instead we relied on software testing by running our code, then checking if the behavior is what we expected when navigating the GUI. This is not optimal, since with unit testing we can keep track of which parts have been tested. Also once a test is written it is much faster to run the test again, than having to run the code each time to check if the problem was fixed. This would have saved us a lot of resources in the projects, so if not because of the late introduction to unit testing, we might had saved some time.

Another way to test the code would be to crudely follow the sequence of events for the use cases, and verify if the code matches the expected behaviour, as well as pre- and post conditions.

13.2 Test - Server

Since the game is a multiplayer, we need to make sure that our server can hold that many clients accessing the server. We would test our servers load and performance by stress testing it. This is done by creating a lot of virtual users creating an account, logging in, searching for matches and makes moves in the games. Simultaneously we would create virtual users that log in and out, and have others that just stay in the lobby.

This test will simulate how the game will be used by users, and the server should therefore be able to handle such load, or else we need to optimize our server performance.

13.3 Test - Game

The purpose for this test is more for the functionality and general feel for the game, and should be tested with real users doing user acceptance testing.

We could go about this in different ways:

1. Getting different focus groups to play the game and after that interview them.
2. Publish the game prematurely to a fixed amount of users, then get as much feedback as possible. Also known as alpha/beta testing.
3. Publish the game prematurely to all users, while still developing the game. Feedback will be collected from users that voluntarily chooses to do so. Also known as Early access and is a form of alpha/beta testing.

4. A combination of test 1 and 2. First do test 1, then test 2 if the feedback was great.

Another test of the game is to test the game for bugs, which could be done together with test case 1 and 2 stated above, or just during development since our game is quite small. It would still be possible with 3, but is not desired since the general public will play it, and the game should therefore be kept to a version that is stable. Test 4 could be desired if we were developing a major game, but since not, we would probably go with test 1 as our test case.

14 Discussion

In this chapter we will briefly go over our accomplishments in the project period and what we think could have been done differently. There will also be a short description on how we used the courses of the semester in the project.

We started out with the idea of making an online multiplayer turn based game which had the functionality of battling against each other. Through the semester we have expanded the project with the knowledge we learned from our courses.

14.1 Did we reach our personal goals?

We did manage to develop a simple online game where multiple people can play against each other. We would have liked to get more graphical in our game, but this was not in the requirements so we made a simple GUI instead of going after an advanced one, so we could get more of our use cases implemented.

One of the major things we wanted to learn while making this game, was how to use the programming language C#, and how it was different from Java. We all learned to varying degrees to code in C#. This is a great benefit for our future, as it is a language that is often used and therefore very a useful skill to have.

14.2 Did we reach the goal for our semester?

In terms of distribution and networking, we successfully implemented a distributed product, allowing connections between client and server, and server and database. We tried our best to integrate the different courses into our project. Our plan was to implement encryption between the client and server, but due to time constraints we couldnt. So now we only have a token system that verifies the user connecting to the server, but the password is send as a plain text file over the network, instead of being encrypted. We integrated CCM in the form of our market analysis. OPN was integrated as our code, and was fulfilled since we managed to create a distributed system. Finally DES was integrated as a guideline for our architecture where we split the code up into three main layers: Client, Server and Database. It made more sense to the group and made it so we had a good view of the code.

14.3 Did we follow SOLID and GRASP?

We did not quite follow the guidelines from SOLID and some of the patterns from GRASP, which we should change if we were to continue working on the game. The reason we lost sight of using them, might be because most of us had not used C#, WCF, or WPF before. So our focus was instead to try and implement a working distributed system.

14.4 Could we sell the game as it is now?

If we want to sell the game ourselves then no, but we would be able to sell the game if we fixed the things mentioned above. It is hard to say if we could sell the game on the international market, since our market analysis does not go into great detail how the international market would respond to such a game. If we were to go on the international market, it would be wise to make a new analysis that looks into how the market would respond and whether or not it would be viable for us to do so. This would

imply that we would have to maintain the server and the cost would have to be covered by the sale of the game. However, as our market analysis shows if we use Steam we could reach an international audience, but we do not know if it would sell.

14.5 Project Problems

During this project there has been some issues that the group have had to deal with. One was that the group chose to write the code in C#, which is a programming language that most of the group had not worked with before. The reason the group chose to write in C# was to expand the skillset of the group, but it caused the development of code to be slow since the group had to learn the differences between Java and C#. One of these differences was that the group was taught how to network in Java and that it would be perfect for client-server interaction. Because of that the group tried to find C# equivalents, which WCF was, culminating in the use of this in the project.

The distribution of the system is not currently across multiple machines but rather only locally. This is due to several issues with compatibility of certain technologies and time involved. The provided Linux server can not directly use .NET without some intermediary. The mono framework can be used to replace .NET, but we did not have time to get this installed, as we were still trying to figure out .NET. In terms of the database, the provided one used PostgresSQL, which requires additional setup in Visual Studio in order to get to work with the ADO.NET entity model framework. We did initially work with a PostgresSQL library to create a connection to the database and work directly with SQL queries, but it quickly became time-consuming. In the end we settled with keeping it locally, but still open for change in terms of connectivity.

14.6 Project successes

On the other side there were also some things that the group did very well. In this project the group have worked very well together without any problems in the group, and everyone have done their share. Also the use of Kanban made the development of the software very structured and easy to keep an overview on.

If there was anything the group should have done differently it might be to make sure the entire group had studied more on the new programming language. There were a few in the group that had worked with it before, but the rest had never touched it. If those that had never touched it had studied it more diligently then maybe there would not have been the problem of slow code development. Otherwise we should have written either server or client in java, and the other in C#.

15 Conclusion

The goal of this project was to create a distributed software system. This culminated in a digitized board game, which was done through an analysis of an existing physical board game Battleships. The analysis allowed the group to see the essential components of the game required in order to implement a digital version.

The distribution of the software allowed a client and server to communicate in order to exchange requests and messages. This allowed two clients to play a networked multiplayer game of Shipwars Online through the server. The game can be played from start to finish, where a winner is declared in the end. Multiple multiplayer games can exist at any time allowing many pairs to play concurrently.

Connecting to the general services allowed a user to login, returning a token to be used for the game services. Connecting to game services with the token verified the user in the system, allowing the game services to store the user connection in a session instance. This way users could be identified through their session, making communication both ways possible.

The overall conclusion for the project is that though the group had some problems with the new programming language and frameworks (WCF and WPF) used, it managed to answer the questions that was stated at the beginning of the project. And the group made a prototype that shows how it decided to make a distributed software system.

16 Bibliography

16.1 Printed Works

- Silberschatz, A., & Galvin, P.B., & Gagne, G. (2010). Operating System Concepts with Java: John Wiley & Sons. Inc.
- Winn, J., & Bright, T., & Jensen, C.G., & Teng, C. (2013). Utilising game server technology on fully distributed architectures for collaborative multi-user CAx applications: Taylor & Francis Group.
- Vogel, O., & Arnold, I., & Chughtai, A., & Kehrer, T. (2011). Software Architecture: Springer

16.2 Online References

- Microsoft 1. Remotable and Nonremotable Objects. Retrieved 16/12/2015, from https://msdn.microsoft.com/librar
- Microsoft 2. .NET Framework Remoting Architecture. Retrieved 16/12/2015, from https://msdn.microsoft.com/en-us/library/2e7z38xb.aspx
- Microsoft 3. What Is Windows Communication Foundation. Retrieved 16/12/2015, from https://msdn.microsoft.co us/library/ms731082.aspx
- Microsoft 4. How to: Create a Duplex Contract. Retrieved 16/12/2015, from https://msdn.microsoft.com/en-us/library/ms731184(v=vs.110).aspx
- Microsoft 5. NetNamedPipeBinding Class. Retrieved 16/12/2015, https://msdn.microsoft.com/en-us/library/system.servicemodel.netnamedpipebinding.aspx
- Microsoft 6. ThreadPool Class. Retrieved 16/12/2015, https://msdn.microsoft.com/en-us/library/system.threading
- Microsoft 7. Dispatcher Class. Retrieved 16/12/2015, https://msdn.microsoft.com/en-us/library/system.windows.t
- Microsoft 8. Concurrency Management. Retrieved 16/12/2015, https://msdn.microsoft.com/en-us/library/orm-9780596521301-02-08.aspx
- Microsoft 9. Types Supported by the Data Contract Serializer. Retrieved 16/12/2015, https://msdn.microsoft.com/e us/library/ms731923.aspx
- Microsoft 10. Using Data Contracts. Retrieved 16/12/2015, https://msdn.microsoft.com/en-us/library/ms733127.aspx
- Microsoft 11. Chapter 7: Message and Transport Security. Retrieved 16/12/2015, https://msdn.microsoft.com/en-us/library/ff648863.aspx
- Microsoft 12. Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication. Retrieved 16/12/2015, https://msdn.microsoft.com/en-us/library/ff649205.aspx
- srp.pdf, Retrieved 02/2002, Robert C. Martin: http://www.objectmentor.com/resources/articles/srp.pdf
- ocp.pdf, 01/1996, Robert C. Martin: http://www.objectmentor.com/resources/articles/ocp.pdf
- Principles_and_Patterns.pdf, 01/2000, Robert C. Martin: http://www.objectmentor.com/resources/articles/Princip

1 Appendix

1.1 Class Diagrams

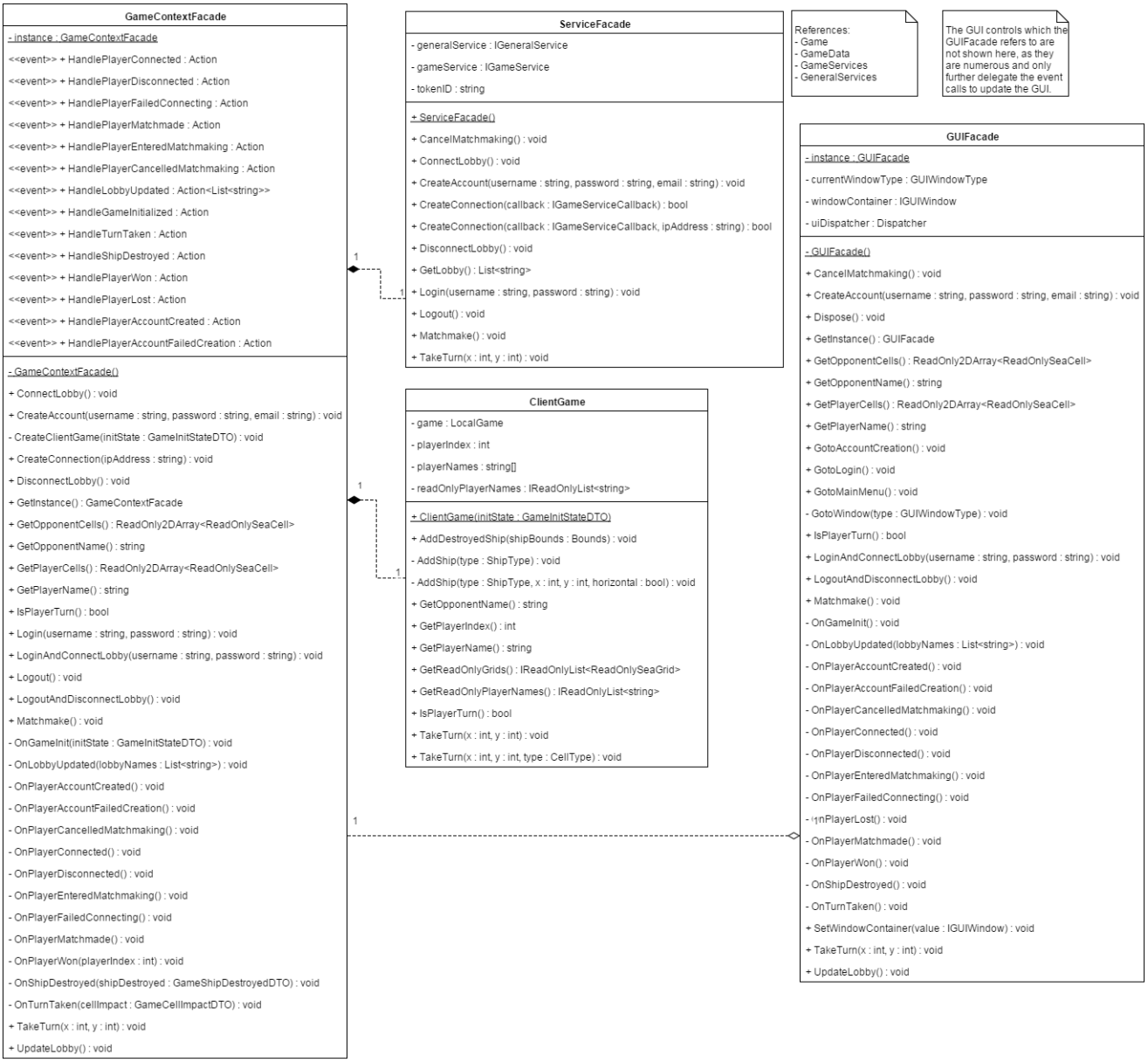


Figure 20: Client diagram

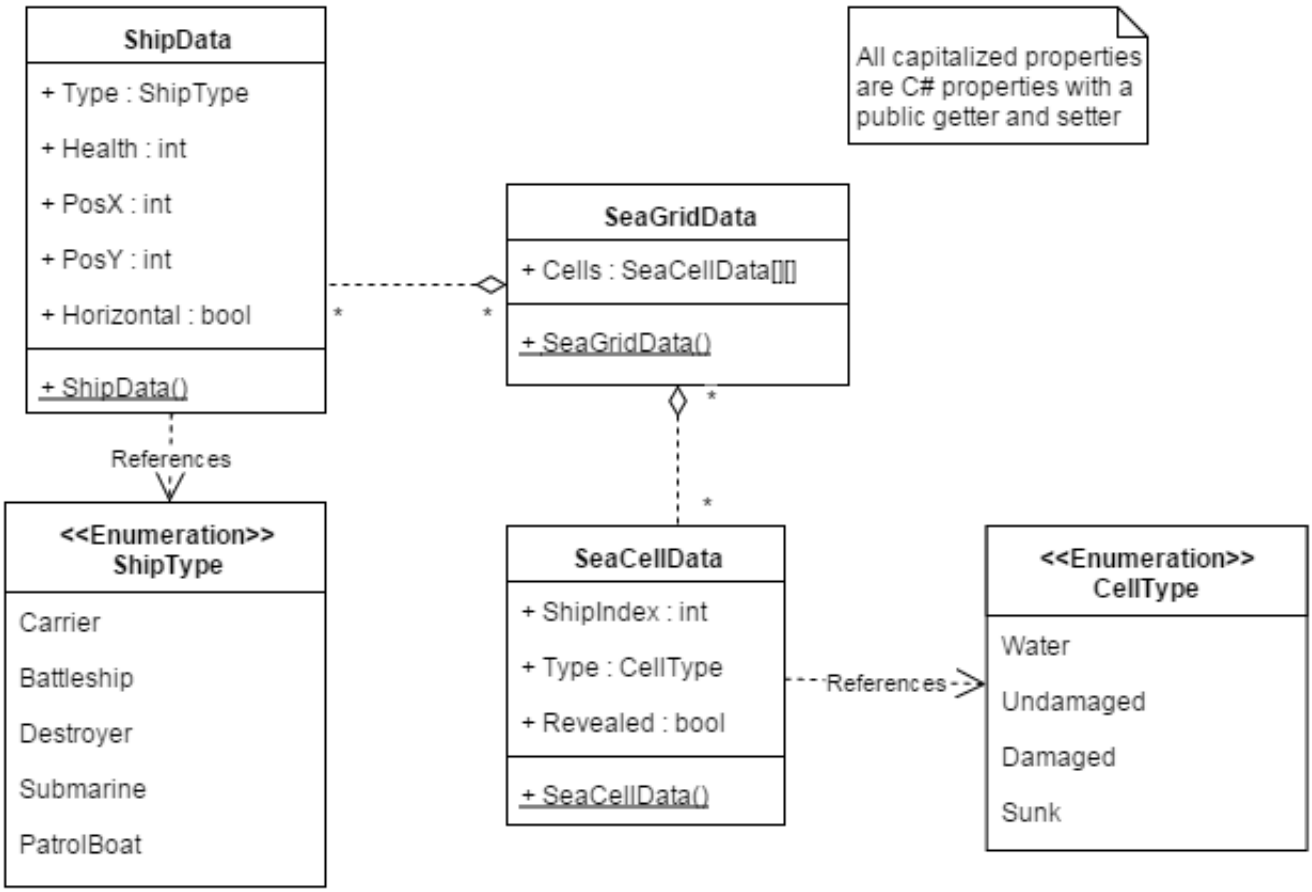


Figure 21: Game Data diagram

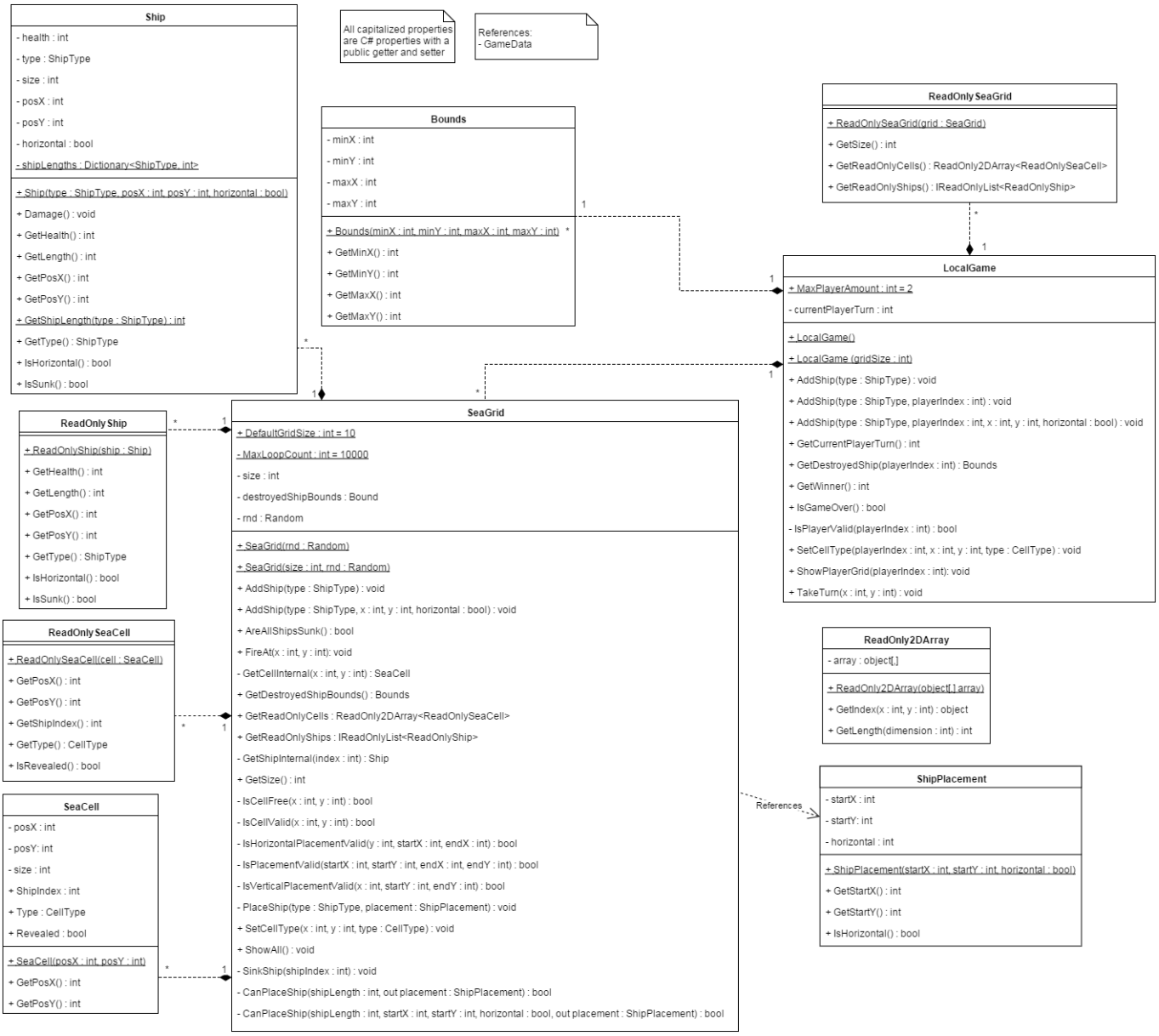


Figure 22: Game diagram

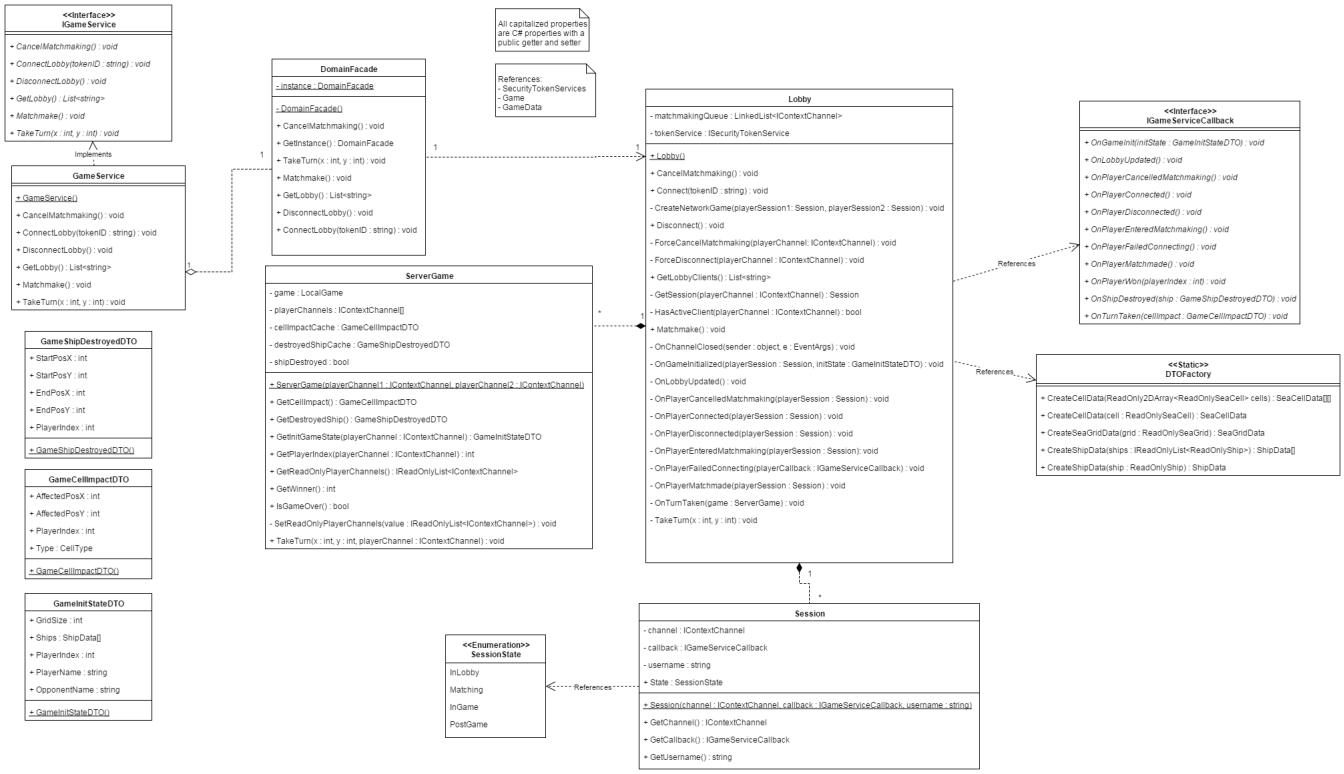


Figure 23: Game Services diagram

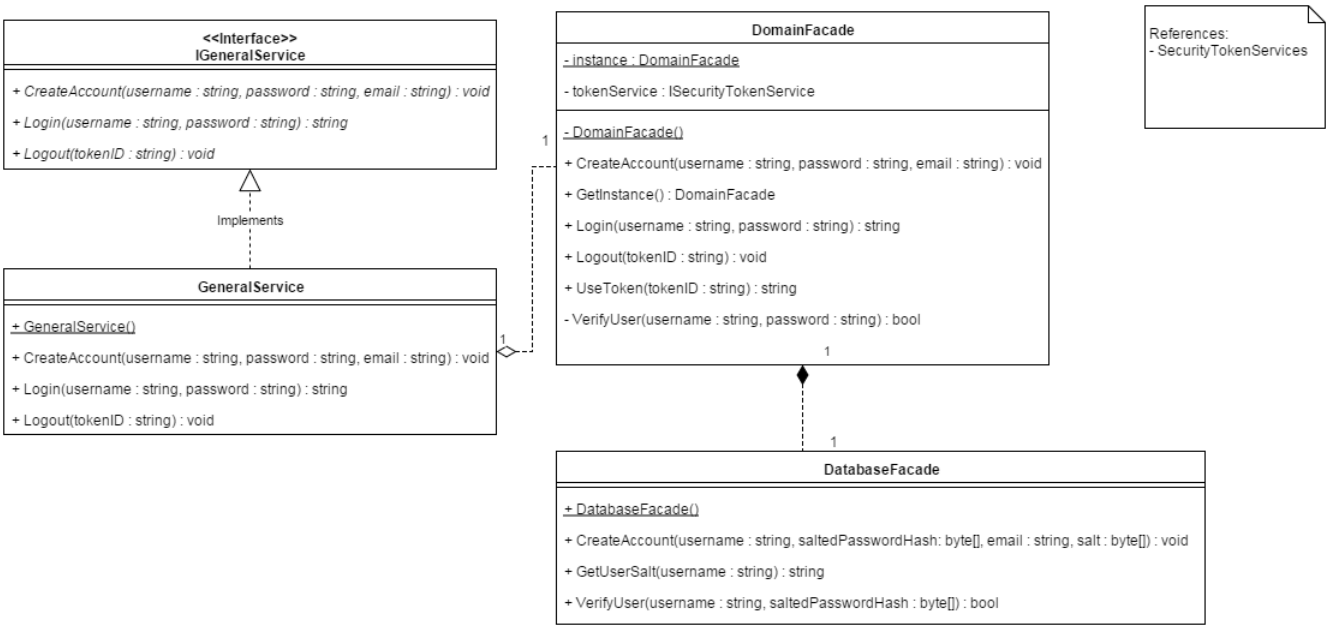


Figure 24: General Services diagram

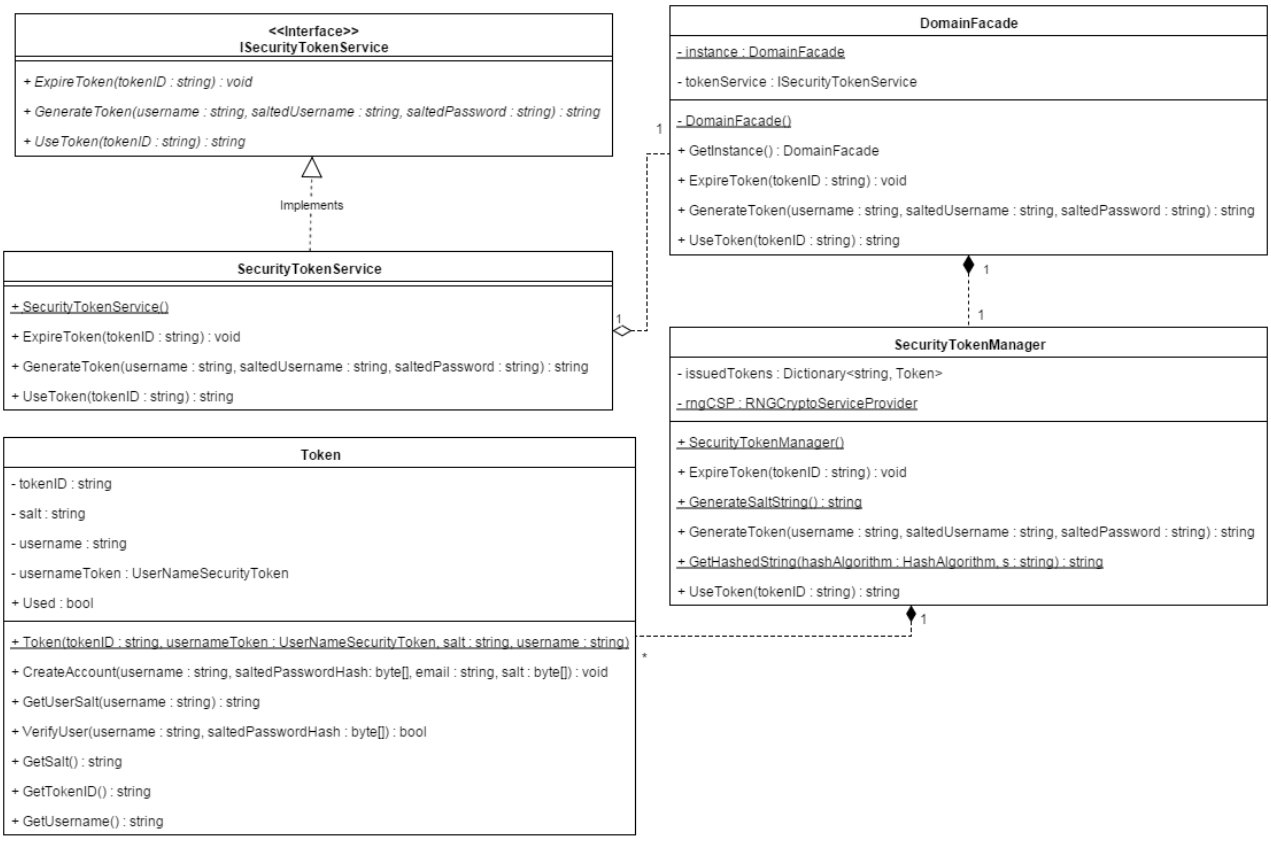


Figure 25: Security Token Services diagram

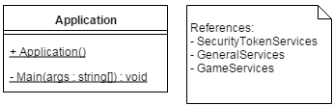


Figure 26: Server diagram

1.2 SOLID

There are 5 principles to SOLID:

Single Responsibility Principle (SRP)
There should never be more than one reason for a class to change.[24]

This principle focus on cohesion, and means that a class should only be assigned one responsibility, by splitting a class that has multiple responsibilities into different classes. This will allow for good maintainability, since if later some requirements changes, that would mean a change in responsibility amongst the classes as well.

Open/Closed Principle (OCP)
"Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."[25]

The principle means that we should write our software entities to be extended without modifying the code. This will be achieved by abstraction, since it allows us to change what the software entity does, without changing the code.

Liskov Substitution Principle (LSP)
"Subclasses should be substitutable for their base classes."[26]

This means that a subclass of the base class should still function properly, even if a derived class of the base class is passed to it. A violation of LSP, will also result in a violation of OCP.[27]

Interface Segregation Principle (ISP)
"Many client specific interfaces are better than one general purpose interface."[28]

This means that we should create specific interfaces for each client and inherit multiple of them, instead of one class with all the methods a client would need. This does not mean that every class that has a service, should have a special interface. Clients should instead be categorized by their type, and then create interfaces for each type. Then if different client types need the same method, it should be added to their interface.[29]

Dependency Inversion Principle (DIP)
"Depend upon Abstractions. Do not depend upon concretions."[30]
This means that every dependency should be on an interface or an abstract class, not a concrete class or function.

[24] srp.pdf, 02/2002, Robert C. Martin, page 1
[25] ocp.pdf, 01/1996, Robert C. Martin, page 1
[26] Principles_and_Patterns.pdf, 01/2000, Robert C. Martin, page 8
[27] Principles_and_Patterns.pdf, 01/2000, Robert C. Martin, page 12
[28] Principles_and_Patterns.pdf, 01/2000, Robert C. Martin, page 14
[29] Principles_and_Patterns.pdf, 01/2000, Robert C. Martin, page 15
[30] Principles_and_Patterns.pdf, 01/2000, Robert C. Martin, page 12

1.3 GRASP

There are 9 patterns to GRASP[31]:

Controller
Controller pattern is used to create a non-UI class that can represent more than one use case. An example of this could be create user and delete user which would have a single controller together.

Creator
The creator pattern uses one class to create other classes in cases where the creator monitors or uses the class it creates. It can have the initialization information for the class and passes it on when the class is created.

High Cohesion
High cohesion is a pattern that attempts to keep objects manageable and understandable. This is achieved by breaking programs into smaller classes and subsystems. If the system is a low cohesion the code can be hard to reuse, maintain and averse to change.

Indirection
Indirection pattern is a low coupling pattern. It takes the mediation between two classes and give them to an intermediate object. An example of this is the introduction of a controller for meditation between data and its representation.

Information Expert
Information expert is a principle is used to delegate responsibilities. Responsibilities includes methods, computed files, and so on. A common use is to look what information is needed to complete it and then determine where that information is stored.

Low Coupling
Low coupling is how much an object have knowledge, relies on and connected to other elements. The low coupling pattern is used to assign the responsibilities to support: low dependency, higher reuse potential and change in one class does not have a big impact on other classes.

Polymorphism
If it is needed to handle alternatives based on a type, polymorphism should be used. An example of this would be if there was two types Dog and Cat, they are both just alternatives to each other. Instead polymorphism should be used, which results in creating a common type Animal, where Dog and Cat are its alternatives by extending or implementing depending on the type.

Protected Variations
This pattern is similar to the Open/Closed principle of SOLID, and should be used when a change/variation in the software will or would cause the software to be unstable. This is done by using an interface and polymorphism to wrap the old code instead of modifying it directly.

Pure Fabrication
If the Information Expert fails the Pure Fabrication pattern steps in. This is done by creating a class that does not represent the domain concepts. In other words, if the responsibility assigned by the Information Expert does not meet the high cohesion, create a separate class and assign responsibility to it, so that it results in high cohesion.

1.4 UseCase Specification

[31] Applying UML and patterns, Larman 2005

Use case: UC02 - CreateMultiplayerGame

Short Description: The player wants to create a new multiplayer game

Primary actors: Player

Secondary actors: None

Stakeholders: The player gets a new multiplayer game

Overview: This feature will be used when the player wants to play online against another online opponent.

Precondition: The player is in matchmaking

Main course of events:

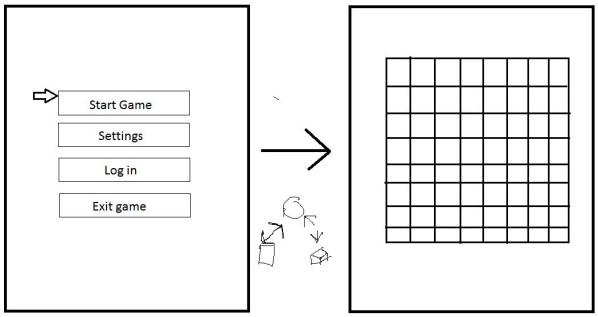
Actor: Player	Actor: Client
1. A player wants to create a multiplayer game.	
<<include>> US04 Matchmaking	
	2. The client receives the game data from the server.

Postcondition: The player has started the game

Alternative course of events: None

Cross references: (US04 - Matchmaking)

Sketch of user interface:



Sprint: 01
Responsible: Niels Nielsen, Patrick Florczak
Status: Ongoing

Use case: UC06 - TakeTurn

Short Description: Allows a player to take a turn in the game

Primary actors: Player

Secondary actors: Client

Stakeholders: Player

Overview: A player decides to take a turn, which allows the opponent to make their turn

Precondition: The player is connected to a game

Main course of events:

Actor: Player	Actor: Client	System
1. A player wants to take a turn	2. The client sends a request to take a turn for the player	3. The server takes the turn in the game
		4. The server sends a response back to both players
	5. The client gets the game data.	

Postcondition: The players has taken a turn

Alternative course of events: None

Cross references: US05 - ConnectGameServer

Sketch of user interface: None

Sprint: 05
Responsible: Martin Hubel
Status: Done

Use case: UC08 - CreateAccount

Short Description: Allows a player to create an account

Primary actors: Player

Secondary actors: Client

Stakeholders: Player

Overview: A player decides to create an account on the server with a username, password and email. The email is used to verify the account

Precondition: The player is not logged in, but is connected to the general services

Main course of events:

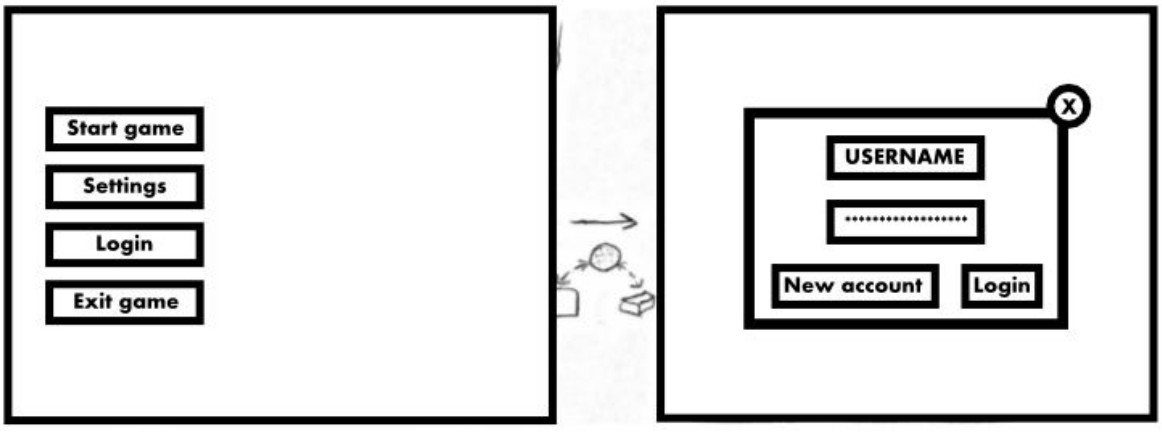
Actor: Player	Actor: Client	System
1. A player wants to create an account.		
2. A player inputs their username, password and email.	3. The client encrypts the credentials and sends them to the server.	4. The server creates an unverified account and sends an answer back to the client.

Postcondition: The player has created an unverified account

Alternative course of events: None

Cross references: US01 - ConnectToServer

Sketch of user interface:



Sprint: 02
Responsible: Martin Hubel, Klaus Riisom
Status: Done

Use case: UC09 - Connect lobby

Short Description: When a user log in as a player, they will then be put into a lobby with other players.

Primary actors: Player

Secondary actors: System

Stakeholders: The player will be able to easily see a list of other online players.

Overview: When a user wants to play, they login as a player. Then the player is put into a available lobby, in which the player can see other online players.

Precondition: The user has logged in.

Main course of events:

Actor: Player	System
1. A player wants to join a lobby.	
2. The player sends their token ID to verify their login status.	3. The system verifies the token ID.
	4. The system finds an appropriate lobby and adds the player.
	5. The system call back the player to notify a successful connection.

Postcondition: The player is put into a lobby.

Alternative course of events:

Cross references: US02 - Login

Sketch of user interface:

Iteration: #02 & #03
Responsible: Martin Fabricius, Patrick Blomberg Florczak
Status: Done

Use case: US01 - ConnectGeneralServer

Short Description: When the player login, the client creates a connection to the server.

Primary actors: Player

Secondary actors: Client

Stakeholders: The player gets access to the game's online features.

Precondition: None

Main course of events:

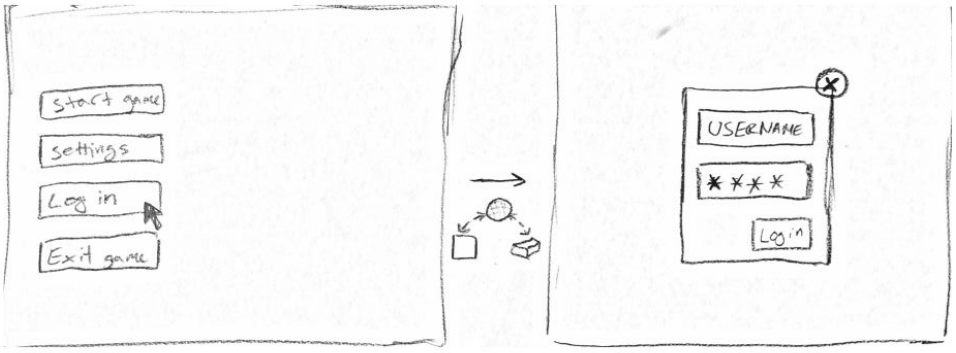
Actor: Player	Actor: Client	System
1. A player wants to connect to the general server.	2. Client sends message to Server.	3. Server receives the message and sends confirmation back to Client
	4. Client receives confirmation.	

Postcondition: Connection to the general server has been established

Alternative course of events:

Cross references: US02 - Login

Sketch of user interface:



Sprint: 01
Responsible: Martin Fabricius, Klaus Riisom
Status: Done

Use case: US02 - Login

Short Description: Allows a player to login on the server, verifying account information

Primary actors: Player

Secondary actors: Client

Stakeholders: Player

Overview: A player decides to login where they provide a matching username and password as credentials to the server.

Precondition: The player is not logged in yet

Main course of events:

Actor: Player	Actor: Client	System
1. A player wants to login.		
<<include>> US01 ConnectGeneralServer		
2. A player inputs their username and password, and submits it.	3. The client encrypts the credentials and sends them to the server.	4. The server authenticates the credentials and sends an answer back to the client.

Postcondition: The player is logged in

Alternative course of events: NewAccount

Cross references: US01 - ConnectGeneralServer

Sketch of user interface: See US01 - ConnectGeneralServer

Sprint: 01
Responsible: Martin Hubel, Rasmus Lassen
Status: Done

Use case: US02 - Login:NewAccount

Short Description: The player account does not exist, presenting the player with the option of creating a new account.

Primary actors: Player

Secondary actors: Client

Stakeholders: Player

Overview: When the player account does not exist when trying to login, the player is required to create a new account in order to be able to continue.

Precondition: The player does not have an account in the system.

Alternative course of events:
The alternative course of events begin after step 4 in the main course of events.

Actor	System
<<include>> US04 CreateNewAccount	

Postcondition: A new player account has been created.

Use case: US03 - Logout

Short Description: Allows a player to logout of the server

Primary actors: Player

Secondary actors: Client

Stakeholders: Player

Overview: A player decides to logout of the server after being logged in.

Precondition: The player is logged in

Main course of events:

Actor: Player	Actor: Client	System
1. A player wants to logout.	2. The client sends the token id to the server.	3. The server authenticates the id, logs out the client and sends an answer back to the client.

Postcondition: The player is logged out

Alternative course of events: None

Cross references: US01 - ConnectToServer

Sketch of user interface: See US01 - ConnectToServer

Sprint: 02
Responsible: Martin Hubel, Klaus Riisom
Status: Done

Use case: US04 - Matchmaking

Short Description: Allows 2 players to enter into a match against each other

Primary actors: Player

Secondary actors: Client

Stakeholders: Player

Overview: A player decides to matchmake, entering a queue waiting for another player

Precondition: The player is logged in and connected to lobby

Main course of events:

Actor: Player	Actor: Client	System
1. A player wants to start matchmaking.	2. The client sends a request to enter the player into a waiting queue.	3. The server enters the player into the waiting queue for the lobby.
		4. The server waits for more players to enter the queue.
		5. The server starts a game when another player can be matchmade.
		6. The server sends a message to the clients that they have been matchmade.
	7. The client gets the game data.	

Postcondition: The players has been matchmade

Alternative course of events: None

Cross references: US05 - ConnectGameServer

Sketch of user interface: None

Sprint: 05
Responsible: Martin Hubel, Klaus Riisom
Status: Done

Use case: US05 - ConnectGameServer

Short Description: Allows a player to connect to the game server

Primary actors: Player

Secondary actors: Client

Stakeholders: Player

Overview: A player decides to connect to the game server by supplying the server an access token.

Precondition: The player is logged in

Main course of events:

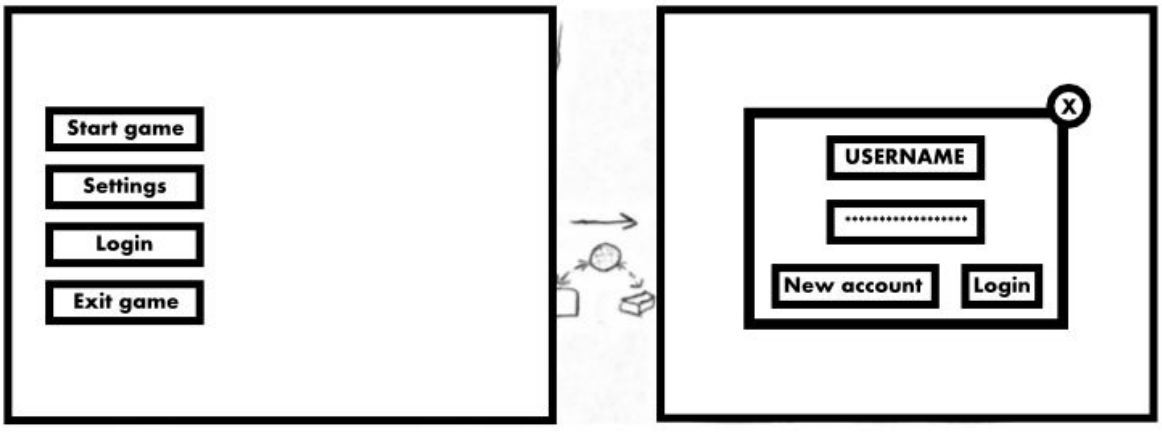
Actor: Player	Actor: Client	System
<<include>> US02 Login		
1. A player wants to connect to game server.	2. The client sends the access token to the server.	3. The server establishes an authenticated connection and sends an answer back to the client.

Postcondition: The player is connected to the game server

Alternative course of events: None

Cross references: US02 - Login

Sketch of user interface:



Sprint: 04
Responsible: Martin Hubel, Klaus Riisom
Status: Done

1.5 Collaboration Agreement

Academic goals

- Reach an understanding in the subject of Software System Design in a Global Context.
- Ability to utilise the experience from last semesters project.

Social goals

- Reach an acceptable collaboration environment.
- Sustain a reasonable atmosphere in the group to avoid problems between group members.
  - If any problems should occur, they should be solved swiftly and completely to avoid further issues.
- Problems are solved through direct contact with the person in question.
- If a problem still remains unsolved after contact with the person in question, contact should be established with the advisor Claudio Giovanni Mattera (cgim@mmmi.sdu.dk) and supervisor Ulrik Pagh Schultz (ups@mmmi.sdu.dk)

Level of ambition
The group's level of ambition is that all the members of the group perform their best to reach the highest of results.

Organisational and work norms

- In the event of major group disagreements a democratic voting system will be used, ensuring all voices are heard and compromises are made.
- The group meets every friday at 10 a.m., unless otherwise announced.
- If meeting attendance is not possible, it should be announced to the rest of the group as quickly as possible.
- Each group member is responsible for the content of the final report and other deliverables.
- If a break is needed, the group should be notified beforehand.
- If an assigned deliverable cannot be done in time, the group should be contacted beforehand, allowing other group members to be able to help if possible.

Group Norms

- Discussions must be kept at a reasonable level.
- Each group member cleans up after themselves.
- If attendance is not possible and no announcement has been made thereof, the matter may be brought up with the advisor or supervisor depending on the severity.

1.6 Advisor Agreement

The advisor has the responsibility to help the group in the right direction, but the responsibility of the group work and learning is entirely on the group and its members. All group members are expected to attend advisor meetings.

An agenda of subjects for discussion will be prepared and sent to the advisor at least a work day before a meeting. This agenda will be read by the advisor before the meeting. This meeting will be expected to take place once a week if requested by the group. Any cancellations of the meeting must occur a day before the meeting.

Other inquiries including deliverables and other material sent to the advisor is read and sent back with comments in a reasonable timeframe.

1.7 Sequence Diagrams

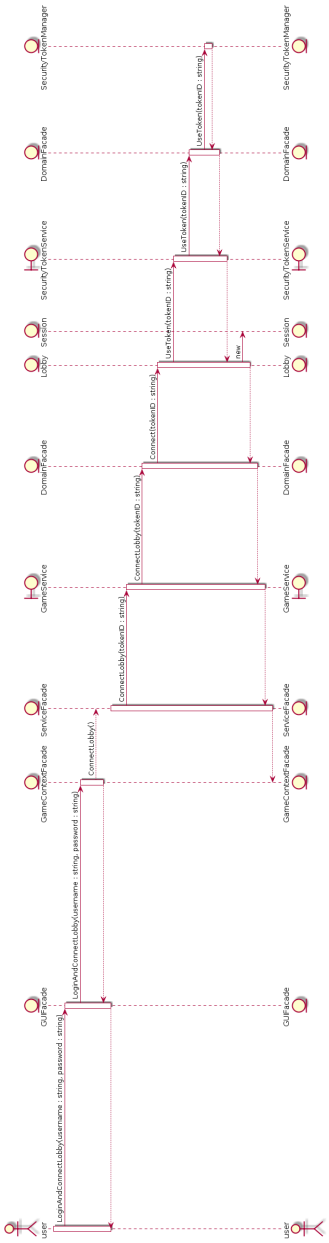


Figure 27: Connect to Lobby Sequence Diagram

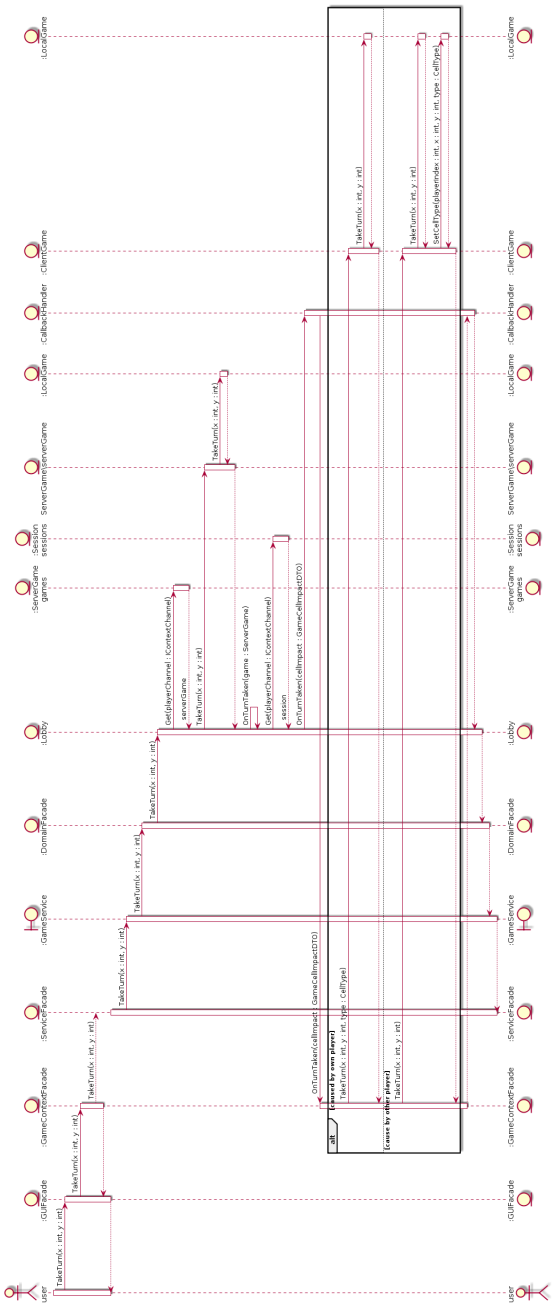


Figure 28: Take Turn Sequence Diagram