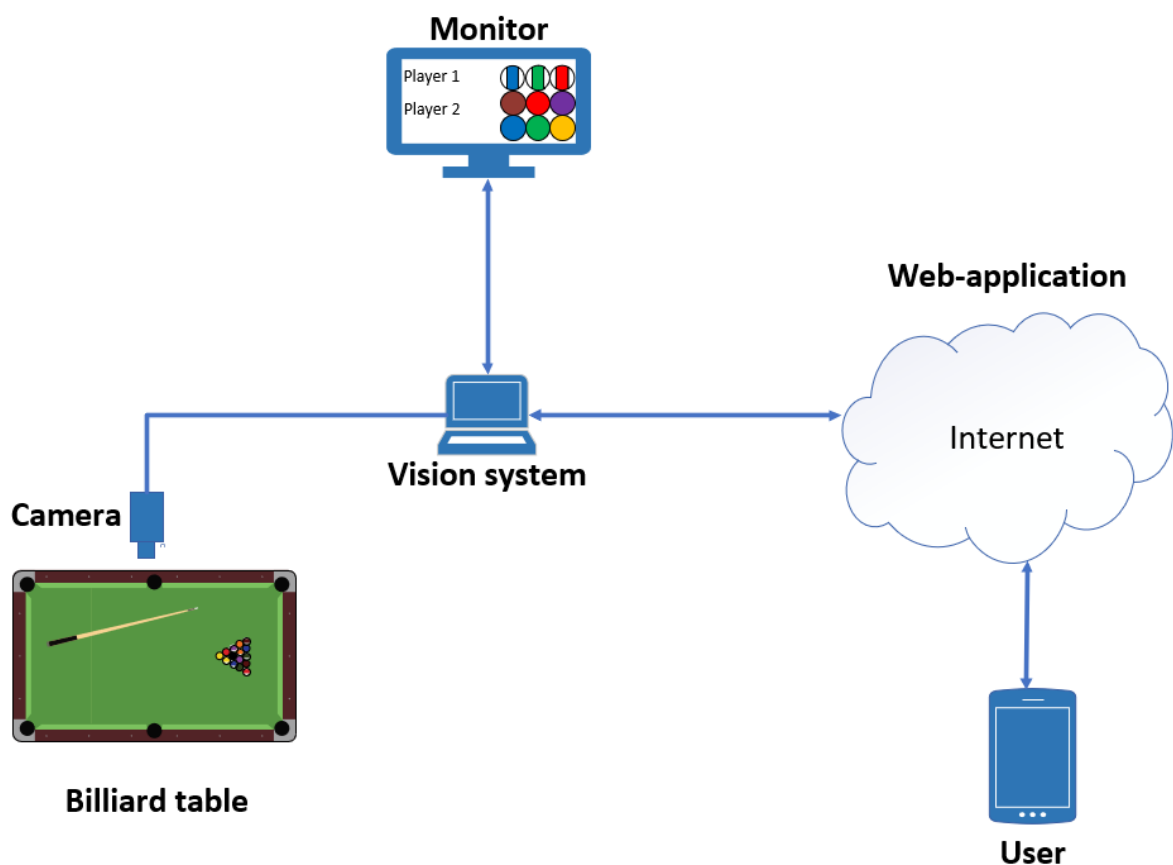


PRH612 Bacheloroppgave

# Development of a camera-based system for 8-ball using Azure Custom Vision



IA3-6-22

**Course:** PRH612 Bacheloroppgave

**Title:** Development of a camera-based system for 8-ball using Azure Custom Vision

This report forms part of the basis for assessing the student's performance on the course.

**Project group:** IA6-3-22

**Availability:** Open

**Group participants:**

Sander Mikal Blomvågnes  
Christian Hagrupsen  
Amanuel Isak

**Supervisor:**

Hans-Petter Halvorsen

**Project partner:**

Grenland Biljardklubb / Robert Immerstein

**Approved for archiving:**

---

**Summary:**

In the modern world, the need for digitalization to be included in various spheres of economic activities is increasingly growing [1]. This is because of the accessibility and availability that digitalization offers. Even international billiard tournaments and matches are no exception to this reality [2]. But what about relatively small and local billiard clubs, where there is activity relating to billiard games and a lack of digitalization. Therefore, in collaboration with Grenland Biljardklubb, an initiative to develop a digital solution for billiard games is embarked upon.

The main goal of this project was to create a camera-based system that is capable of monitoring and presenting the billiard game, 8-ball. To achieve this goal, a vision system application and a web application were created. The vision system application was created in Visual Studio, by using C#. The vision system is a WinForms desktop application that utilizes a machine learning model, through the internet to monitor billiard games. This application includes a GUI for presenting and configuring billiard games as well. The web application was created by using JavaScript in Visual Studio Code. The web application is the main approach for users to interact, by utilizing data transactions, with the vision system. Through the web application, the users are capable of viewing and configuring billiard games.

The project resulted in a prototype system that fulfills most of its required functionalities. To further optimize the system, additional development of existing, as well as implementation of new functionalities will be of the utmost importance.

# Preface

This project, and the following report is the result of a bachelor thesis developed by students at the University of South-Eastern Norway (USN). The group studies Computer Science and Industrial Automation at the Faculty of Technology, Natural Sciences and Maritime Sciences at USN campus Porsgrunn.

Thank you to our supervisor Hans-Petter Halvorsen for good communication and guidance throughout the project. Also, a special thanks to USN for providing a group room for the group, and to Robert Immerstein and Grenland Biljardklubb for letting us borrow their billiard table.

The cover photo used in this report shows an overview of the system developed in this project.

The vision system application and source code are stored in GitHub - <https://github.com/hagru/PoolBachelor/releases/tag/Alpha>

Link to web page - <https://smartpool.no/>

The source code for the web application is stored in GitHub - <https://github.com/SanderBlom/PoolFrontend>

Link to project web page - <https://dev.azure.com/BiljardBachelor/Vision%20based%20system%20for%208-ball%20pool%20Bachelor>

Throughout the project, a variety of different software and services were utilized. These services are listed below:

- Adobe Photoshop
- Custom Vision
- PgAdmin
- Draw IO
- Azure Custom Vision
- Azure DevOps
- Digital Ocean virtual machines and networks
- GitHub
- Microsoft Project
- Microsoft Teams
- Microsoft Visio
- Microsoft Word
- Visual Studio
- Visual Studio Code

Upon reading this report, one may find it easier to understand some of the contents if they have prior knowledge of object-based programming and relational databases.

Porsgrunn, 20.05.2022

# Nomenclature

API	Application Programming Interface
CLI	Command Line Interface
CPU	Central Processing Unit
DB	Database
FAQ	Frequently Asked Questions
GB	Gigabyte
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ID	Identification
IOU	Intersection over Union
IP	Internet Protocol
mAP	Mean Average Precision
NAT	Network Address Translation
NI	National Instruments
NOK	Norwegian Krone
NPM	Node Package Manager
NUC	Next Unit of Computing
RAM	Random Access Memory
SDK	Software Development Kit
SLR	Single Lens Reflex
SQL	Scripted Query Language
TV	Television
UI	User Interface
USB	Universal Serial Bus
USD	United States Dollar
USN	University of South-Eastern Norway
VDM	Vision Development Module
VPN	Virtual Private Network

WPF      Windows Presentation Foundation

# Contents

Preface.....	4
Nomenclature .....	5
Contents .....	7
1 . Introduction.....	10
2 . System description.....	12
3 . Software development plan .....	13
3.1 Scope of the project.....	13
3.2 How will this be achieved .....	13
3.3 Teamwork and development .....	13
4 . Software requirements specification .....	15
4.1 Vision system and GUI .....	15
4.1.1 <i>Vision system</i> .....	15
4.1.2 <i>GUI</i> .....	15
4.2 Web application.....	16
5 . Hardware.....	17
5.1 Camera .....	17
5.1.1 <i>Mobile phone camera</i> .....	17
5.1.2 <i>Logitech StreamCam</i> .....	18
5.1.3 <i>Camera mount</i> .....	20
5.2 Billiard table and balls.....	21
5.3 Computer.....	23
5.4 Final product.....	24
6 . Vision system.....	26
6.1 Evaluated software for object detection and classification .....	27
6.1.1 <i>NI Vision Development Module</i> .....	27
6.1.2 <i>OpenCV Python library</i> .....	27
6.1.3 <i>Azure's Computer Vision</i> .....	28
6.1.4 <i>Azure Custom Vision</i> .....	28
6.2 Azure Custom Vision .....	28
6.2.1 <i>Technology behind Custom Vision</i> .....	29
6.2.2 <i>Custom Vision UI</i> .....	31
6.2.3 <i>Prediction API</i> .....	39
6.3 Class structure .....	42
6.4 Pool rules.....	44
6.4.1 <i>Main methods to enforce the rules as a digital judge</i> .....	46
6.5 Graphical user interface of the vision system.....	47
6.5.1 <i>Ball design</i> .....	48
6.5.2 <i>Cue design</i> .....	49
6.5.3 <i>Game page design</i> .....	49
6.6 Final design .....	50
6.6.1 <i>Live video feed</i> .....	52

6.6.2 Processing .....	52
6.6.3 Timer .....	53
6.7 Simulation mode .....	53
<b>7 . Connection-API .....</b>	<b>54</b>
<b>8 . Database .....</b>	<b>56</b>
8.1 Evaluated databases .....	56
8.2 Database structure.....	58
8.2.1 Table overview.....	59
<b>9 . Web Application .....</b>	<b>61</b>
9.1 Libraries and package manager.....	61
9.2 Use case diagram .....	62
9.3 Program structure.....	63
9.4 Web pages and functions .....	64
9.4.1 Front page .....	65
9.4.2 Register page.....	66
9.4.3 Login page .....	67
9.4.4 Profile page .....	67
9.4.5 Create and join games .....	68
9.4.6 Create a tournament.....	70
9.4.7 Previous games.....	71
9.4.8 Administration panel page.....	72
9.4.9 Live gameplay page .....	74
9.4.10 Scoreboard page .....	75
9.4.11 FAQ pages .....	75
9.5 Communication and security .....	76
9.5.1 Communication with the vision system.....	76
9.5.2 Communication with clients.....	77
9.5.3 Communication with the database .....	77
<b>10 Testing.....</b>	<b>78</b>
<b>11 Deployment and distribution .....</b>	<b>80</b>
11.1 Deploying the web application .....	80
11.2 Installing the Smart Pool application .....	81
<b>12 Discussion.....</b>	<b>82</b>
12.1 Vision system .....	82
12.1.1 Custom Vision.....	82
12.1.2 Visual system GUI.....	83
12.2 Web application .....	84
12.2.1 Tournament mode.....	84
12.2.2 Creating games and tournaments .....	85
12.2.3 Live streaming .....	85
12.2.4 Connection with the vision systems .....	85
<b>13 Conclusion .....</b>	<b>87</b>
<b>14 References .....</b>	<b>88</b>
<b>Appendices .....</b>	<b>91</b>
Appendix A .....	92



## Contents

<b>Appendix B.....</b>	<b>94</b>
<b>Appendix C.....</b>	<b>95</b>
<b>Appendix D .....</b>	<b>96</b>
<b>Appendix E .....</b>	<b>97</b>
<b>Appendix F .....</b>	<b>98</b>
<b>Appendix G .....</b>	<b>99</b>
<b>Appendix H .....</b>	<b>101</b>

# 1 Introduction

Today's world is always changing. The demand for availability and accessibility is higher than ever and will continue to grow in the future [1]. International billiard tournaments and matches can already be followed closely by anyone through television and the internet, but what about the lower end of the scale? To try and make local billiard events more accessible, the group, in cooperation with Grenland Biljardklubb, set about to develop a system for 8-ball games for local billiard clubs. The goal of this project is to create a functional system, that can monitor and present 8-ball games. The system will provide users with information about games and players, as well as making it easily accessible and available to follow from different locations as a spectator. This could lead to a bigger following and increased interest in billiard games in more local areas, which in return will benefit both supporters and organizers.

To achieve the goal set in this project, a vision system application is created in Visual Studio, using C#. The vision system is a WinForms desktop application, that receives images of an ongoing 8-ball game from a camera and utilizes a machine learning model to detect and classify billiard balls within the images. It will also be able to store and retrieve critical data about the ongoing games to the cloud-based database, as well as allowing users to follow and configure an ongoing game on its GUI. The data stored in the cloud-based database, will in return be used to display games in the web application. The cloud-based database will be created by using PostgreSQL. The web application is created in Visual Studio Code, by using JavaScript. The web application is created to provide users with functionalities that are relevant to this project. The web application and the database are both hosted on DigitalOcean, which is a public cloud provider. The database is used by both the web application and the vision system application, to store and retrieve important information. The vision system application and the web application will also utilize an API to establish flexible, scalable, and easy communication between them. The API is called "connection-API" and is created by using Visual Studio.

The scope of this project is the detection and classification of the billiard balls from a still image of an ongoing 8-ball game, using a computer program. From there, the program will implement the game rules of 8-ball to gain information about the ongoing game. The presentation of this information will mainly be realized by the use of a web application and a desktop application. This project is limited to a general description of machine learning models and will not go in-depth of how the Custom Vision Model is constructed.

The list below will give an overview of the report contents and structure:

- Chapter 2 System description provides a short overview of the developed system
- Chapter 3 Software development plan gives an insight into what will be developed and how
- Chapter 4 Software requirements specification specifies the requirements for each module
- Chapter 5 Hardware discusses physical hardware used in the project
- Chapter 6 Vision system dives deeper into the vision system and its features
- Chapter 7 Connection-API explains the concept behind the connection-API
- Chapter 8 Database delves into the database structure and its features

- Chapter 9 Web application explains the workings of the web page
- Chapter 10 Testing discusses the testing process
- Chapter 11 Deployment and distribution explains how the system can be obtained
- Chapter 12 Discussion
- Chapter 13 Conclusion
- Chapter 14 References
- Appendices

## 2 System description

The system is mainly comprised of a vision system that monitors a game of 8-ball, a popular billiard-game. The vision system is created as a WinForms desktop application, that is installed on a NUC computer. The application uses a camera to capture the game of 8-ball. The camera will be mounted appropriately over a billiard table. The vision system uses a machine learning model to classify and detect the billiard balls within images captured from the camera. This machine learning model will be developed by using the Custom Vision UI. Once the model is developed, a prediction-API will be published. It is through the prediction-API that the vision system interacts with the model. Onwards, the vision system will interpret the game by applying the game rules. The vision system will use a modified version of the game rules for 8-ball to analyse the game progression and outcome. Following this task, the vision system will store and retrieve data about the game in a database. For the purpose of displaying an ongoing game locally, meaning physically near the billiard table, a GUI is created. In addition to showing a live stream of an ongoing game, the GUI will also show an updated scoreboard of the game. The web-application is a platform that allows client devices to configure games and tournaments. It also allows users to view player-details and monitor ongoing games remotely. The web-application will store and retrieve data from a database, and exchange data with the vision system through an API. The database and the web-application will be hosted on DigitalOcean. Figure 2-1 illustrates how the overall system is constructed.

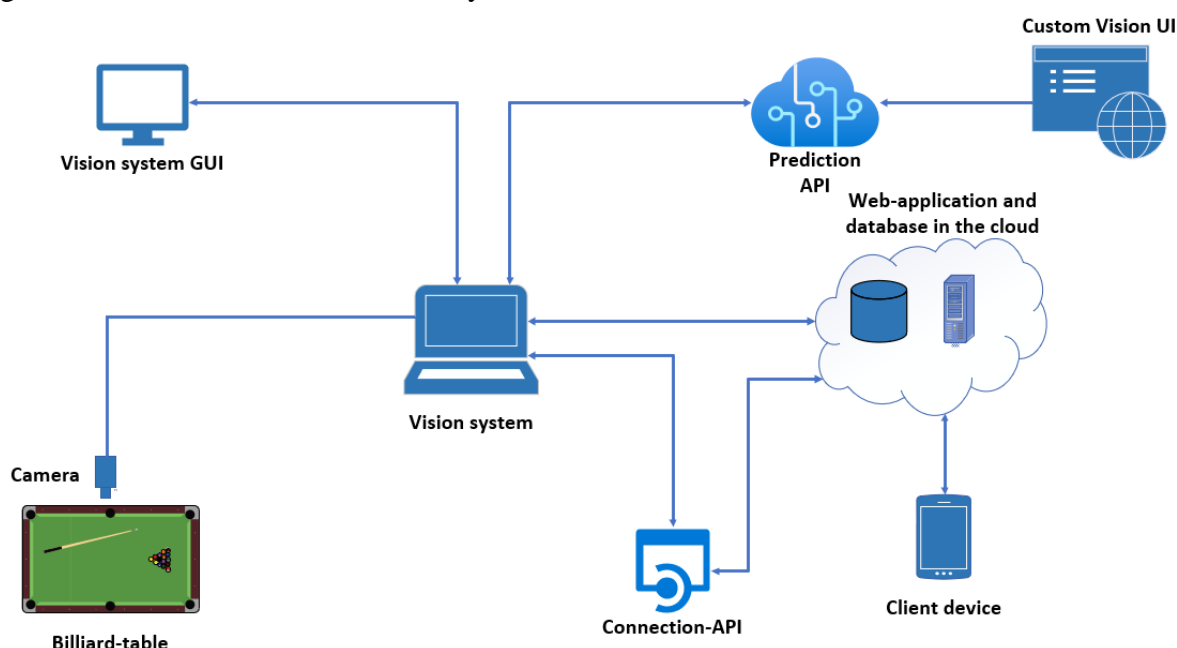


Figure 2-1: Overall system sketch of the vision system

## 3 Software development plan

In order to get a grasp of the project in hand, a short Software Development Plan is important. The plan lists the goal of the project, how to achieve it, and what measures is to be taken to keep track of the progress. Moreover, the relationships and teamwork within the group needs to be taken care of to achieve the best result possible.

### 3.1 Scope of the project

The broad scope of the project is to develop a vision system that tracks and displays information about a game of 8-ball. In more detail, the main task of this project is to detect billiard balls from a still image and determine exactly which ball is present on the table. Following this, the group will make a set of applications that take advantage of and showcases the modern technology of a machine learning model.

### 3.2 How will this be achieved

To achieve this, a main vision module will be set up to interpret an image of an ongoing game. There will be mounted a camera above the table to gain a good view of the table itself. A new picture will be sent to the system continuously, and it will be trained to recognize the wide variety of balls on the table. Data accumulated from the vision system, such as the position of the balls, as well as if the ball is even on the table, will be sent to a database, where it can be used by the web application. Because of this, there must be a connection between the application and the database.

Through the web application, one should be able to register and / or log in to an existing user. The users should be able to select an ongoing game, and get information, such as who's playing, whose turn it is, statistics, and even arrange tournaments or games from anywhere in the world. The vision system should display information locally about the current game and it should be easy to understand for both players and spectators.

Further into the report, the reader will get a better overview of the applications and their features.

### 3.3 Teamwork and development

Good teamwork and structure are important to get the best result possible. Azure DevOps is a phenomenal tool for project planning. A project is set up in Azure DevOps, and tasks are added to a task board in the beginning of each sprint, which lasts 1-2 weeks, depending on the amount of work that needs to be done. The group is using scrum as a framework for developing and working together on the project, giving a good environment for teamwork and structure. As well as making teamwork easier, it also allows for each group member to be in charge of their own parts, while keeping close touch with the other members through scrum meetings and continuous talks about progress, problems, or discussions [3]. Azure DevOps is integrated in Microsoft Teams, which is the group's preferred way of storing and working together simultaneously on documents such as this report. GitHub will be used to store the source code and work together on the development of the applications.

### **3 Software development plan**

The overall schedule for this project is January 10<sup>th</sup> through May 20<sup>th</sup>, which is the deadline day for this project. Development should therefore be done in good time ahead of this deadline, and the report should be continuously updated throughout the entirety of the duration.

In relation to development, everything should be planned carefully, and it will be important to take one task, or add one feature at a time. That way, as the time to develop a fully operational system is short, even if the group doesn't have time to implement every feature, the ones that have been added will work, and the base of the whole project will be solid. Programming languages used in this project will primarily be C#, HTML, JavaScript, and SQL. CamelCase notation will be used for methods and variables across the entirety of the coded product.

As stated above, the project is divided into two main parts. The vision system and the web application. The vision system is the basis of the project, while the main purpose of the web application is to allow people to configure games, but it will also include other features. Game viewing, tournament configuration and player stat viewing will be among these features. These two applications will run in harmony together and are both dependant of each other in certain ways.

## 4 Software requirements specification

Before the development process could start, the software requirements had to be determined and discussed. Figure 2-1 above shows an illustration of various parts that collaborate to create a system that tracks a game of 8-ball. Two of the most important collaborating parts are the vision system and the web application. This chapter will describe their functional requirements in detail.

### 4.1 Vision system and GUI

One of the fundamental parts of the system is the vision system. The vision system consists of a program responsible for making predictions by utilizing the Custom Vision model, as well as keeping track of the game and the game rules. Working together with the graphical interface, it also works as a display for the system.

#### 4.1.1 Vision system

This subchapter presents the functionalities that the vision system must establish, for the system to operate accordingly. This means that the vision system will not function accordingly, and subsequently the web application will not be fully functional if these requirements are not fulfilled. The functionalities are listed below.

- Receive images and a live video feed of ongoing game from the camera
- Ball detection and classification by utilizing Azure's Custom Vision
- Interpret games by implementing game rules of 8-ball.
- Update, store and retrieve data of ongoing game from database
- Retrieve and send data to the connection-API.
- Transfer data and a live video feed of the ongoing game to the GUI

#### 4.1.2 GUI

The GUI should be a viewer friendly application that can be displayed on screens around the pool table to display information about the ongoing game. Included as the most important features is a timer to show the full duration of the specific game. On the more technical side, the GUI shows a live video feed from a bird's eye perspective, allowing on site spectators a better look at the current situation on the table. To assist the players, the on-screen display should show how many, and which balls remains to be pocketed for each player. It should also be possible to easily see whose turn it is, the name of the players, showing who's playing whole and who's playing half balls.

- Show live video of ongoing game
- Show scoreboard
- Indicate whose turn it is
- Instruct the vision system to activate next play
- Display which balls remains to be pocketed for each player
- Show duration of the game

## 4.2 Web application

The web application will be the main application that the users directly interact with. The application must work just as great on a computer as on a phone or a tablet. The UI should be minimalistic, easy to understand and navigate. When entering the website, the user must immediately understand how start playing.

The system has a hard requirement, that all users who wants to use the system must be registered with an account or sign up for one. The billiard tables will still be usable but without the data acquisition.

The web applications requirements are listed below.

- Users should be able to register an account.
- Users should be able to sign into their account and be presented with their own personal page.
- Users should be able to see statistics about their previous games.
- Users should be able to see a playthrough if their games.
- Users should be able to start a regular game or a tournament.
- Users should be able to cancel a game or leave a tournament.
- Everyone on the page should be able to see the game rules (including the ones without an account).



## 5 Hardware

In this chapter about hardware, the physical equipment that have been tested will be listed with some information about the results. Throughout the project, and especially the testing stages, a lot of equipment have been put to the test to see what works best. Availability, compatibility, and affordability are three keywords that have been given a lot of weight, especially with a good result in mind. Billiard tables and equipment as far as billiard is considered have been limited, but every other aspect have been going through a lot of changes and trials to achieve the best finished product as possible within achievable reason and time frame. As Figure 2-1 shows, a pool table, a camera and a computer are the minimum requirements for the system to work as intended. The camera will also need to be mounted in a stationary place above the table. These aspects will be covered more in this chapter.

### 5.1 Camera

During the early stages, a variety of cameras were evaluated to get an indication on what works best for this project. With quality for a relatively cheap price in focus, everything that was available got put to the test. The results vary a lot, so it is important that the solution is relatively affordable and delivers a good and reliable output for the viewing pleasure of the spectators. The camera, which can be anything from a simple web camera to a professional SLR camera, needs to be mounted above the table. Preferably the camera should only capture the playing area of the table. Starting with various available mobile phones, moving on to web cameras with different software options, as well as having an expensive SLR camera available if needed.

#### 5.1.1 Mobile phone camera

Mobile phones were available and came in handy while testing the vision systems. Picture samples were gathered by using both an iPhone 8 and an iPhone 13 Pro Max. While the camera on the 13 Pro Max were considerably better, the results didn't vary that much during testing. Figure 5-1 below shows a picture from an iPhone 13 Pro Max, and the result is pretty good. The quality is top notch, and the colours look crisp and clean, just like in real life. For usage with a finished project, the phones are not a viable option, as it is preferred to have a set camera always mounted above the tables for simplicity, cost, and other reasons like the players being required to bring their own phones. Phones therefore turned out to be a useful tool for testing and early-stage vision system training, but not viable as a final solution.



Figure 5-1: Image taken with an iPhone 13 Pro Max

### 5.1.2 Logitech StreamCam

As the testing went along, the option to use a web camera presented itself. The Logitech StreamCam is a high end, quality web camera, and seemed like a good alternative for the task at hand. In Figure 5-2 below, there is a picture of the Logitech StreamCam that has been tested. The camera itself is a so-called plug and play camera, which means it doesn't require any additional software. After some testing, it became clear that extra software existed, and was very much a requirement in order to achieve a good result. The addition of extra software will be discussed further later in this subchapter. The cable connected to the camera is a USB-C cable, however, it is a little short for easy use being mounted in the ceiling above the pool table. A USB-C extension cable is therefore something that needs to be added to the final solution.



Figure 5-2: Logitech StreamCam, similar to the one tested and used in this project [4]

During testing in a group room, the camera looked really promising. Everything was high quality, and the autofocus worked very well. However, in the room where the pool table is stationed, the lighting is not the best, so without the option to manually adjust focus or any other settings, the outcoming result was mediocre at best. The result without adding custom camera settings can be seen in Figure 5-3 below.



Figure 5-3: Picture taken during testing of Logitech StreamCam without custom settings

From the picture above, it's clear that the quality is not very good. The image looks washed out, and the lighter the colour of the ball, the whiter the ball looks, which for example makes the yellow and orange balls look completely white. This is a big issue if the system can't determine what colour the ball is. The format and resolution that the camera uses to capture an image also needs to be altered to fit the whole table. To give a good example of the poor conditions with lighting and how it reflects and looks to the camera, one can look at the figure above. Focusing on the orange ball near the lower centre of the image, even the human eye struggle to determine if it is an orange ball, and furthermore, if it's a solid or half ball. The cost and quality of the camera in environments where the light is good make this camera an option. However, in the environment where the system is being developed and tested, it seems like a poor choice without extra software to adjust focus, white balance, sharpness, and other necessary options.

Looking at the results above, it is obvious that without some extra help from software to adjust settings, this camera would not be a viable option. A camera settings menu was added to the project. This menu lets the user set up and adjust settings to their liking and helps massively in this case. Logitech StreamCam has a built in autofocus function, which worked flawless in a well-lit environment, even at longer distances. However, the autofocus struggled a lot with focusing, and most of all keeping focus on the pool table and the balls spread across it. The settings menu lets the user turn off this autofocus feature and adjust the focus, as well as white balance, sharpness, contrast, brightness, colour saturation, and more to customize and adjust the quality of the video in real time. Another great feature is the ability to scale the image,

making sure the pool table covers the entire frame, which in turn for example makes the ball positions more accurate for drawing in the web application further along the process, among other things. It is important to mention that this settings menu is not developed during this project but is a part of Advanced Camera Settings in Windows. This camera settings menu is implemented through a few files that includes command lines to launch this particular settings menu for the selected camera. An overview of this settings menu and its recommended settings can be seen in Appendix B.

After a lot of adjustments and testing to find a good balance, the result compared to the picture without custom settings is night and day. This is displayed in Figure 5-4 below, the picture looks colourful, sharp, and high definition. The balls are easily distinguishable, especially to the human eye. Yellow no longer looks white, but yellow, like it is supposed to. Furthermore, the model must be trained using a wide variety of images. Of course, the model can be trained to recognize balls in each different table, where the lighting and software adjustments might be different. Training with different environments proved difficult with the resources available during the project. However, after testing a few different options, this is the camera and configuration that will be used for further development of the finished product.



Figure 5-4: Result after adjustments through third-party software

### 5.1.3 Camera mount

To achieve a bird's-eye view of the pool table, the camera needs to be mounted in the ceiling, or in another overhanging structure above the table. The Logitech StreamCam comes equipped with a camera mount right out of the box. As one can see in Figure 5-5 below, the mount is solid and flexible, giving it a lot of options for adjustment. In this project, where the options for mounting above the table is limited, it has been mounted into a panel in the ceiling, giving it just about enough space to cover the whole table. Considering the focal length of 3,2

millimetres that the web camera has to offer, the field of view is good. There's no option to mount a different lens to the camera. To cover the entire 180-centimetre-long table used in this project, it must be mounted about 150 centimetres above the table. Consequently, it must be mounted even further above the table if it is of a bigger size. When mounting the camera, it must be kept in mind that there needs to be a physical connection through a cable between the camera and a computer. The cable from the camera itself is not too long, so an extender will have to be considered for a clean and comfortable setup.



Figure 5-5: Logitech StreamCam mounted on the camera mount [5]

## 5.2 Billiard table and balls

The billiard table that has been used during the development of the software has been a standard small 6 by 3 foot, or about 180 by 90 centimetres table with a blue cloth, just like the one in Figure 5-6 below. This type of table is not the most common around billiard clubs and other public places. For playing at home, this type of table makes the most sense, as it occupies less space, it is easier to move around, and cost less compared to a bigger table of higher quality. However, as they are less sturdy, they are more prone to damage, defects and other factors that could cause problems that would not be acceptable at a higher level of play. The table used in this project had bends around the centre holes, making balls in their vicinity change direction and roll into the holes without necessarily being hit in that direction. Adding to this, the legs of the table were not the most solid ones, making the table very sensitive to contact, meaning a ball right on the edge of the hole could easily be put in by giving the table a little wiggle.



Figure 5-6: Pool table and equipment identical to the one used in this project [6]

The most common table in professional and recreational play outside the home is 9 by 4,5-foot tables. This is the standard regulation size. However, there are a lot of variations, as shown in Table 5-1 below, such as 7 and 8-foot tables that are also widely used, especially in venues like bowling alleys, bars or similar venues that find it beneficial to fit more tables in a limited space. They are built for that kind of environment and are more solid than the smaller tables. Standard 8-ball pool balls with the traditional colours were what the group had available. Consisting of 16 balls, 7 half, 7 solid, one black and the white cue ball, this is the most used ball configuration around the world.

Table 5-1: Different sizes of standard pool tables [7]

Table	Usage	Size in inches (length x width)	Size in centimetres (length x width)
<b>6-ft</b>	Small home table (Used in this project)	70-74" x 35-37"	178-188cm x 89-94cm
<b>7-ft</b>	Bar table	74-78" x 37-39"	188-198cm x 94-99cm
<b>7-ft+</b>	Large bar table	78-82" x 39-41"	198-208cm x 99-104cm
<b>8-ft</b>	Typical home table	88" x 44"	224cm x 112cm



<b>8-ft+</b>	Professional 8ft table	92” x 46”	234cm x 117cm
<b>9-ft</b>	Standard regulation size	100” x 50”	254cm x 127cm
<b>10-ft</b>	Oversized	112” x 56”	285cm x 142cm
<b>12-ft</b>	Snooker	140” x 70”	357cm x 178cm

As seen in Table 5-1, there are a handful of options. The table used in this project is the 6-ft table with a blue cloth. To support other types of tables like 6-foot tables with green or red cloths, or bigger tables like 9-foot tables, it is possible that the model would have to be trained and adjusted to the new colours. Size wise, there shouldn't be much of a problem if the camera captures the whole table, but the colour of the cloth, and the resulting reflection on the balls could have an impact. This has not been tested, and it is not clear whether this would affect the model and its accuracy to the point where it is straight up unreliable. This is because of the limited resources that were available.

### 5.3 Computer

The vision system application is based on Microsoft's WinForms, and the connection-API is based on web API. Therefore, the software will only run natively on a Windows platform. Windows 10 or 11 will therefore be a hard requirement for the software to run as intended. Other hard requirements would be USB 3.0 input, .NET 4 and 6 runtimes installed, HDMI out and enough processing power and ram to run the application. During testing, 4GB of RAM on a newer 2 core processor was found to be the minimum requirements for the application to run smoothly.

Table 5-2: Evaluated computers

Name	Price NOK	Power consumption	Operating system	Size in litres
<b>HP 260 G4</b>	5500+	5 – 35W	Windows	1
<b>Intel NUC BXNUC10I3FNH2</b>	5500+	5.5W – 25W	Windows	0,7
<b>Lenovo Thinkcentre M710q (Used)</b>	3000+	10 – 50W	Windows	1.1
<b>HP EliteDesk 800 G1 (Used)</b>	2500+	8 – 50W	Windows	1.1
<b>Lenovo Thinkpad L440 (Used)</b>	2000	9 – 37W	Windows	2.38

In Table 5-2 one can see an overview of computers that have been evaluated as candidates for the vision system. Ideally, a small, low power and inexpensive computer that could be mounted behind a TV or placed in a location out of view to run the software is preferred. Because of these soft requirements, both new and used hardware have been evaluated.

## 5.4 Final product

As stated in the beginning of this chapter, the group had different options regarding pretty much every aspect of the hardware side of the project. When it comes to the of the size of the billiard table, it will most likely not have an impact on the rest of the system, other than how far above the table the camera needs to be mounted. The Logitech StreamCam and the included camera mount has turned out to be a very good option, as it was light weight, inexpensive, and easy to set up. The price of the camera is about 1000 NOK and including an extension cable that is most likely needed, the total camera cost would be about 1200 NOK at the time of writing this report.

A billiard table may be the most expensive part of the system, but the price of this has not been taken into consideration, since this is something that is expected that the costumer already owns.

To run the application, a computer is needed. Multiple computers have been used during testing and development, both older and newer models. At the end of the project the group was provided with an older Intel NUC computer for use during testing of the system. Even though the NUC was released in 2015, it could run the required software without any significant issues. Newer versions of NUC are available for about 5500 NOK.

The fourth part needed for the system is an external monitor or TV that can display the vision system GUI in native 1920x1080 resolution or higher. The size of the monitor should be 50 inches or bigger. The further away the audience are sitting, the bigger monitor is needed. During the development, a 55-inch monitor has been used.

The last hardware needed is the mouse, keyboard, and necessary cables. An USB-C extension cable for the webcam is most certainly needed because of its short length. A wireless keyboard with a touchpad would be preferred over a wired one.

Looking at the hardware, which often can be biggest capital expense, the system is relatively inexpensive. There is more expensive hardware that could improve the look, finish, and performance of the system, but the gains are miniscule, and it will push up the overall cost up as well.



Table 5-3 provides an overview of the required hardware and an estimated price for all the hardware needed for one table.

Table 5-3: Estimated cost of required hardware

Product	Estimated Price	Description
<b>Web camera</b>	1000 NOK	Logitech StreamCam
<b>Computer</b>	5000 NOK	Mini desktop with i3 processor and 8GB RAM
<b>Monitor</b>	5000 NOK	Monitor with 55 inches
<b>Cables</b>	500 NOK	HDMI, USB and Power
<b>Wireless mouse and key-board combo</b>	500 NOK	LOGITECH K400

To decrease the price, the customer can buy the hardware in bulk and or buy refurbished hardware (used). Buying refurbished hardware can reduce the initial hardware cost and it will result in less new hardware being produced which can positively impact the environment [8]. This can reduce the carbon footprint of the customer.

## 6 Vision system

The vision system's main responsibility is to detect, classify the billiard balls, and interpret the game of 8-ball, by applying the game rules. It will keep track of the scoreboard and manage whose turn it is to play. Additionally, it will store data about an ongoing game in the database. This data can later be used by the web application. The vision system is developed in Visual Studio as a WinForms application and installed on a NUC. The vision system utilizes Azure's Custom Vision for the detection and classification of billiard balls. This operation is achieved by sending images to an API that Custom Vision provides as part of their service. The vision system will receive live video and images from the camera. The live video will be transferred to the vision system GUI, along with data about the ongoing game. The GUI is responsible for displaying the live video and additional graphical elements. This chapter presents a description of how the vision system and its GUI is constructed. This description will entail a presentation of the design-basis and the final design for the GUI. It will also give an evaluation of various object detection and classification software, an overview of the technology behind Custom Vision, and how Custom Vision is used in the project. Additionally, this chapter will go through the structure of the vision system and give a description of how the game-rules are applied. An overview of the vision system is illustrated in Figure 6-1.

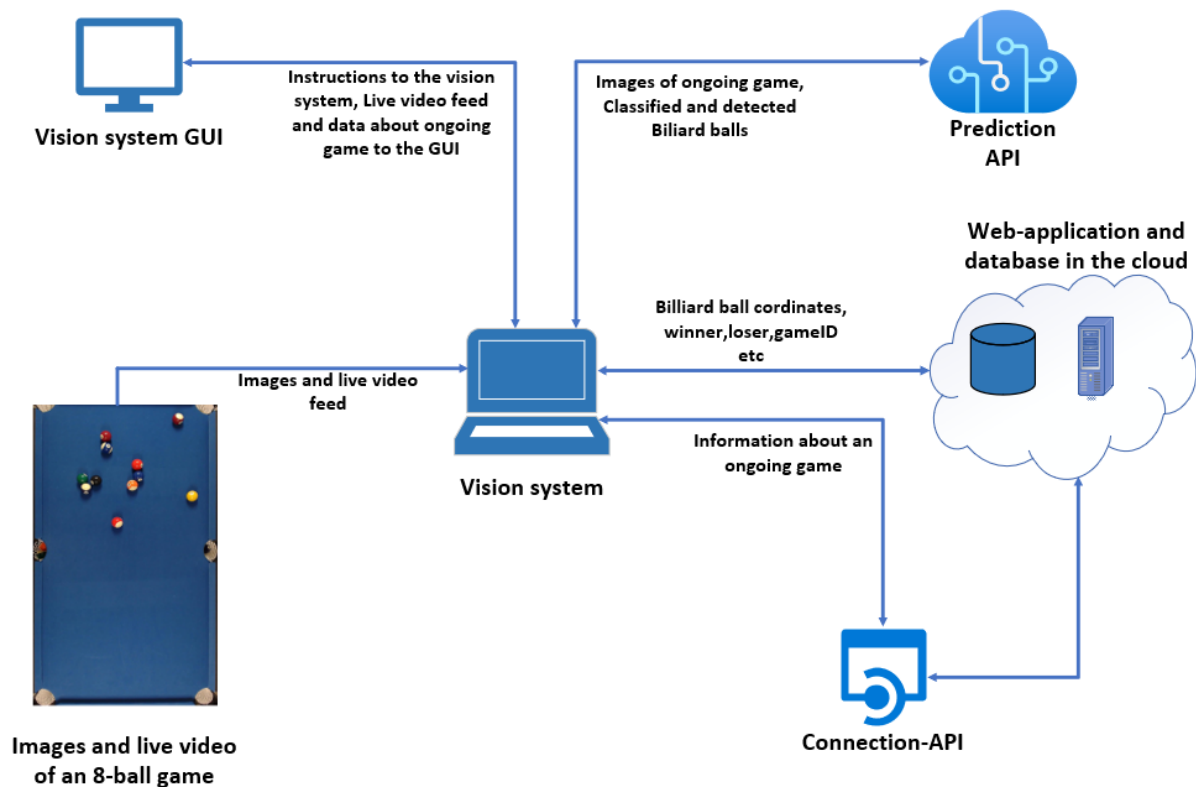


Figure 6-1: An overview of the vision system

## 6.1 Evaluated software for object detection and classification

One of the requirements of the vision system, is the detection and classification of billiard balls through an image. This subchapter will describe and specify the evaluated software, for the object detection and classification part of the system. Table 6-1 presents a list of various specifications of the different software that were evaluated.

Table 6-1: List of evaluated software

Name	Cost	Languages	Supported OS's
<b>NI Vision Development Module</b>	Expensive	LabVIEW, C, C++, and C#	Windows, Linux
<b>Python's OpenCV libraries</b>	Free	Python	Windows, Linux, macOS
<b>Azure's Computer Vision</b>	Free	C#, python, Java and GO	Windows, Linux, macOS
<b>Azure Custom Vision</b>	Free or inexpensive	C#, python, Java and GO	Windows, Linux, macOS

### 6.1.1 NI Vision Development Module

The NI Vision Development Module is relatively expensive. It consists of a development module, a deployment module, and a debug module. The development module will cost approximately 23 500 NOK to license per year, while the deployment module will cost approximately 5 500 NOK for a yearly licence. At last, the debug module will cost approximately 14 500 NOK to license each year. The VDM can be developed using programming languages such as LabVIEW, C, C++, or C#. The group decided that using the VDM for tracking a game of 8-ball pool would not be feasible. The group's reasoning for this decision, is mainly the cost of the license and the group's lack of knowledge in LabVIEW programming [9].

### 6.1.2 OpenCV Python library

OpenCV is open source and free to use. The service can only be utilized by using python and it is rather difficult and time consuming to implement to the larger system. The applications that utilize OpenCV can run on any system that supports Python 3. The group decided that,

because of the group's lack of experience with python programming, would not be a good choice in this insistence [10].

### 6.1.3 Azure's Computer Vision

Azure's Computer Vision is a free service. This service can be utilized by using C#, Python, Java or GO. The application that incorporates the library can run on Windows, Linux or macOS. The group decided not to pursue this option. This decision was made because the group discovered another service, that offered more flexibility. The other option is Azures Custom Vision [11].

### 6.1.4 Azure Custom Vision

Azure's Custom Vision has both a free and a paid version. The paid version is significantly cheaper than the option from NI. The user is only required to pay for the training of the machine learning model. This will be a onetime fee instead of a subscription-based model. This will be explained more detailed later in the report. This service can be utilized by using C#, Python, Java or GO. The application that incorporates this service can run on Windows, Linux, Unix and macOS. The group decided that Custom Vision would be the best option, because it is relatively affordable and in comparison, to Azure's Computer Vision, it offers more flexibility [12].

## 6.2 Azure Custom Vision

Custom Vision is the software that is used, to detect and classify billiard balls in the vision system. This software is a cloud-based SaaS that Azure offers as part of their cognitive services. Custom Vision empowers developers by offering prebuilt machine learning models. Developers can then quickly implement the models to develop an application that uses Custom Vision, without having any prior knowledge on machine learning models [13]. The Custom Vision UI, which is reached by searching "customvision.ai" on the web [14], is used to create, train, and publish a model that can detect and classify billiard balls. The vision system uses this model, through a prediction-API, that is provided by Custom Vision itself. Figure 6-2 illustrates how the Custom Vision is used. This subchapter will give an overview of the technology behind Custom Vision, a description of the Custom Vision UI and a description of how the Custom Vision prediction-API interacts with the vision system.

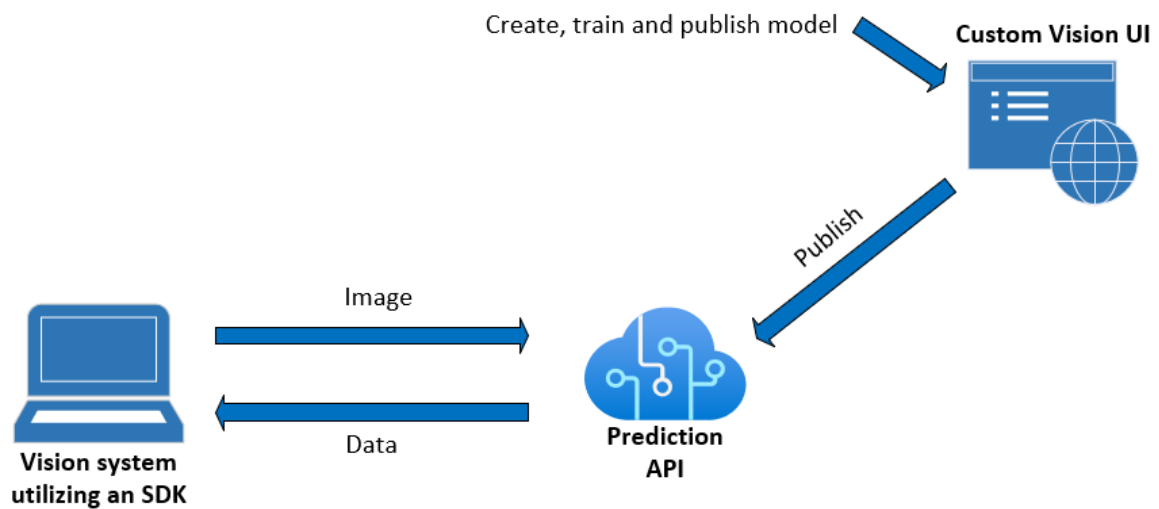


Figure 6-2: The vision system sends an image of an 8-ball game as a request to the prediction-API and receives data, that includes the names of billiard balls and the corresponding image-coordinates.

### 6.2.1 Technology behind Custom Vision

In the section above, Custom Vision was described as a service that offered prebuilt machine learning models. But more specifically, it is mainly deep learning models that it offers. Deep learning is a subset of machine learning and machine learning is a subset of artificial intelligence. So before defining deep learning, let's define what machine learning and artificial intelligence are. Artificial intelligence allows machines to learn, sense, reason and adapt, just like ordinary humans do. On the other hand, machine learning is the method that enables machines to learn without being programmed through access to data. Deep learning is a method used within machine learning that utilize neural networks with vast amount of data [13]. Figure 6-3 illustrates the relationship between artificial intelligence, machine learning and deep learning.

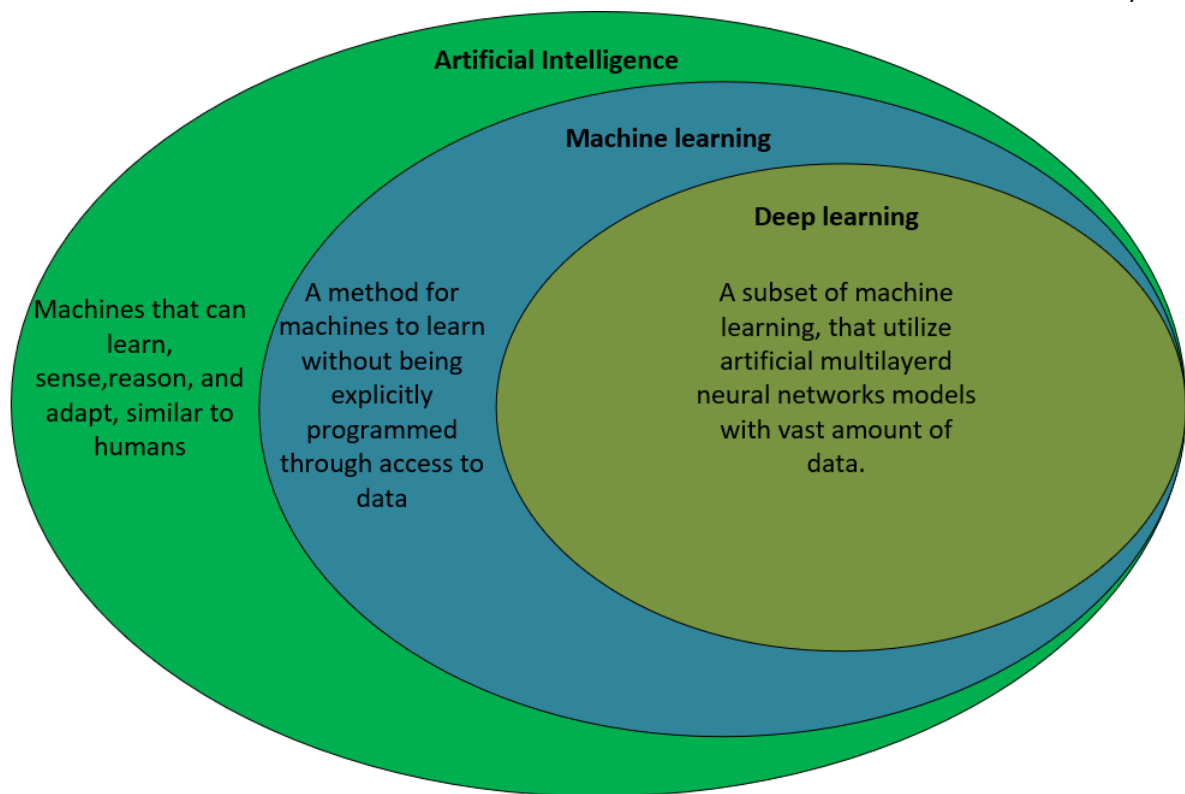


Figure 6-3: Relationship between artificial intelligence, machine learning, and deep learning

A common approach to solving the problem of object detection and classification, especially in recent years, is through deep learning methods. Deep learning methods use multilayered neural network models to solve the problems specified above. It achieves this by processing the raw input, for example pixels, through a series of functions. These functions are basically the “neurons” in the network. Multilayered neural networks are usually comprised of dozens of these neurons. The neurons are also organized in layers, and the networks will usually include dozens of layers, hence the terminology “deep”. Each neuron will contain several parameters, and modern neural networks will contain millions of parameters. The idea behind these networks, is to find the optimal parameters of the functions, so that eventually the model can correctly predict the input data. The parameter-value adjustments are achieved through an examination of a large amount of data, and gradually correcting itself as it compares the predicted result with the actual input data [13].

Custom Vision makes use of a deep learning technique called transfer learning. The concept behind transfer learning is the utility of knowledge gained from solving one problem, to solve a different but related problem. This can greatly decrease the time and data needed for creating the models and consequently jump start the process of solving a deep learning problem. It achieves this by utilizing pre-trained multilayered neural networks. For example, a multilayered neural network can be trained on the large dataset of images, with millions of examples. This network will then have gained the knowledge of how to process images well. This will entail general knowledge such as how to detect edges, shapes, and patterns to differentiate between objects. This knowledge, captured within the parameters of the network, can be used in different kind of scenarios with significantly less data, such as distinguishing between a set

of billiard balls [13]. The latter problem is the focus of this report, and it is through Custom Vision’s transfer learning abilities that it will be solved.

### 6.2.2 Custom Vision UI

The Custom Vision UI is a web application that is used to create, train, and publish a model, that can detect and classify billiard balls. This subchapter will give a closer look into how the model is created, how the model is trained and how the model is published, using the Custom Vision UI. Figure 6-4 shows the main page for the Custom Vision UI.

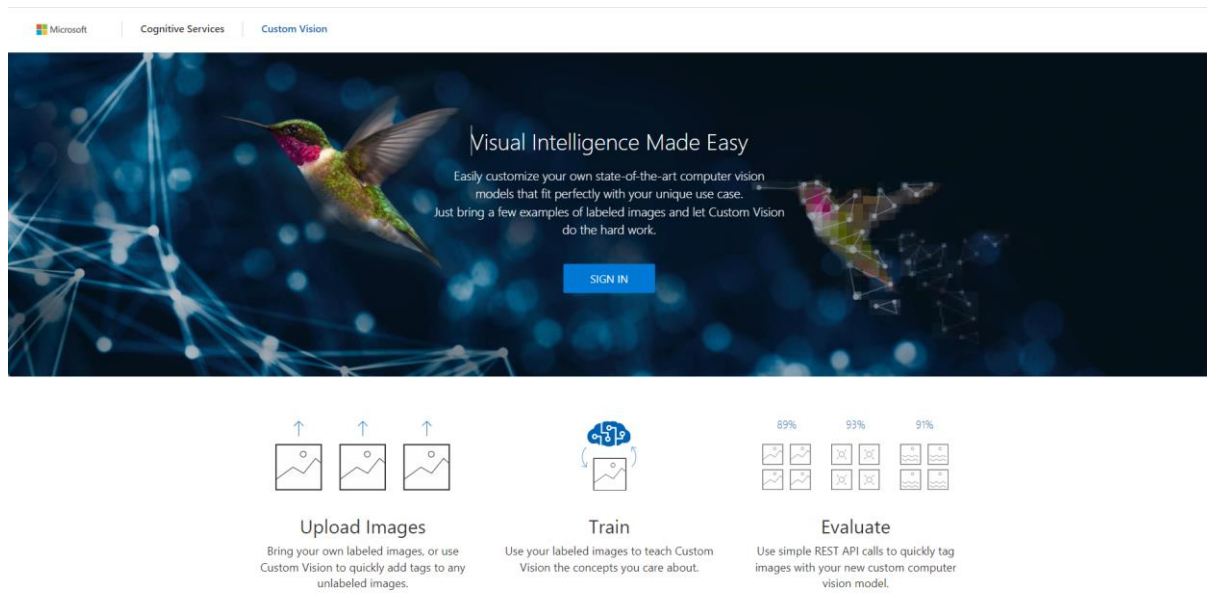


Figure 6-4: Main page for the Custom Vision UI

To create the model, Custom Vision UI presents the page shown in Figure 6-5. This page consists of a form intended for creating a model. The top-half of the form includes a textbox for what the model is called, a textbox for a description of the model and a combo box for selecting resources. The bottom half includes checkboxes for which function is wanted for the project. The choice is between classification and object detection. The main difference between these two options, is that the classification function has the ability of applying labels to objects, within an image. While the object detection function has the same ability as the classification function, it will also generate a bounding box that surrounds the object. Therefore, “Object Detection” will, in this instance, encompass detection and classification. Because of this, the “Object Detection” alternative is chosen for this model.

Lastly, the bottom half of the form will also include checkboxes intended for choosing a domain. These domains optimize the model to detect and classify certain types of datasets. For example, the “Logo” domain optimizes the model to detect and classify images that contain logos. As Figure 6-5 demonstrates, the “General [A1]” domain is chosen in this project. The documentation for Custom Vision suggests that the “General [A1]” domain is optimized for better accuracy with comparable inference time as General Domain. Furthermore, it is recommended for larger datasets or more difficult user scenarios. It also requires more training time. The choice was based on the group’s assessment on which domain fitted this specific scenario:

Detecting and classifying billiard balls [15]. Figure 6-5 shows the implemented information for this model.

Create new project

Name\*

Ball detection

Description

Detecting and classifying billiard balls

Resource\* [create new](#)

Visionoppgave [S0]

[Manage Resource Permissions](#)

Project Types ⓘ

☐ Classification

☒ Object Detection

Domains:

☒ General [A1]

☐ General

☐ Logo

☐ Products on Shelves

☐ General (compact) [S1]

☐ General (compact)

Pick the domain closest to your scenario. Compact domains are lightweight models that can be exported to iOS/Android and other platforms. [Learn More](#)

Cancel Create project

Figure 6-5: Page for creating a model on the Custom Vision UI and subsequently, the page also shows the implemented information for this particular project.

Now that the model is created, the next step is to train the model. There are two steps needed to train the model, to detect and classify billiard balls. The first step is to capture images of billiard balls and import them to the Custom Vision UI. Figure 6-6 shows some of these uploaded images. The second step is to specify the objects that are going to be detected and classified, within the images. This means that a bounding box, surrounding the relevant object,



## 6 Vision system

must be carved out. This can either be done manually, or Custom Vision can generate these bounding boxes by itself. Each bounding box must also coincide with a tag. A tag is the name of the object that is surrounded by the bounding box. Figure 6-7 shows an image, where all the billiard-balls within the image have been tagged. Meaning, they have a bounding box surrounding them, as well as the corresponding tag names.

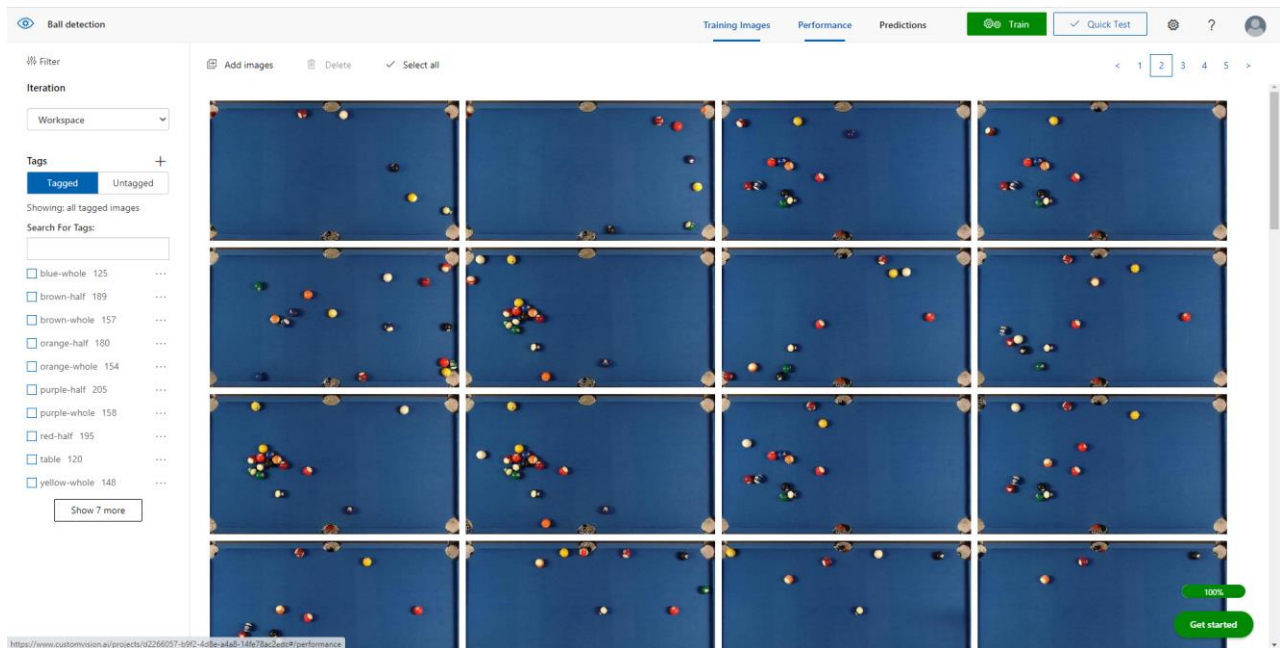


Figure 6-6: Images of billiard-balls on a pool table, uploaded to Custom Vision.

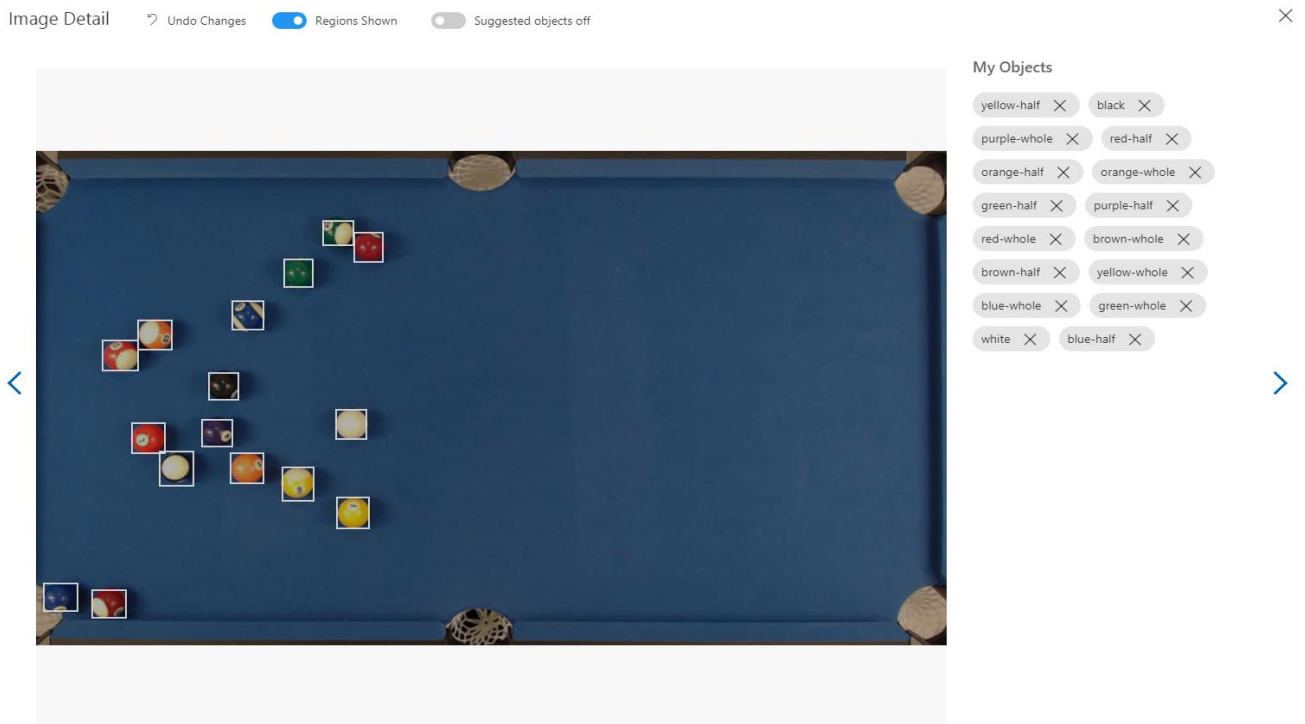


Figure 6-7: Image of Billiard-balls that have been tagged. The white rectangles surrounding the balls are the bounding boxes and the corresponding tag-names are listed on the right side of the page.

In total, there were 395 images of billiard balls, uploaded to the Custom Vision UI for training. These images varied in their visual characteristics. These variations entailed ball-positions, lighting, camera angles and size. The documentation for Custom Vision recommends these varieties to the images, such that the model can be trained more effectively. The documentation also recommends that uploading 50 images per tag, is generally considered a good number of images to start with. For example, 50 images can be uploaded where the tag “yellow-half” is included in all of them. Table 6-2 shows the final number of images per tag that were uploaded and used to train the Custom Vision UI [12].

Table 6-2: Number of images per tag

Tag	Number of images
Blue-whole	123
Blue-half	122
Yellow-half	111
Yellow-whole	234
Orange-whole	234
Orange-half	145

<b>Purple-whole</b>	121
<b>Purple-half</b>	112
<b>Green-whole</b>	145
<b>Green-half</b>	165
<b>Red-whole</b>	178
<b>Red-half</b>	154
<b>Brown-whole</b>	134
<b>Brown-half</b>	165
<b>White</b>	323
<b>Black</b>	234

Once alle the images have been tagged, the execution of training the model can proceed. The Custom Vision UI will put forward two alternatives of training the model. These alternatives are displayed in Figure 6-8 and are based on what kind of subscription that is being used. There is a free subscription and a standard substruction. Advanced training is available with the stand-ard subscription and custom vision defines it as the ideal alternative for challenging datasets and for improved model-performance. Furthermore, as Figure 6-8 illustrates, advanced training also allows the user to specify the duration of the training. This can be anywhere from 1 to 96 hours. This alternative does come at a cost and the amount of the cost depends on the number of hours the training duration is set to. It costs 10 USD per hour to train the model. Quick training is available with the free subscription and Custom Vision describes this alternative as ideal for quick training-durations and does not offer the user to specify the training-duration [16]. Table 6-3 shows the final count for the model's training sessions. As Table 6-3 illustrates, both quick training and advanced training were utilized. For a more detailed overview of the training-sessions, see Appendix C.

Choose Training Type ×

Training Types ⓘ

☐ Quick Training
   
☒ Advanced Training

In most cases, the more time you select the better the model will be. You're charged based on the compute time used to train your model, so choose your budget based on your need.

Training budget: 1 hour ⓘ
   
 1 hour | | | 96 hours

☐ Send me an email notification after training completes
   
 Email address

Train

Figure 6-8: Pop-up page showing the two alternatives of training the model.

Table 6-3: General overview of training sessions

	Quick training	Advanced training
Number of sessions	8	2
Average training-duration in hours	1.1	3

After a training session is done, the publishment of the model can take place. This is achieved by publishing a prediction-API. This is done by using the Custom Vision UI. It is required to specify the model's name and the resource that is used. The model-name will later appear as one of the parameters needed to interact with the prediction-API, from the vision system. Figure 6-9 shows the model's name as "BallModel10". The reasoning behind the number at the end, is to specify which iteration of the model is being published. This process of uploading training-images, training and publishing the model is repeated continuously, until the model works sufficiently.

## Publish Model



We only support publishing to a prediction resource in the same region as the training resource the project resides in.

Please check if you have a prediction resource and if the prediction resource is in the same region as the training resource.

Model name

BallModel10

Prediction resource

Visionoppgave-Prediction

Publish

Cancel

Figure 6-9: Pop-up page for publishing a model

After every training session, three metrics are going to be shown in the Custom Vision UI. These metrics can be seen in Figure 6-10. The metrics represent an evaluation of how well the model is performing with the training images. More specifically, how well the model is detecting the various billiard balls within the training images. The metrics are titled as precision, recall, and mAP, or mean average precision.

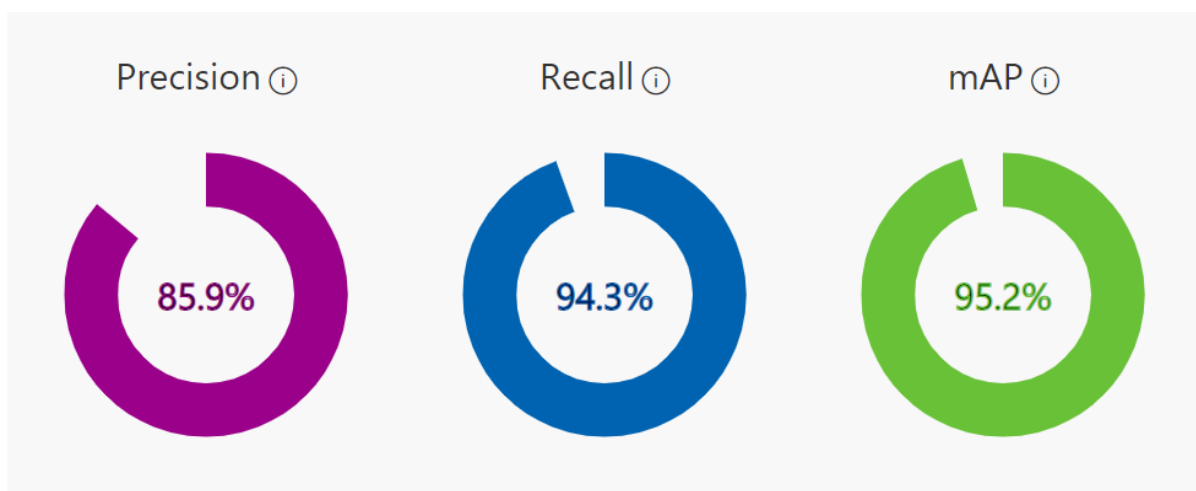


Figure 6-10: Example of how the Custom Vision UI shows the metrics that represent the performance of the model

Before defining precision, recall or the mean average precision, there needs to be a definition for what the model considers a correct detection. This can be achieved by using a method called intersection over union. The basic idea behind using this method, is that producing a correct detection does not require the predicted detection to 100% match the ground truth. This means that the predicted bounding box is not required to completely match the bounding box that was used in the training images. Instead, an IOU-threshold can be set.  $IOU = \frac{D \cap G}{D \cup G} * 100$  (6-1) shows the equation that is used to calculate the IOU. This equation is used to compare the two bounding boxes: the predicted bounding box produced by the model, which is defined as “D” in the equation, and the bounding box that was used in the training images, which is defined as “G” in the equation. If the result from this equation is over a predefined value, then the predicted detection is defined as correct. Otherwise, it will be defined as incorrect [17]. The IOU threshold in this project was set to 30%. This specific threshold was the default threshold that Custom Vision sets and is only used when training the model.

$$IOU = \frac{D \cap G}{D \cup G} * 100 \quad (6-1)$$

The precision indicates the ratio between the number of detected balls that were correctly predicted, to the total number of detected balls within an image. For example, if the model identified a ball in 100 images as “yellow-half”, and 99 of them were actually “yellow-half” balls, then the precision would be 99% for the “yellow-half” ball. (6-2) shows the equation that is used to calculate the precision for one type of billiard ball. The precision shown in Figure 6-10 and Figure 6-11 is the average of the precision from all the billiard balls [18].

$$Precision = \frac{\sum \text{Detected \& correctly predicted balls}}{\sum \text{Number of detected balls}} * 100 \quad (6-2)$$

The recall indicates the ratio between the number of detected objects, that were correctly predicted, to the total number of objects within an image. For example, if there were actually 100 images of “red-whole” balls, and the model identified 80 as “red-whole” balls, the recall would be 80% for the “red-whole” ball. (6-3) shows the equation that is used to find the recall for one type of billiard ball. The recall shown in Figure 6-10 and Figure 6-11 is the average of the recall from all the billiard balls [18].

$$Recall = \frac{\sum \text{Detected \& correctly predicted balls}}{\sum \text{Number of balls}} * 100 \quad (6-3)$$

The mean average precision is a metric that encompasses, both precision and recall. It is defined as the mean, of the average precision of each ball. Average precision is the area under the precision and recall curve. This curve is generated, when a ball’s precision measurements is plotted against the recall measurements of the same ball, for each prediction the model makes

across various probability scores. Probability scores are indicators of how confident the model is in its predictions. (6-4) shows the equation that is used to find the mean average precision. The mean average precision is usually used to compare object detection models [17] [18].

$$mAP = \frac{\sum_{i=1}^N AP_i}{N} * 100 \quad \text{Where } N \text{ is the total number of balls and } AP \text{ is Average precision} \quad (6-4)$$

The final result of the model's performance-metrics is illustrated in Figure 6-11. These metrics were achieved through the training sessions shown in Table 6-3 using the training-images detailed in Table 6-2 and an IOU threshold of 30%.

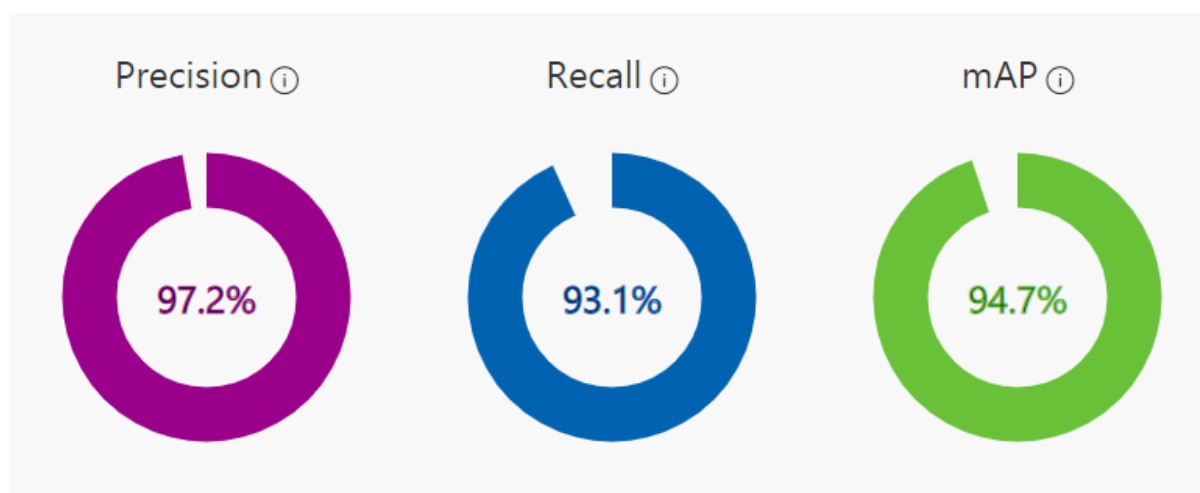


Figure 6-11: Final performance-metrics of the model generated by the Custom Vision UI

### 6.2.3 Prediction API

The prediction-API allows the vision system to interact with the model and make predictions. Figure 6-12 shows an illustration of how the vision system interacts with the prediction-API. There are three alternatives to making a prediction using Custom Vision: The first one is by using the Custom Vision UI, the second is by using an SDK, and the third is by using a REST API endpoint. The method that is used in this report, is the one that involves an SDK. SDK stands for Software Development Kit. It provides developers with the necessary tools to operate various tasks, in an installable package. The SDK that is in use in this project, is installed in Visual Studio by using the package manager “NuGet”. The SDK is called “Microsoft.Azure.CognitiveServices.Vision.CustomVision.Prediction”. It gives access to classes and methods, that can be used to interact with the prediction-API. The “BallDetection” class, illustrated in Appendix F, is the class that is responsible for executing the process of making a prediction. The description below, describes how the SDK is used, within the “BallDetection” class, to

communicate with the prediction-API. More on the C# classes, including the “BallDetection” class that build up the vision system application will be detailed in chapter 6.3.

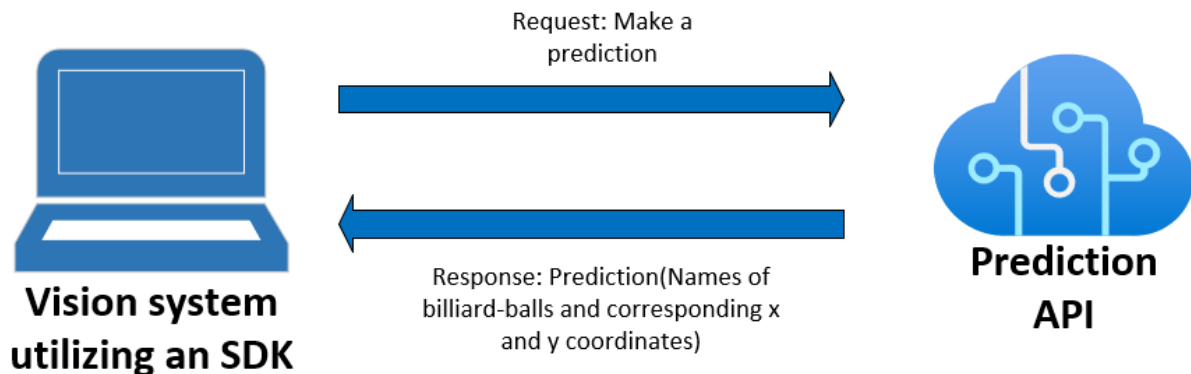


Figure 6-12: Overview of the vision system’s interaction with the prediction-API

The first step of the process is to specify some important parameters. These parameters are the prediction-endpoint and the prediction-key. They can be fetched from the Customs Vision UI. These parameters are then used as input parameters in a method called “PredictionConnection”. The latter method is part of the installed SDK. This method’s responsibility is to establish a connection to the prediction-API and will only be executed at program start-up. If the connection is successful, then the next step of the process can begin. If not, an error will be displayed.

In the next step, some additional input-parameters need to be specified. These parameters are an image of billiard-balls, the project-ID, and the model-name. Both the project-ID and the model-name are fetched from the Custom Vision UI, while the image will be sent from the “GameManager” class or the “Simulation” class. These classes are illustrated in Appendix F. The parameters will then be used as input-parameters in a method called “DetectImage”. This method’s responsibility is to request a prediction from the prediction-API. If the request is processed correctly, the API will send back the result of the prediction. If not, an error will be displayed.

The results of a prediction include the names of detected billiard balls, the x and y coordinates of the corresponding billiard balls, and probability values of the corresponding billiard balls. The x and y coordinate are the coordinate of the upper-left corner of the detected bounding box. This is shown in Figure 6-14. The probability score shows the level of confidence the model has in its predictions. The results that “DetectImage” generates are not filtered to only get the predictions that were made over a certain probability threshold. Therefore, the result will include all the predictions that the model makes. The only predictions that are relevant to retrieve in this case, are the correct predictions. To achieve this, there is implemented code that filters out predictions that have a lower probability value than a certain probability threshold. This threshold is set to 40%. This is because a 40% threshold provides the optimal result. This number was found through trial and error. The final result will include the names of the detected billiard balls and their x and y coordinates. After this data has been processed, the class will wait for a new image. If a new image is sent, the “DetectImage” method will be executed again. Figure 6-13 illustrates the process of making a prediction using the vision system.

The number of predictions that a user is allowed to make will be based on the type of subscription that is used. With the free subscription, users would be able to make 10,000 predictions



per month. But with the standard subscription, users would pay 2 USD for every 1000 prediction they make, without having any kind of limit [16]. In this project, the free subscription was used to make predictions.

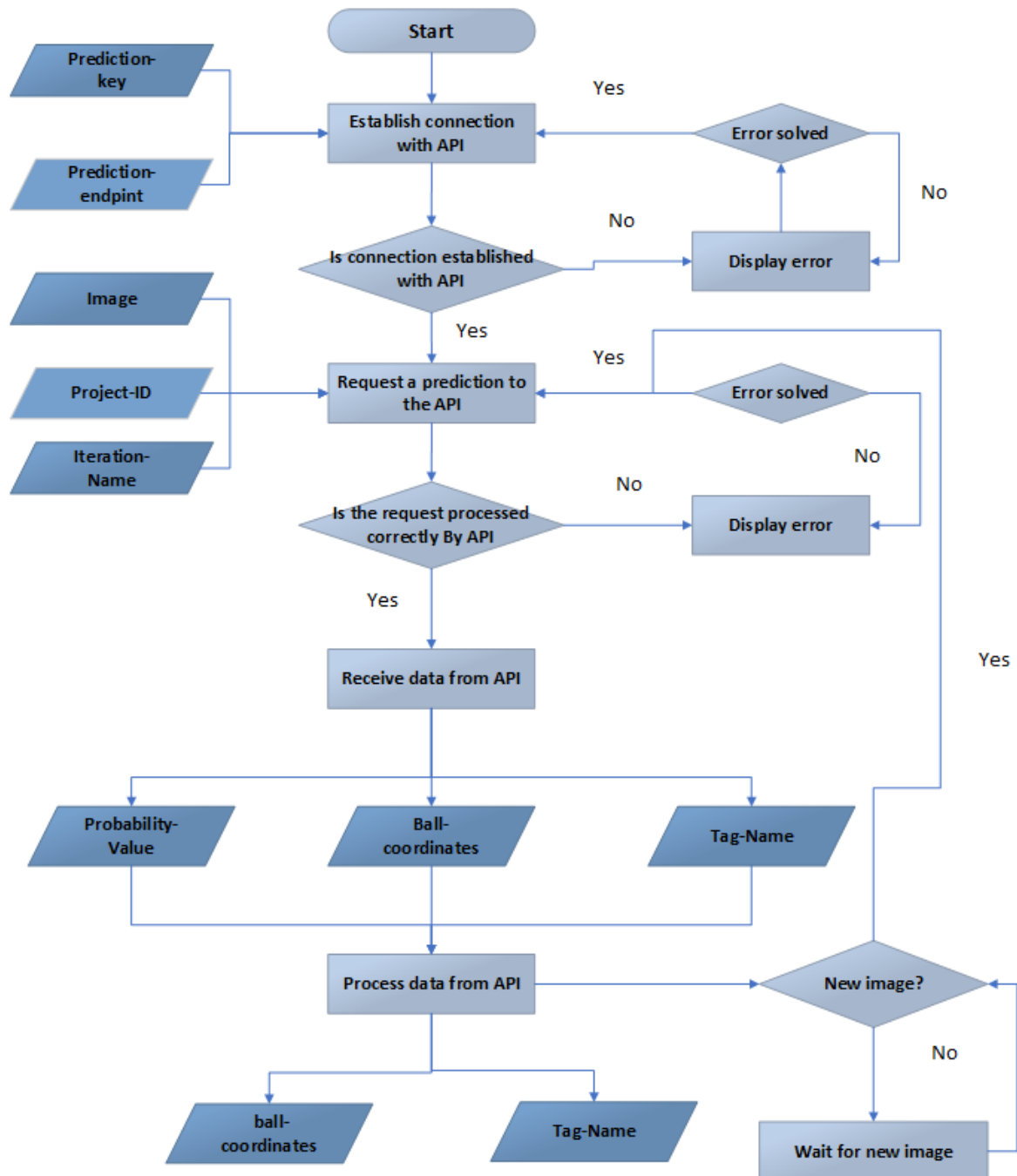


Figure 6-13: Flowchart showing the process of making a prediction

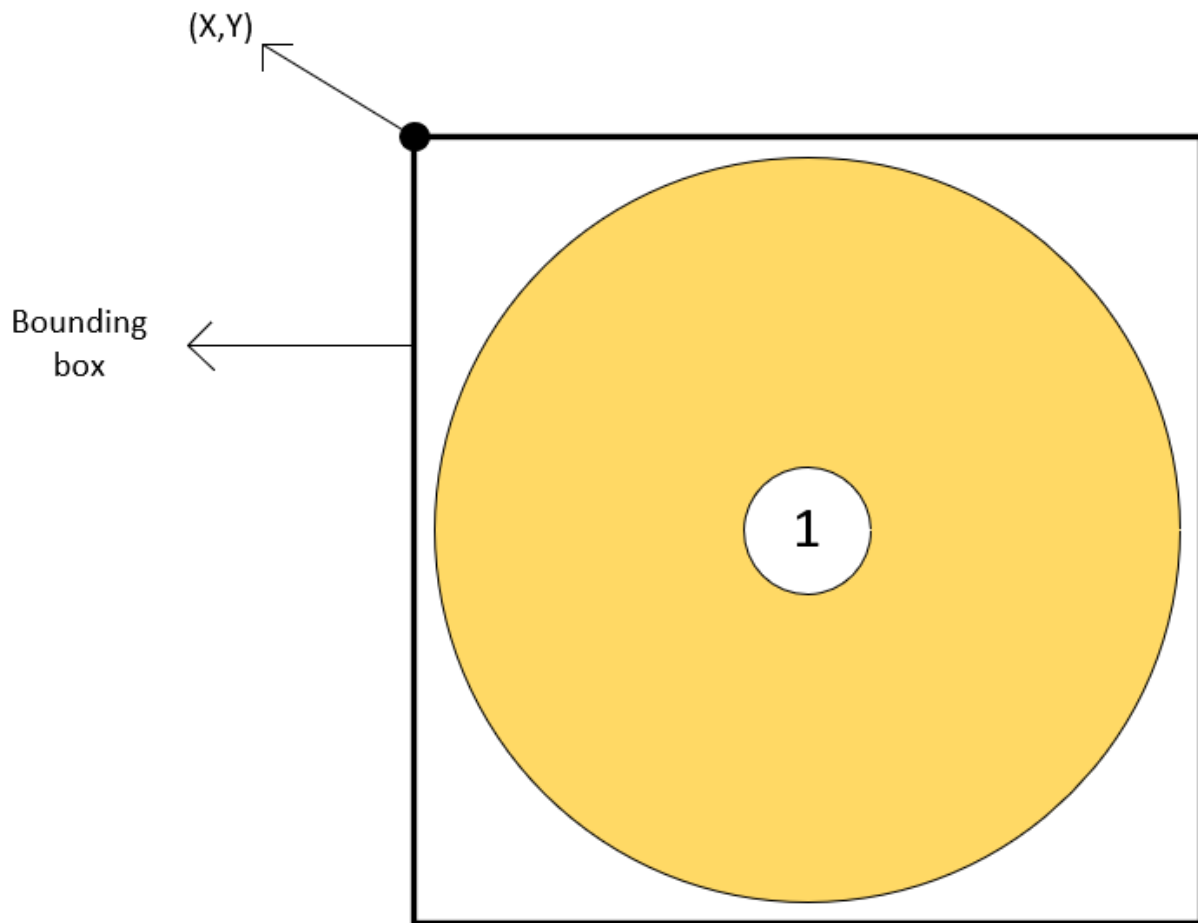


Figure 6-14: Illustration of a detected ball with its bounding box

### 6.3 Class structure

The vision system is responsible for monitoring a game of 8-ball. It must be able to receive live video feed and images of a game, from the camera. Furthermore, it must process the images by utilizing Custom Vision's prediction-API. It must also receive data from the prediction-API, interpret the game by applying the game rules, and store and retrieve information about the game in the database. It is also required to transfer the live video feed to the GUI and wait for instructions from the GUI. Lastly, it must also be able to communicate with the web application through the communication-API. **Error! Reference source not found.** shows an overview of the tasks described above. To achieve these tasks, the vision system is developed as a Win-Forms application in Visual Studio. The vision system is mainly constructed by various C# classes, and it is through these classes that the tasks described above get executed.

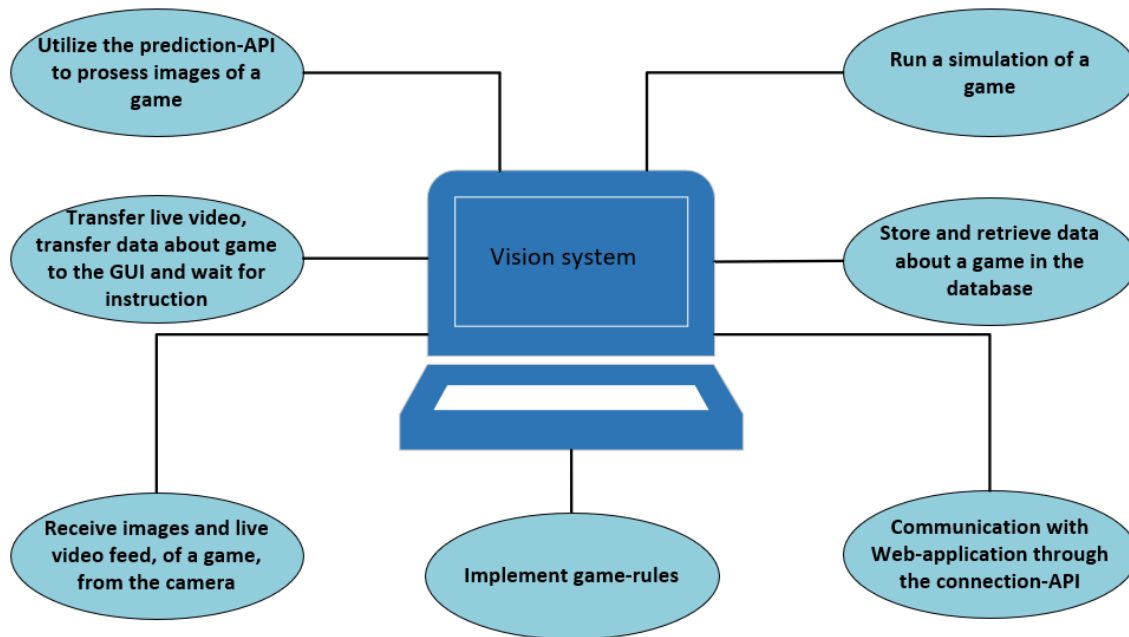


Figure 6-15 : Overview of vision system's tasks

The vision system includes 4 concrete classes and 3 Form classes. The 4 concrete classes are called “Ball”, “Player”, “Game” and “BallDetection”. The Form classes are called “GameManager”, “Simulation” and “StartPage”. Regarding the naming conventions in the program, pascal notation is used for naming classes, methods, and properties. Camel case is used when naming variables. Figure 6-16 illustrates how the various classes are constructed to execute the tasks at hand. Appendix F also shows the class diagrams of the concrete classes.

These classes contain methods, properties, and variables that in combination can execute tasks. The “Ball” and “Player” classes will mainly include properties that act as placeholders for relevant data. The “Game” class will also include some properties, but it will also include logic-based methods that implement the game-rules. Additionally, this class will be used to store and retrieve data from the database. The library “Npgsql” is used to make the interaction between the class and the database possible. “Npgsql” contains various methods and classes that can be used to store or retrieve data. The “BallDetection” class will include various methods and variables. The main purpose of this class is to generate the coordinates and names of the detected billiard balls. A detailed description of how this is achieved by the utilization of the prediction-API, is given in chapter 6.2.3.

The “GameManager” class will take advantage of the concrete classes by using objects that refer to the concrete classes. Meaning that the methods and properties contained by the concrete classes will be called from the “GameManager” class. The “GameManager” class will also include other methods. These methods will be responsible for different tasks, which are as follows: running an interface with the camera, communicating with the connection-API and running an interface with the GUI. The library “AForge” is used for handling the interface with the camera. The library “System.Net.Http” is used to run the interface with the connection-API. There is no need for a library to interact with the GUI. This is because they exist within

the same environment, Visual Studio WinForms. The “Simulation” class is almost identical to the “GameManager” class in relation to content and structure. The only difference is that the “Simulation” class doesn’t include methods that run an interface with the camera. Instead, it will use predefined images taken during a game to showcase the system. Lastly, the “StartPage” class will include methods that initialize communication with the communication-API, as well as methods that send and receive data from the connection-API.

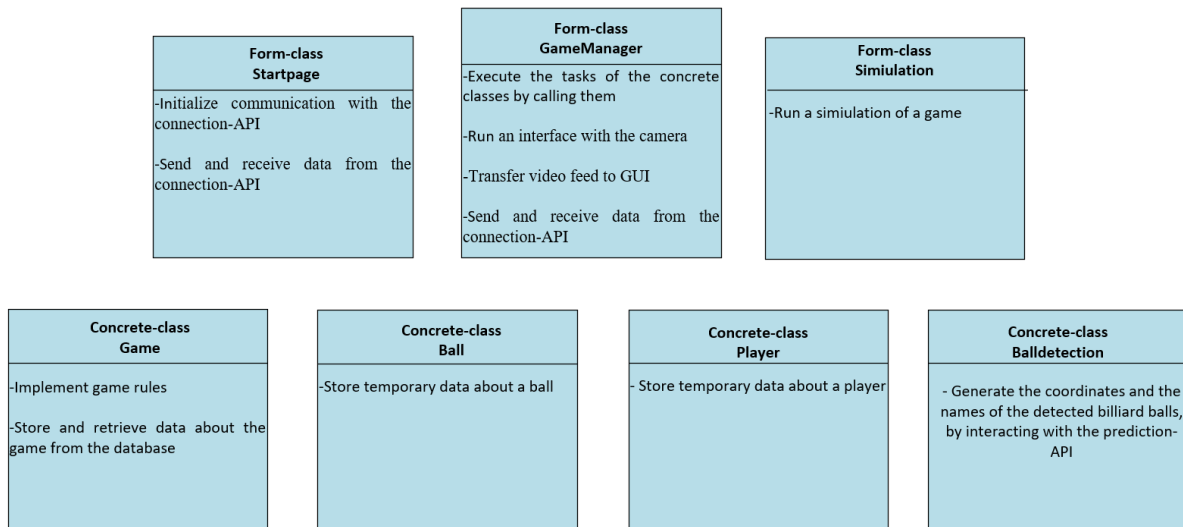


Figure 6-16: Illustration of the classes that execute the various tasks for the vision system

## 6.4 Pool rules

The game of 8-ball is a game that is associated with a lot of lighthearted fun, but the game involves a very competitive nature as well. Most people know the basic rules of the game, however, diving deeper into the documentation of the competitive nature of 8-ball, there are a lot more rules that apply than one might first think.

A healthy combination of the rules, combined with convenience to make a well-functioning system, as well as a limited time and difficulty finding a good way to implement some of the more advanced rules, an adjusted set of rules had to be implemented. In this project, some of the rules have been made simpler. Some rules have been cut out completely due to difficulty seeing how to make them work on a consistent basis in a digital system only reviewing pictures, such as players hitting the other players ball first resulting in a foul. The most notable however, might be the rule that states a player should call the pocket they intend to put the 8-ball, or potting it in the hole directly opposite of where their last ball was potted. This rule is deemed not necessary, as it is hard to track and judge for the system. Other rules have not been touched and are implemented in the system to make the game flow and feel realistic, even if the full set of rules is not present in the current system.

Starting the game, the 15 balls are racked in a triangle shape. The 8-ball should be placed in the middle, but the players can decide on the order of the balls. For simplicity reasons, the players decide who plays which set of balls before initializing the game, unlike regular pool where the players break and pocket the set of balls they desire to continue playing with.

Pocketing the other player's balls and failing to pocket one of their own balls will result in the player losing their turn. Pocketing the white ball at any point will result in the cue being handed over to the other player, even if they have pocketed one of their own balls. Having pocketed all their balls, the black ball is the last ball to be pocketed. The player that first successfully pockets all their balls and then the black ball at the end, wins the game. Pocketing the black ball at any point without having pocketed all the balls of their set will result in the game being lost. If the player by any chance pockets all their balls, then finishes off by pocketing the black and white ball at the same time, they will lose the game, as it is considered a foul. In Table 6-4 below, one can see an overview of the most common rules that have been implemented, adjusted, or left out of the project completely.

Table 6-4 Overview of the most common 8-ball rules [19]

Rule	Implemented	Adjusted	Comment
<b>Coin toss to decide who breaks</b>	✓		Player 1 breaks, but who's set up as Player 1 can be decided with a coin toss
<b>Legal break requires a ball to be potted or four balls to hit the cushion</b>	X	Adjusted for simplicity reasons	As long as the player hits the balls, it will count as a legal break
<b>The first player to pot a ball of any sort will have that sort assigned to them</b>	X	Adjusted for simplicity reasons	The players choose which set of balls they want before the game starts by deciding who is player 1 (solid), and who is player 2 (half)
<b>A player continues to play if they pot balls of their designated sort</b>	✓		
<b>Call the pocket and sink the 8-ball after all balls of their sort have been pocketed to win the game</b>	✓	Adjusted for simplicity reasons	The 8-ball can be pocketed in any pocket, as it is difficult to track which pocket was called, and if the 8-ball hit that pocket. All balls tied with that player must be potted before sinking the 8-ball to win
<b>Failing to hit their own set of balls</b>	X		Difficult to track if the right balls were hit, so this is left out

<b>Hitting the cue ball off the table</b>	✓		If the white ball is not present on the table, the turn is handed to the other player
<b>Potting the opposition's balls</b>	✓	Slightly adjusted	Potting the opposition's ball while also potting one of their own will not result in a foul
<b>Hitting the cue ball twice, or pushing the cue ball</b>	X		Difficult for the system to track. For fair play, this can be judged by the players themselves
<b>Pocketing the 8-ball and the white ball at the same time</b>	✓		Pocketing the black and white ball at the same time will result in a loss

#### 6.4.1 Main methods to enforce the rules as a digital judge

After setting up the game, either starting a quick game or going through the web app, names and ball type should be ready. The next step, and the main task for the vision system, is to make sure the game goes through, and that the game rules are enforced in the right way. These following methods, which are located in the “GameManager” class, could be considered the backbone of the program.

- **Snapshot()**  
The snapshot method is what sets off the whole program. This method captures an image from the video feed and sends it to the “BallDetection” class to be processed.
- **ShowBalls()**  
This method receives a list of detected billiard-balls. After receiving the list of balls that are detected, this method goes through the list and transfers this data to the GUI.
- **CheckBalls()**  
Using the same list of balls, this method checks and counts the amount of solid and half balls in the list. If the number of solid or half balls potted are 7, it means the player can challenge the black ball, displaying this information in the GUI by making the 8-ball visible in the display of balls left to sink.
- **CheckWhite()**  
The name of this method speaks for itself. CheckWhite does exactly that, checks if the white ball is in the list. If the white ball is not on the table, it means it has been pocketed by one of the players. This will overrule all other things in regard to whose turn it is, and hand the cue over to the other player.

- **CheckBlack()**

This method works closely with the two previous methods. Similar to the method that checks the white ball, this method checks the black ball. If the black ball is no longer on the table, it will go through a couple of checks to see whose turn it was, if all balls were potted before potting the black, and if the white ball is still on the table. If all these requirements are met, the player sinking the black ball wins. However, if one of the other requirements are not fulfilled, the player in question consequently loses the game.

- **CheckResult()**

After receiving the data from the above methods, it will be clear if the black ball is potted, and this method will display the winner, as well as uploading the information to the database. A timer will also start in order to auto close the main application and head back to the start page to start another game.

- **TurnLogic()**

If no winners are declared, this method will check if any balls were potted, and if they were the right sort of balls, as well as if the white ball is potted. If the right balls were potted, the player will continue their turn. Otherwise, the cue will be handed over to the other player. The “Game” class will be utilized to execute this task.

## 6.5 Graphical user interface of the vision system

The graphical user interface of the vision system is created to provide the user with a graphical interface, with the aim of using the vision system. The following subchapters will dive deeper into the graphical interface that is developed in tandem with the vision system. Figure 6-17 shows the interaction between the GUI and the vision system. It will touch on the planning phase, including the illustrations that started as an idea and became the final pieces that can be seen in the application, as well as challenges, decisions, and how the result have turned out. The GUI will be displayed on the TV, is mentioned in chapter 6.6.

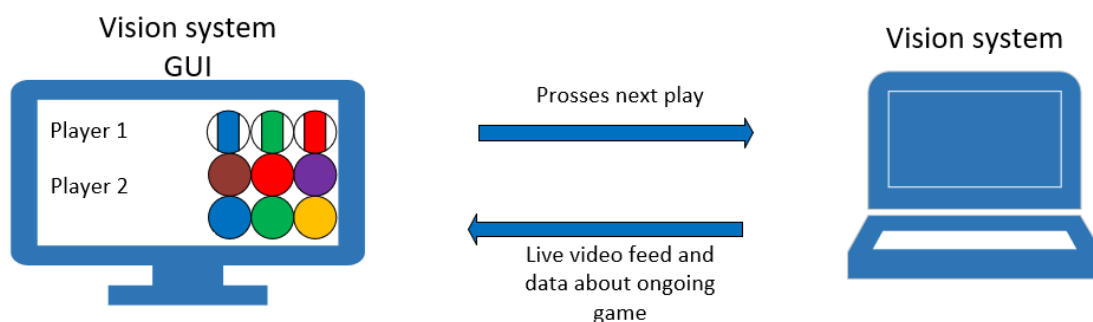


Figure 6-17: Illustration that shows the interaction between the GUI and the vision system

Working closely with the vision system, a graphical interface must be developed in order to show the results of the 8-ball games. Without the interface, the users would not be able to determine how the game is going, or even instruct the vision system to start processing the next play. The vision system will transfer data about the state of an ongoing game to the GUI. This

data includes whose turn it is to play and the number of balls each player has left to pocket. The GUI will also show a live video feed, that the vision system provides.

Development of the GUI was done in Visual Studio and is a WinForms application. The program code is written in C# and takes advantage of the form classes and objects to achieve a simple, yet powerful interface. A user manual will be available in Appendix D.

In the planning process, there is a need for a rough design to lay out the basics when it comes to look and functionality. The group decided to go with a modern and clean design, as the application should display vital information about the game and should therefore be easy to read for both players and spectators. Designed by the group, the artwork is specially made for this app with a simple but effective design in mind. Some of the illustrations will also be used in the final design. All designs and illustrations used in this chapter have been made in Adobe Photoshop.

### 6.5.1 Ball design

Following the need for illustrations in both the design and in the GUI itself, a set of balls were drawn. The main purpose of this ball grid shown in Figure 6-18 below, is to display which balls remain on the table. The grid consists of 15 balls, seven solid balls, seven half balls, and one 8-ball, numbered 1 to 15 like how they would look in real life. The colours of the balls have been adjusted slightly to achieve better visibility and differentiation. The base colours are still the same, however, the orange and blue balls have been slightly altered to stand out more from purple and yellow.



Figure 6-18: Ball design for vision system



### 6.5.2 Cue design

In order to keep track of whose turn it is to hit the balls, a simple cue was designed, see Figure 6-19. The idea is that the player who is currently playing, has the cue next to their player-name. As soon as they miss, or otherwise do something to lose their turn, the cue is handed over to the other player.



Figure 6-19: Design of cue to indicate whose turn it is

### 6.5.3 Game page design

Using the drawings made earlier, a simple design was made. The pool table displayed should, according to the planned design, be a live video showing the current game. In Figure 6-20 below, one can see a rough sketch of the GUI. By quickly scanning the design, the reader of this report can see there are two players, who can each be given a chosen name. In this design, for simplicity, they're named Player 1 and Player 2. Player 1 in this case have three solid balls left to put in the holes, while Player 2 has four half balls left, before they can challenge the black ball. The cue is displayed next to Player 2, indicating that Player 2 currently has the cue and is up next.

Central by the lower border of the snapshot, a timer is added to display how long the current game have been going for. This particular game has lasted for 8 minutes and 36 seconds.

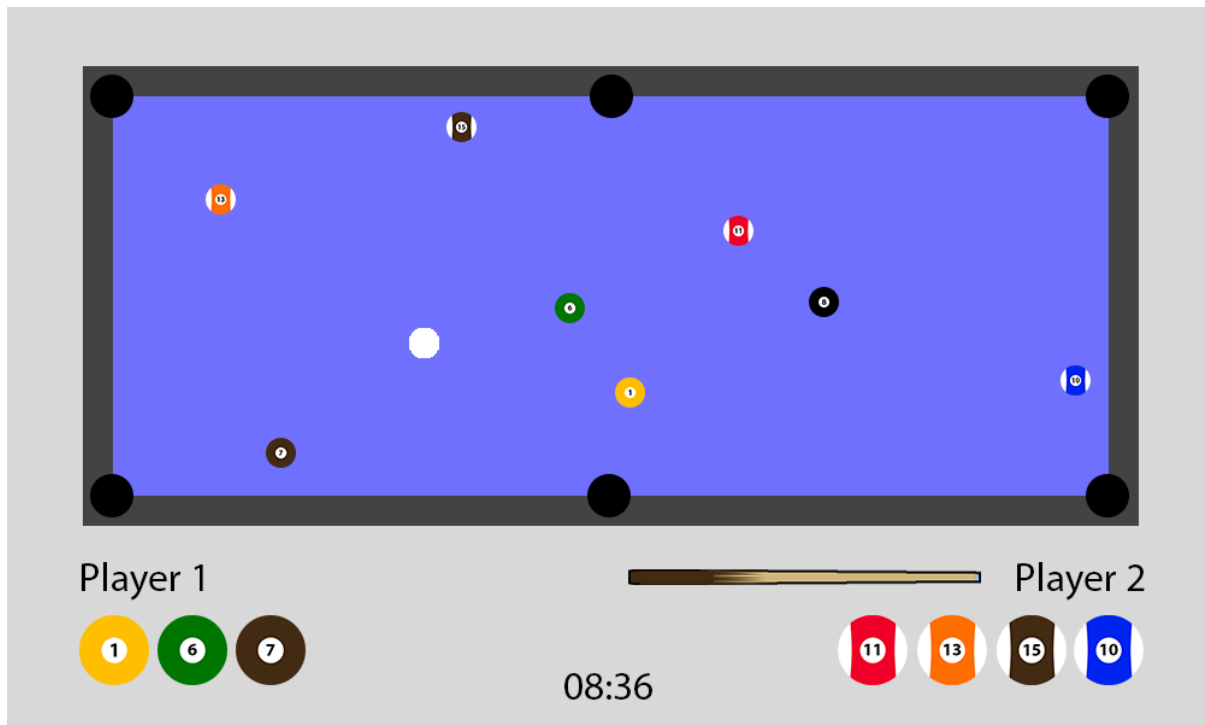


Figure 6-20: Proposed design of desktop application

## 6.6 Final design

Opening the application, the first thing the user will face is the start page. The start page is designed to give a clean and fresh interface with user friendliness and a visual good look heavily in mind. The colours in the interface are based on the colours presented in the logo, and matches the rest of the application interfaces, as well as the design of the webpage developed together with the rest of the system. The final design of the start page can be seen in Figure 6-21 below.

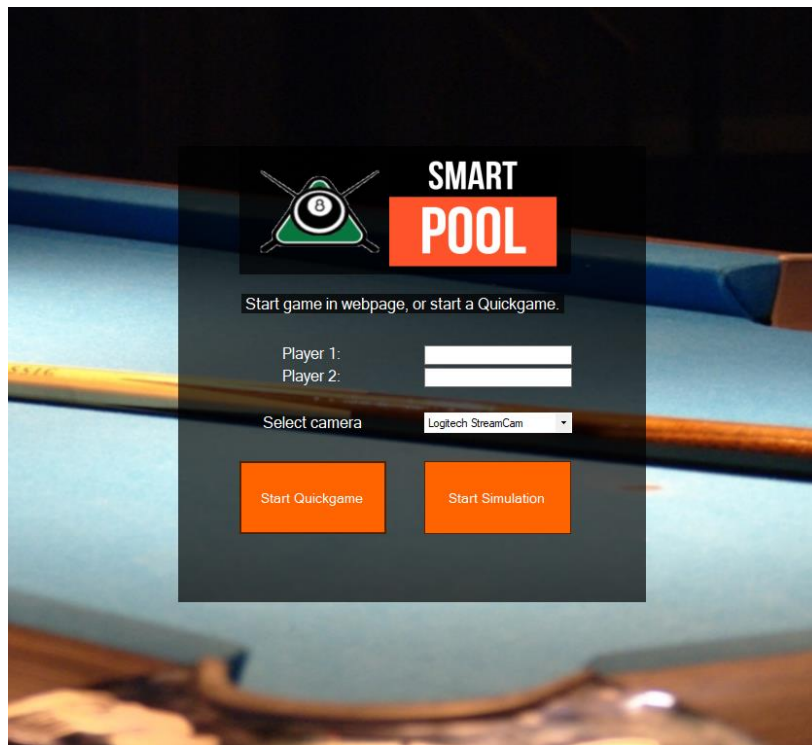


Figure 6-21: Start page design

Comparing the final design in Figure 6-22 to the idea of a design that was proposed in the early stages of the project, see Figure 6-20, it looks very similar. Throughout development it received a lot of attention and looks even better than the proposed design. The background image and the faded panels give it a more modern look. The design is made up of five main components and is supposed to display information clearly to the players and spectators. These five components consist of the player-name input, the “Select camera” dropdown menu, the “Start Quickgame” and “Start Simulation” buttons, and the “Camera settings” button at the bottom. Using and navigating the start page may not be needed, as the players can set up a game from the webpage, making the game start automatically with the correct name configuration. The option to start a quick game is implemented in order to start a game without connecting to the webpage. Pressing the “Start Quickgame” button and starting a quick game will mean that the game is not connected to any users and will not be stored anywhere for the players to look back on in the future. Pressing the “Start simulation” button will take the user to a simulation mode, which will be touched on later on in this chapter.

Looking at the final design, the most important components are the video feed, the displaying of balls and cue, the timer, and the game ID. A camera is connected, and the video is being displayed live throughout the entirety of the game. From this video feed, important information is gathered and displayed as balls left on the table, right below the video feed. A timer is also located in the lower center to display the duration of the current game. Knowing that the players are connected to the right game might also be vital, therefore a label displaying the game id have been implemented in the top left.

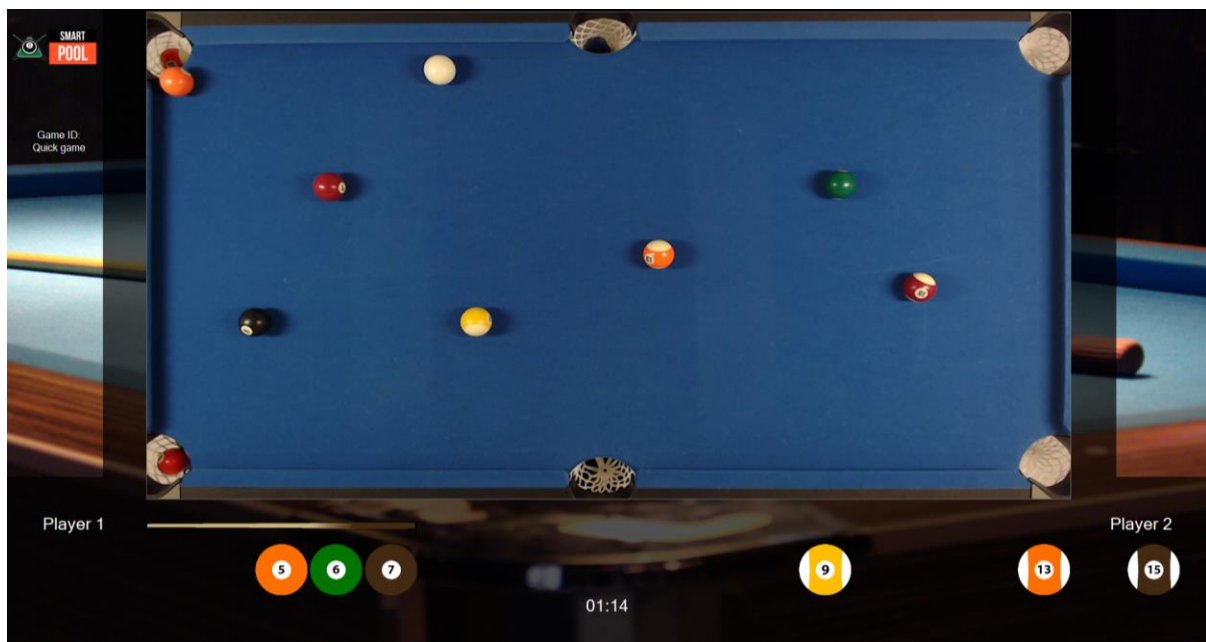


Figure 6-22: Final design of the game page

### 6.6.1 Live video feed

As mentioned in chapter 5 about hardware earlier in the report, the main base of the whole system is the camera mounted above the billiard table. A Logitech StreamCam is mounted at an elevated surface and connected directly to the computer running the vision system through a wired connection. If the camera is connected properly, one should be able to select it in the interface and have the live video displayed in the application. Without selecting the right camera in the interface, the application itself will not work, simply because the video feed is 100% necessary in order to send snapshots to the Custom Vision model. Speaking of the video feed, it is a high-quality video stream in 720p, assuring a good viewing experience for everyone choosing to watch through the application, as well as a good product as far as pictures for the Vision System itself is concerned. The camera and system also support 1080p full HD, but the NUC used in this project struggle a bit with 1080p. Through testing, the group have found multiple computers to work well with full HD, however, the NUC finds it a bit tough to compute that sort of resolution.

### 6.6.2 Processing

Upon pressing “SPACE”, the GUI instructs the vision system to execute the methods described in chapter 6.4.1. This process takes about 5 seconds at most, depending on the internet connection. While the system is processing the image, the user will not be able to send another snapshot before the process is finished. It should be easy for the user to know if the processing is still going, as a loading icon should be displayed in the interface. Promptly after the processing have finished, the loading icon will disappear, and the interface will display which balls are left on the table, and whose turn it is, according to which balls have been potted since the last image was processed.

### 6.6.3 Timer

In order to keep track of how long a game has lasted, a simple timer running in the background was implemented. The timer works as a stopwatch. It starts counting as soon as the game is initialized by pressing “SPACE” and stops when a winner is announced at the end of the game. The heavier and more resource consuming tasks run in a background thread, meaning it won’t disturb the main thread where the timer and GUI runs. This means that the timer and GUI can keep updating even when the heavier and more time-consuming tasks are taking place.

## 6.7 Simulation mode

Throughout the development of the main program, a simulation mode has been kept from the very beginning using much of the same code and methods as the main application. The simulation mode has been updated continuously together with the desktop application and have been an amazing tool for testing the logic and system without needing to connect to the pool table itself. The group decided to keep the simulation mode in the final delivery, as it can be a good tool for new users as training or getting used to the system itself.

The simulator is designed to look exactly like the main program and works pretty much exactly the same way. One noticeable change however is the lack of video in the application. Without a camera connected, it is difficult to show a live video feed, but that’s not what the simulator is for at the end of the day. Replacing the video, a set of pictures are loaded up, and shuffled through as the simulated game goes on. Pressing space to process will send the picture shown in the application to the vision system, and it will show which balls are left on the table and whose turn it is, much like the main application does with a snapshot from the video feed. Looking away from the obvious difference with the pictures, the code behind is very similar to the main program and have been tied closely with the main development since the very beginning.

## 7 Connection-API

When designing multiple applications, there was a need for the applications to communicate with each other. The group decided to make an API as a communication interface between the vision system and the web application. This API is called “connection-API”, and it is created in Visual Studio by using ASP.Net Web API. This application also runs in the NUC, together with the vision system. The group choose to create an API since this will provide a flexible, scalable, and easy way to establish communication between the applications. In Figure 7-1 we can see a system sketch on how the API endpoints will be structured and accessed by both the web application and the vision system.

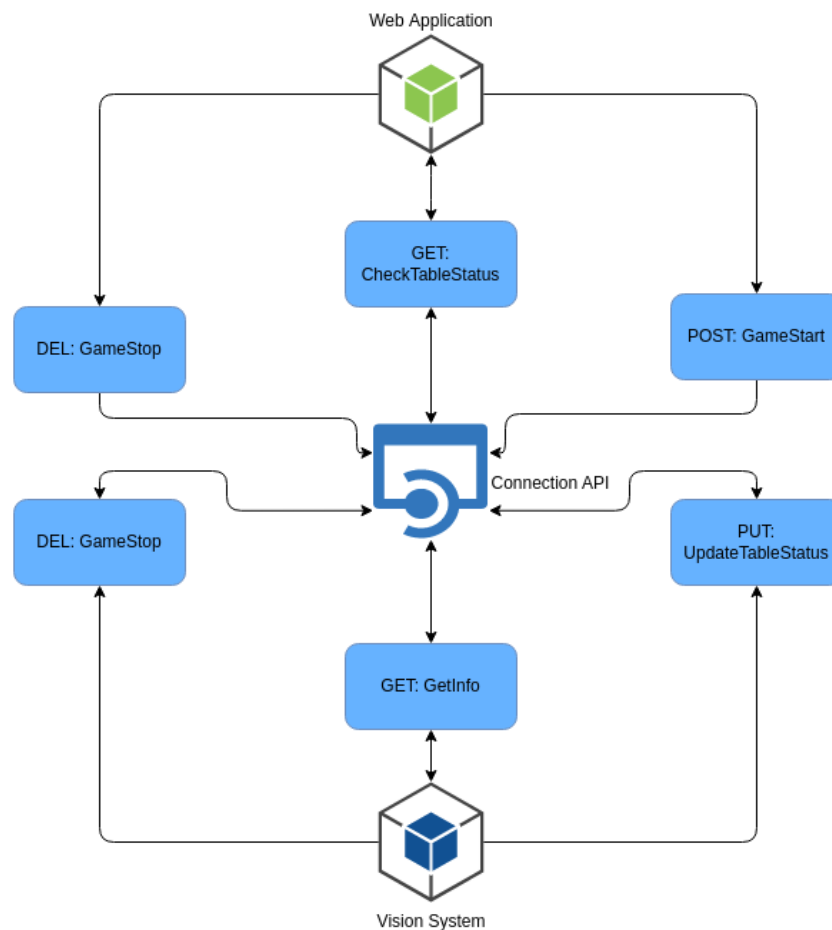


Figure 7-1: API endpoints

The API will expose multiple HTTP endpoints with different functionality that can be accessed by different applications. The endpoints can be grouped into four different HTTP methods.

- GET endpoints when called will request some data from a specified resource.
- POST endpoints when called will take provided data from the client and create or update data.
- PUT endpoints when called will update a specific resource. The difference between PUT and POST is that calling a PUT method multiple times will update the same data multiple times, while calling a POST method will add the same data multiple times.
- DELETE endpoints when called will delete a specific resource.

In this project, there are five different endpoints that can be called by the applications that retrieve, provides, updates, or deletes data on the vision system.

- **<GET> TableAvailability** is used to provide information about the status of the table. If there is an active game on the table it will return false and if the table is not in use, it will return true. This can be used to decide if the web application can create a new game on the specific table or not.
- **<POST> StartGame** is used to start a new game. This endpoint requires five parameters. The game-ID, the two player-ID's and the usernames. After receiving the data, the endpoint will validate the data and if valid start a new game and return an "ok" response. If the data is invalid or wrongly formatted it will return a bad request and not start the game.
- **<DELETE> GameStop** is used to stop an ongoing game. This endpoint only requires one parameter that is the game-ID. If the provided game-ID is matching the game-ID on the active game, the game will be stopped, and an "ok" response is returned. If the game-ID is not matching, it will return a bad request, and the game is not stopped.
- **<PUT> UpdateTableStatus** is used to update the status of the table. The method requires one parameter which is the status (true/false). The vision system will use this to update the table-status when starting and stopping a game.
- **<GET> GetInfo** is used to access information about an ongoing game. This is mainly used by the vision system to acquire the game details of a new game.

## 8 Database

To store and access data from the different applications, a database was created. The database will be accessed by both the web application and the vision system. The vision system is mostly storing and retrieving data about a specific game. While the web application is responsible for creating new users, games, tournaments, and update these accordingly. The data stored in the database will also be used for different statistical calculation in use on the web application.

### 8.1 Evaluated databases

When choosing a database, there are mainly two different database types to choose from. The first is a relational database where the data is structured in tables, often with relationships and dependencies. The other type of databases are non-relational databases. Here, the data is not structured in the same way as in a relational database and can support larger amounts of data.

Since most of the data generated will have natural relations, the best option for the project would be a relational database. The databases in Table 8-1 have been evaluated according to price, documentation, ease of integration, and scalability.

Table 8-1: Overview of evaluated databases

Name	Open source	Price in NOK	Ease of use	Documentation
<b>MS SQL</b>	No	9000+	Easy	Very good
<b>MS SQL Express</b>	No	Free	Easy	Very good
<b>Oracle DB</b>	No	30 000+/-	Medium	Good
<b>MySQL</b>	Yes	Free	Easy	Good
<b>PostgreSQL</b>	Yes	Free	Easy	Good

- Microsoft provide both a free and a paid version of their SQL Server database. The free version limits the database size to 10GB, the maximum memory usage to 1GB and a maximum of 4 CPU cores. These limitations would not affect the project in the startup phase but could lead to scaling issues in the future.  
The paid versions have higher limits which would increase the scalability, but it would increase the cost, and it would introduce a more complex licensing depending on the usage. Each version mentioned above integrates well with both .NET and NodeJS.



## 8 Database

- Oracle is providing both a free and a paid database solution. The free option is MySQL, which is an open-source database. MySQL has been around for a long time and has proven its stability and reliability. It can scale well for small to medium applications and integrates well with .NET and NodeJS.  
Oracle's paid database is Oracle DB, which is an enterprise database comparable with MS SQL server from Microsoft. Oracle DB can manage large amount of data and can scale to meet the most demanding applications and data needs, but it comes with a huge cost and a complex licensing system.
- The final evaluated option is PostgreSQL, which aims to be an open-source enterprise database like MS SQL and Oracle DB. PostgreSQL provides many of the same features, but for free. It is one of the most popular open-source databases in 2022 and provides great scalability, documentation, and integration with both .NET and NodeJS.

After evaluating the different database options, the best option for this project would be PostgreSQL. The main deciding factor for this choice over MySQL, was the improved performance of PostgreSQL, and the ease of use of the administration-tool for the database (PGAdmin). The other options were dismissed because of excessive cost or low scalability compared with PostgreSQL and MySQL.

## 8.2 Database structure

After deciding on what database to use, the planning of the data structure and relations could begin. The database "Billiard\_System", contains eight tables with various relevant columns. Table 8-2 shows all the tables with a description of the various columns.

Table 8-2: Overview of tables and their columns

Player	Game	Game_ Player	User	Billiard_Ball	Tournament	Tourna- ment Players	Table
Players player- ID	Game ID	Game ID	User ID	Game ID	Tournament ID	Tourna- ment ID	Table-ID
Number of wins	Timestamp of game creation	Player 1's Player- ID	Username	X position	Tournament name	Player ID	IP address
Number of losses	Timestamp of game start	Player 2's Player- ID	First name	Y position	Timestamp of tournament start		Active status
Players user-ID	Timestamp of game end		Last name	Player ID	Timestamp of tournament end		
	Player-ID of winner		Email	Timestamp of ball location			
	Player-ID of loser		Password	Play count			
	Associated table-ID		Active status	Ball color			
				Ball ID			

### 8.2.1 Table overview

The tables, and the relationships between the tables shown in Table 8-2 above are based on information that the database must contain. The bullet-points below describe the information the database should contain.

- **“User”** is the table where the information about all the users who register on the web application will be stored. Here, there will be generated a user-ID, which will function as the table’s primary key. The table also contains the user’s first and last name, email address and password. To ensure that the password is stored in a secure way, all passwords are hashed and salted before being stored in the database. The last column of the table is the active column. This column is used to indicate if the user is active or not. Instead of deleting a user, the administrator can disable the account by changing the content of the column from true to false.
- **“Player”** is the table where the users are associated with players by including a column for user-ID. The player-ID will function as the primary key for this table. The table also contain columns for wins and losses, where the number of wins and losses associated with the player is stored. The last column is the “user-ID”, which is as foreign key from the “User” table used to link the user-account and the player together.
- **“Game”** is the table where all the game data is stored. Each game is assigned a unique game-ID, which acts as the table’s primary key. The table also contains three timestamp columns. One for when the game was created, one for when the game started, and one for when the game ended. The table also has two columns for the winner and loser. These columns are populated with the players player-ID’s. The last column is the table-ID, which associates the game with specific table. This is a foreign key inherited from the “Table” table.
- **“Game\_Player”** is the table where the games are associated with the players. The player-ID’s (foreign keys) of the players associated with a game-ID’s (primary key) are going to be stored in this table.
- **“Billiard\_Ball”** is the table where ball positions are associated with a game. Here the “ball-ID” column functions as a primary key and the “game-ID” column (foreign key) are associating the data with a specific game. The ball coordinates are stored in the column “x position” and “y position”. To differentiate the balls from each other, the “ball color” column is used. The “play count” column will increment by one for each turn taken by the players. Each turn is also associated with a player-ID (foreign key) in the “player-ID” column.
- **“Tournament”** is the table where the information about different tournaments is stored. The “tournament-ID” column is used as a primary key. The table also contains the name of the tournament in the tournament name column. The table also contains three timestamp columns, one for when the game was created, one for when the game started, and one for when the game ended

- “**Tournament\_player**” contain information about which players are associated with which tournament. Here, the “tournament-ID” and “player-ID” column makes up the primary key. This will ensure that we can only have one pair of matching tournament-ID and player-ID.

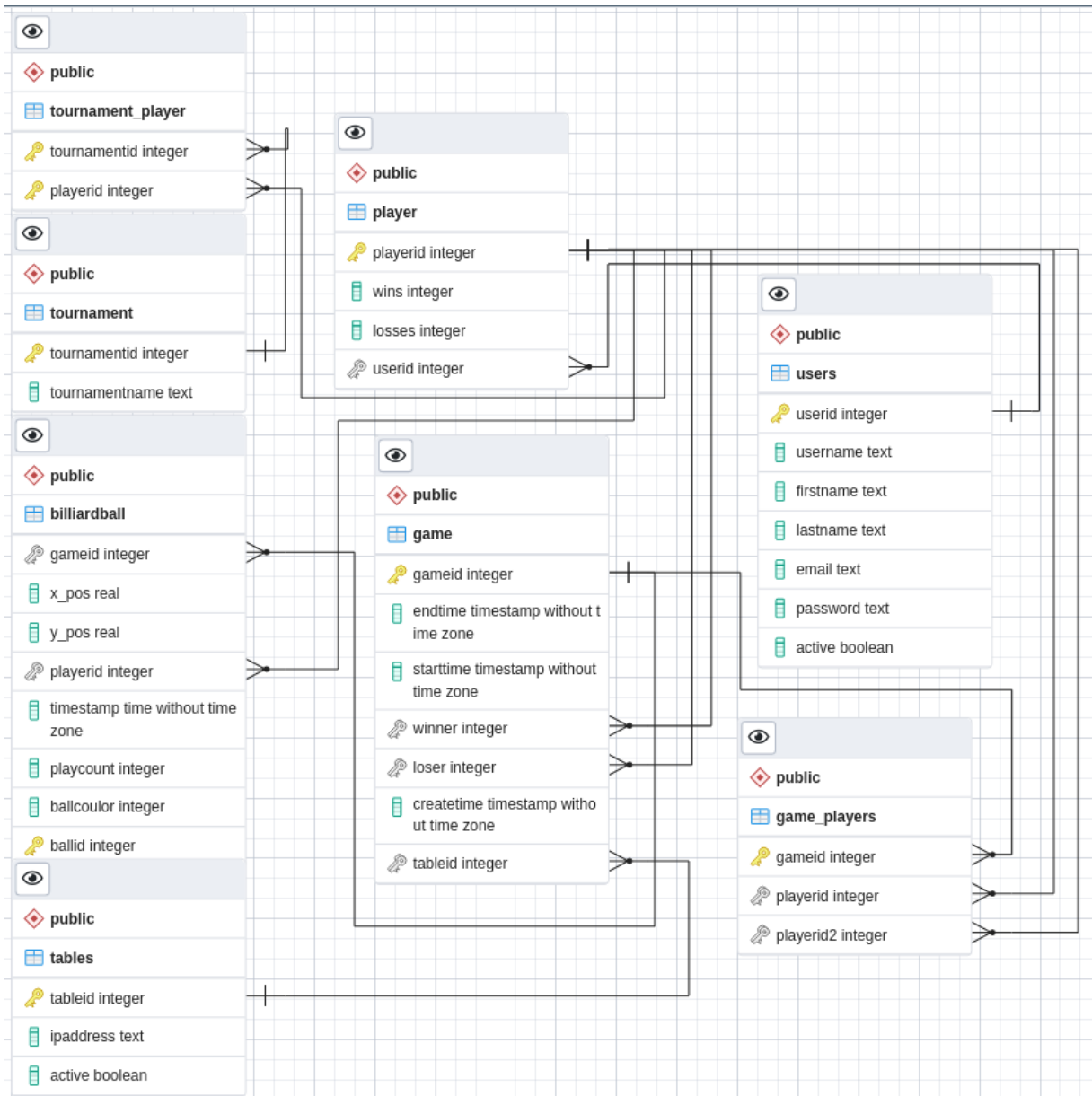


Figure 8-1: An overview of the tables and the relationships between them

## 9 Web Application

The web application is the main connection between the users and the vision system. It should be able to connect to the vision system. It should also easily increase or decrease the number of vision systems that are connected to the system, and support multiple concurrent users. The users should also be able to start games, tournaments, and see statistics about their gameplay.

The application is developed with JavaScript using the NodeJS framework in Visual Studio Code. NodeJS was chosen since it's a widely used, very agile and is widely supported with plugins, libraires and documentation online.

The subchapters below will go more into details on how these requirements were solved. To see the full requirement list, see chapter 4.2.

### 9.1 Libraries and package manager

When developing an application, the use of libraries can be a very useful thing. The use of libraries can decrease the time used for development and make integration with other application and services much easier. To install these libraries and keep them updated the package manger node package manger, or for short NPM, has been used.

Without these libraries the main application will not work, and they will therefore be considered dependencies. All the dependencies for the web application are listed in Table 9-1.

Table 9-1: The web application's dependencies

Name	Usage	Version
Express	The web application framework	4.17.2
Express session	Used to keep track of user sessions	1.17.2
Express flash	Used for displaying flash messages	0.0.2
PG	Used to interact with the Postgres database.	8.7.1
Passport	Used to authenticate users across the webapp	0.5.2
Passport-local	Used to authenticate locally	1.0.0
bcrypt	Used to hash and salt passwords	5.0.1
dotenv	This is used for storing and accessing sensitive information (passwords, API keys, etc)	10.0.0
Node-fetch	Used to interacts with API	2.6.1
Canvas	Used to draw HTML canvases	2.9.1
EJS	Template engine to generate HTML content	3.0.2

Figure 9-1 shows the use case diagram of the web application. The diagram is used to display all functions accessible to the different users using the application.

- 
- The diagram illustrates the permissions for three user roles: 'User with account' (purple), 'User without account' (green), and 'Administrator' (red). A central column lists ten system functions. Purple lines connect 'User with account' to 'Join games', 'Create games', 'Create tournaments', 'View statistics', 'Rewatch old games', and 'Update user details'. Green lines connect 'User without account' to 'Watch live games', 'Public scoreboard', and 'FAQ pages'. Red lines connect 'Administrator' to all ten functions.
- | User Role            | Permissions  |
|----------------------|--|
| User with account    | Join games, Create games, Create tournaments, View statistics, Rewatch old games, Update user details  |
| User without account | Watch live games, Public scoreboard, FAQ pages   |
| Administrator        | Join games, Create games, Create tournaments, View statistics, Rewatch old games, Update user details, Admin panel, Watch live games, Public scoreboard, FAQ pages |

62

### 9.3 Program structure

The application is mainly divided into five different JavaScript modules. Each module (except `server.js`) is exposing different methods that can be utilized by the other modules. The application is designed in a way that makes the reuse of code possible and encouraged, like the classes described in the vision system's chapter 6.3. Figure 9-2 provides a visual representation on how the different modules make up the web application.

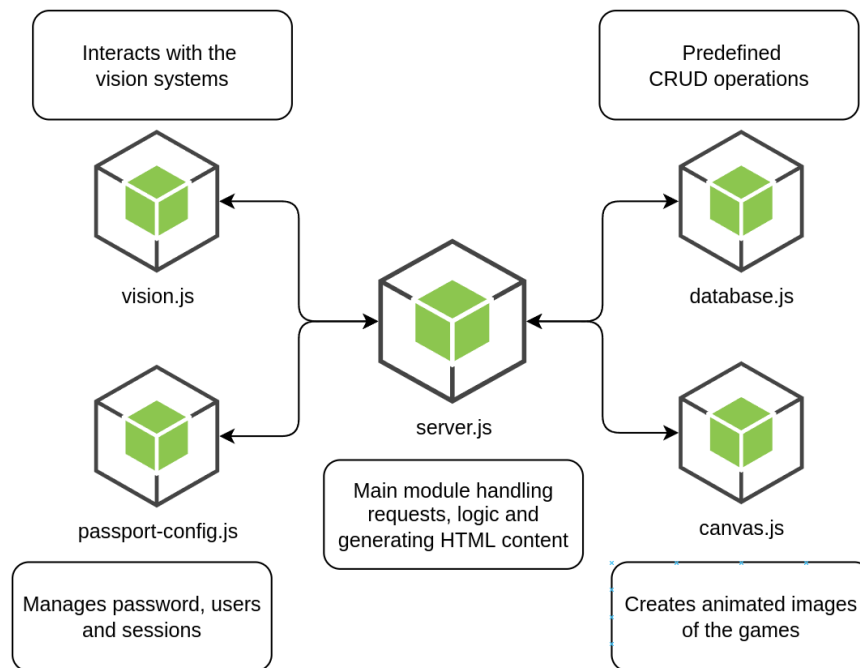


Figure 9-2: Structure of the web application

- The vision module provides different methods that allow the connection-API to interact with the different vision systems. The API operates as a message broker between the vision system and the web application, and the methods in the module can start and stop games or check if there is an active game or not.
- The passport-config module is the module responsible for the authentication and authentication parts of the application. This will keep track of sessions, and use bcrypt to hash passwords when registering a new user, or compare hashes when user tries to log in.
- The database module contains different methods to interact with the database. This module is used to get game details, update user data, delete a game, fetch ball coordinates, etc.
- The canvas module mainly contains one method which serves the purpose visualizing a billiard table when provided with the coordinates of a set of balls.
- The last module is the server module. This is the module running the express web application and provides the routes to all the different pages that make up the web application. This module is combining many of the other methods from the other modules to respond to requests and provide response back to the user with the required data.

## 9.4 Web pages and functions

The individual pages on the web application are sectioned into three different sections. This was done to create a structure for organizing the different content. To increase the visual appeal of the web application a background image of a billiard table was added. The background image can be seen implemented on the front page in Figure 9-7 and the source of the image is listed in the reference section [20].

- The first section of the page is the navigation bar. The navigation bar is what the user will use to navigate the page. The different menu options will be highlighted in response to what page the user is accessing. The navigation bar can be seen in Figure 9-3. The “Home” section is highlighted in white since the user is accessing the home page.



Figure 9-3: Navigation bar

The navigation bar is mostly static except the FAQ menu and the user menu on the far right. Pressing FAQ will reveal three new options that can be seen in Figure 9-4.

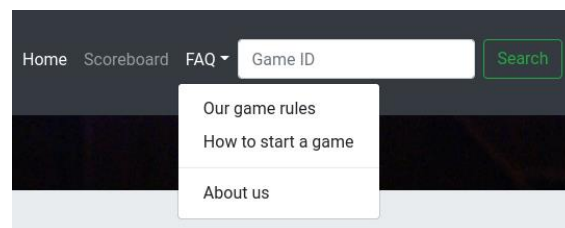


Figure 9-4: FAQ dropdown menu

The other dropdown menu will change from “Login or register” to the user’s username when they’re logged in. When not logged into an account, the menu will provide the user with the option to log in, or to register. When logged into an account the menu will provide the user with the possibility to either access their profile page or log out. When the admin account accesses the menu, it will be populated with an extra option to access the admin panel. These three options can be seen in Figure 9-5. The admin panel is described in more detail in chapter 9.4.8.

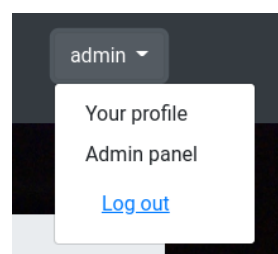


Figure 9-5: Dropdown menu to access profile or log out



- The second part of the page is the body. This is where the different content will be displayed depending on what page the user load. The size of the body is dynamic so it will scale depending on the page content and the user's screen size.
- The last section is the footer. The content in the footer is static and is displayed in the bottom of every page. This is mostly used to indicate the bottom of the page. The footer also contains a link to the about page where the user can read about why this page exists and who created it. The footer can be seen in Figure 9-6.

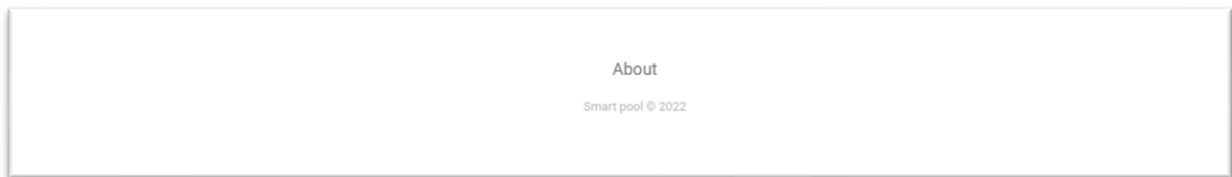


Figure 9-6: Web page footer including an “About” section

#### 9.4.1 Front page

The front page is the first page the user will access and it's therefore very important that it gives a good first impression. Therefore, the decision to make it a very simple page was made early in the design process. If the user is not logged in, they will be prompted with a welcome message, and information about how to get started playing. This can be seen in Figure 9-7.

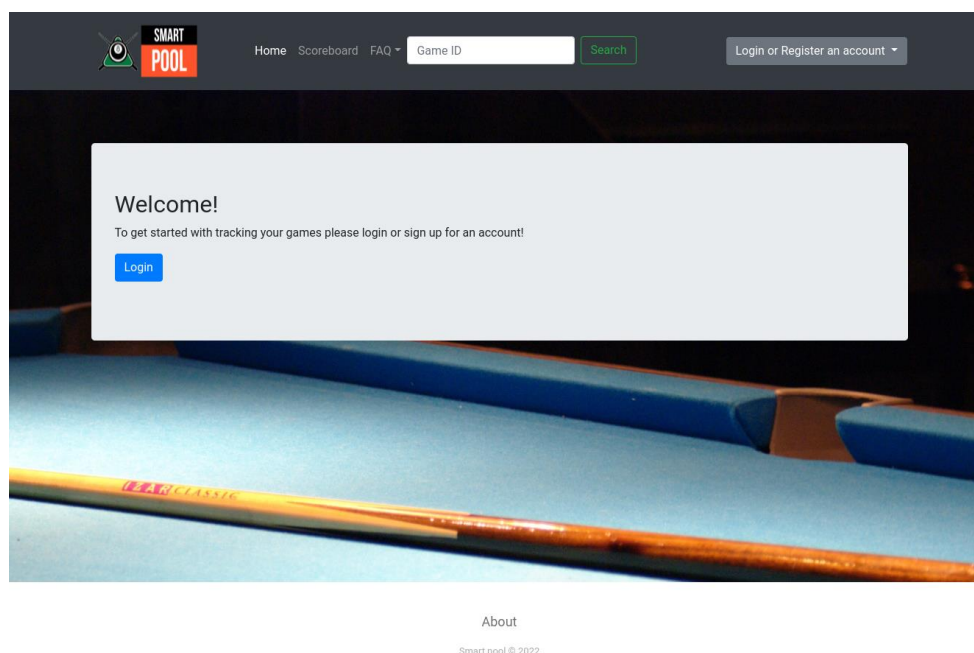


Figure 9-7: Frontpage when user is not logged in.

If the user is logged in, they will instead see a message saying that they could join a game by providing a game-ID or press a link to create a new one. This can be seen in Figure 9-8.

**Welcome!**

To join a game, please provide a game ID and press the join game button.  
If you would like to start a game or a tournament press [here](#).

Game ID

[Join game](#)

Figure 9-8: Frontpage when user is logged in

### 9.4.2 Register page

The “register” page is where users can sign up for an account for the website. The page requires the user to input their first name, last name, email, and a password. The password must meet a minimum requirement of 12 characters, minimum contain one number and a special character. If the user inputs mismatching passwords or a password not meeting the minimum requirements, they will be provided with an error message. The different messages can be seen in Figure 9-9.

**Register a new user**

Please pick a username:

Demo

First name:

demo

Last name:

bruker

Email:

demo@bruker.com

Password:

Choose a strong and secure password

Retype password:

Retype your strong and secure password

Passwords requirements:

- 12-24 characters,
- Minimum one number,
- Minimum special character

Please fill out the form

Do you already have an account? [Login here](#)

**Register a new user**

Please pick a username:

Demo

First name:

demo

Last name:

bruker

Email:

demo@bruker.com

Password:

\*\*\*\*\*

Retype password:

\*\*\*\*\*

Password does not contain a special character and/or a number

Please fill out the form

Do you already have an account? [Login here](#)

**Register a new user**

Please pick a username:

Demo

First name:

demo

Last name:

bruker

Email:

demo@bruker.com

Password:

\*\*\*\*\*

Retype password:

\*\*\*\*\*

Password is not long enough

Please fill out the form

Do you already have an account? [Login here](#)

**Register a new user**

Please pick a username:

Demo

First name:

demo

Last name:

bruker

Email:

demo@bruker.com

Password:

\*\*\*\*\*

Retype password:

\*\*\*\*\*

Passwords are matching!

Create account

Do you already have an account? [Login here](#)

Figure 9-9: Registration form and possible outcomes

After the user has submitted the form, the data is validated to ensure the email and username is not already taken, and that the password meet the minimum requirements. If the validation succeeds, the password will be hashed, salted, and stored in the database. This will ensure that if the database is compromised by an attacker, the passwords will not be humanly readable. After the user data is added to the database, the user will be redirected to the login page displaying a success message.

### 9.4.3 Login page

The login page is where the user inputs their username and password to access pages like their profile page, game creation page, or other login restricted pages. If the password is incorrect or the user does not exist, the user is promoted with a flash message informing them about why they could not log in. The form and the flash message displaying incorrect password can be seen in Figure 9-10.

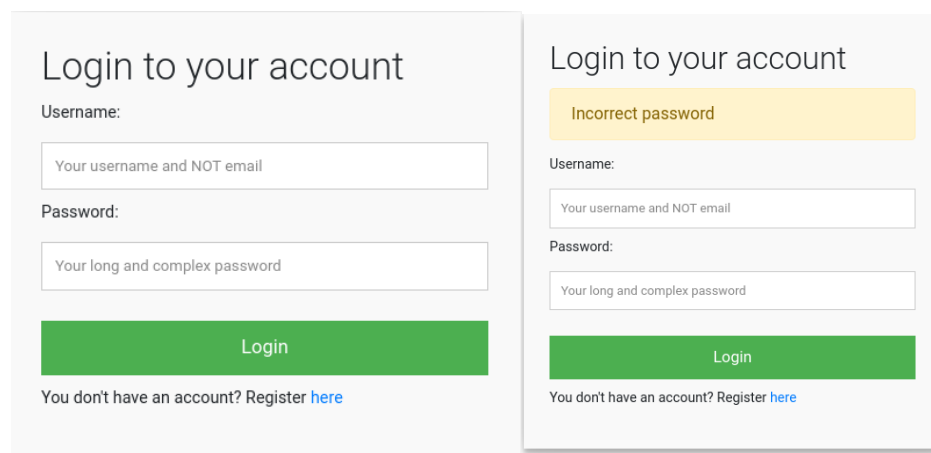


Figure 9-10: Login form

After the user has entered a username and a password, the web application will process the username and password with the passport module, which is described in more detail in chapter 9.3. The Password will validated, if the username and password combination match with the data in the database. If valid, the user is allowed to login.

### 9.4.4 Profile page

The profile page will be the user's main page where they can start new games, tournaments, watch previous games, see statistics, or update their user details. The statistics that's displayed to the user is a bar chart representing their win/loss ratio compared to the average ratio of the other players on the page. An overview of the page can be seen in Figure 9-11.

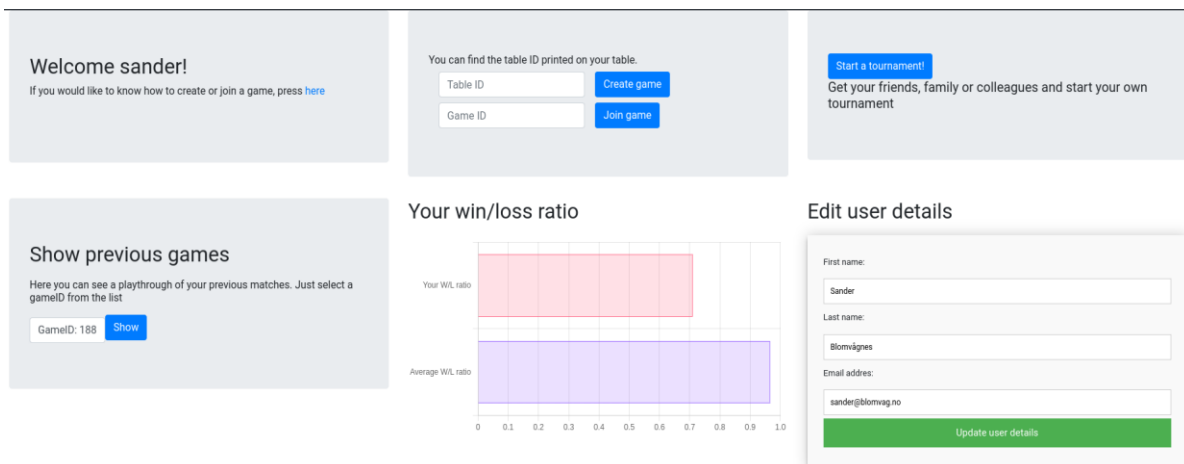


Figure 9-11: Profile page overview

If the user accesses their profile page while in a game, the ability to create or join a game will be replaced by a message saying that they are already in a game. The difference between these two views can be seen in Figure 9-12 and Figure 9-13.

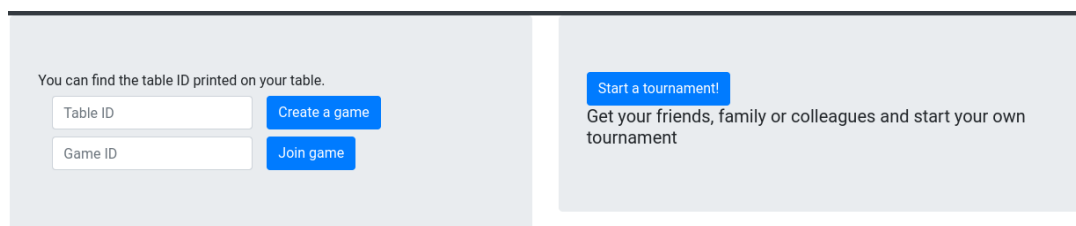


Figure 9-12: Profile page if user is not in a game or a tournament.

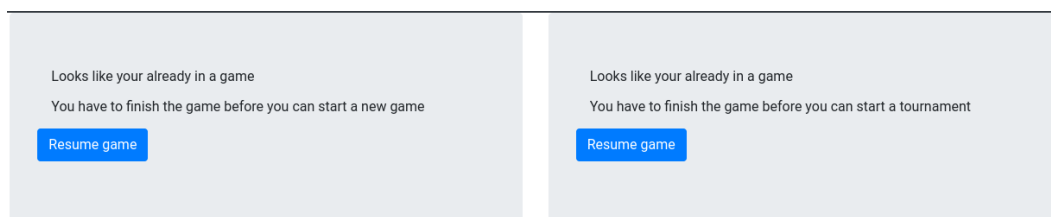
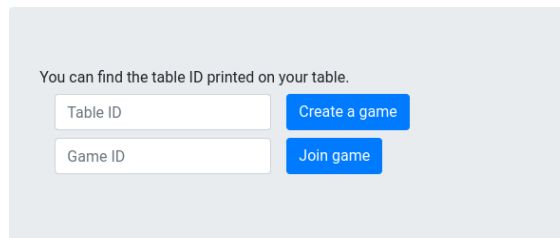


Figure 9-13: Profile page if user is in a game or a tournament.

#### 9.4.5 Create and join games

From the profile page, the users can either choose to create a new game or join someone else's game. This form can be seen in Figure 9-14. To create a new game, the user will have to provide a table-ID which will be printed somewhere on or around the billiard table. After the user enters the table-ID, they will be provided with a game ID, which they can share with the other player. The other player can then enter this game ID into the join game field to join the game.

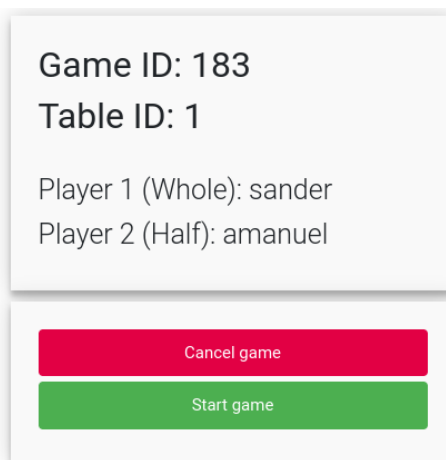


You can find the table ID printed on your table.

Table ID	Create a game
Game ID	Join game

Figure 9-14: Create and join game from

When both players have joined, they have the option to either start or cancel the game. This can be seen in Figure 9-15.



Game ID: 183  
Table ID: 1

Player 1 (Whole): sander  
Player 2 (Half): ammanuel

Cancel game

Start game

Figure 9-15: Game lobby page

### 9.4.6 Create a tournament

The users have also the possibility to start a new tournament from their profile page. After pressing the “Start a tournament” button, they will be redirected to a form where they can enter a tournament name and add all the associated players. The tournament mode has a minimum of 4 players, and a maximum of 12, where the user can dynamically adjust the number of players in the tournament. This can be done with the add or delete button in the bottom of the form. This can be seen in Figure 9-16.

The figure shows three variations of the 'Create new tournament' form. Each form has a title 'Create new tournament', a label 'Give the tournament a name', and a text input field containing 'Demo'. Below this is a label 'Please add all the usernames of all the players' followed by a list of text input fields for usernames. At the bottom of each list are two buttons: a blue '+' button and a red '-' button. A green 'Create tournament' button is at the bottom of each form, with the text 'This game mode is limited to 12 players' below it.

- Left form:** Shows 4 usernames (Username 1 to Username 4).
- Middle form:** Shows 6 usernames (Username 1 to Username 6).
- Right form:** Shows 8 usernames (Username 1 to Username 8).

Figure 9-16: Create tournament page with different amount of players

If any of the provided usernames are not associated with a user, a flash message displaying the invalid usernames are shown. This can be seen in Figure 9-17.

The screenshot shows the 'Create new tournament' form with an error message at the top: 'The username: olebrumm99 is invalid.' Below the error message is a label 'Give the tournament a name' and a text input field containing 'Tournament Name'. Below this is a label 'Please add all the usernames of all the players' followed by a list of text input fields for usernames. The first field contains 'sander', and the others are empty. At the bottom of the list are two buttons: a blue '+' button and a red '-' button. A green 'Create tournament' button is at the bottom of the form, with the text 'This game mode is limited to 12 players' below it.

Figure 9-17: Invalid username in tournament creation

9.4.7 Previous games

To provide the users with a possibility to rewatch their games, there has been implemented a feature, which will let the user choose from a list of their previous games and pick a play-through. This option can be found in the user’s profile page, and one can see an example of this in Figure 9-18.

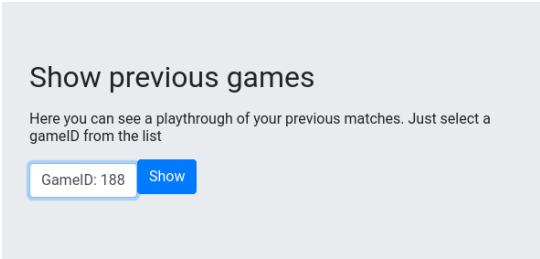


Figure 9-18: Show previous games

After choosing a game it will load within 4-10 seconds depending on the number of images the system has to generate. The user can then see who they played against, playtime, and who won the game. They can use the back-and-forth arrows to cycle between all the turns and see how the game progressed. The images are generated the same way as described in chapter 9.4.9, and a small amount of frontend JavaScript handles the cycling between the images. This menu can be seen in Figure 9-19.

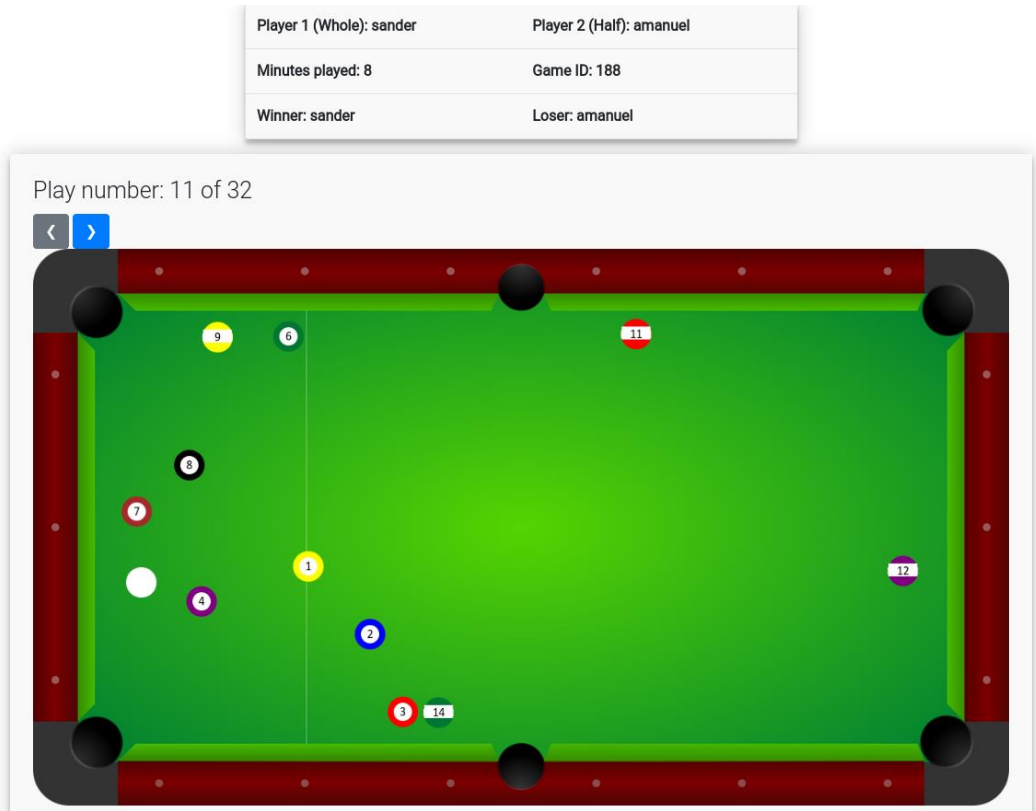


Figure 9-19: Rewatch previous games

9.4.8 Administration panel page

To provide the system administrators the possibility to manage tables, users and games, an admin panel was created. To ensure that the page is only accessible to authorized personnel, it can only be accessed by the user with the username “admin”. After logging in with the admin user, the admin panel can be accessed by the dropdown menu on the right side of the page.

The admin panel is divided into three main sections:

- The first section for the admin panel is for managing the tables connected to the web application. Here, the admin user can add a new table by providing the IP address of the table. If they would like to deactivate a table for being used, they can select the table ID from the dropdown menu and press the “Deactivate table” button. This will prevent any users from creating a new game on the disabled table. This can be undone by selecting the disabled table ID from the list and press the “Activate table” button. These options can be seen in Figure 9-20.

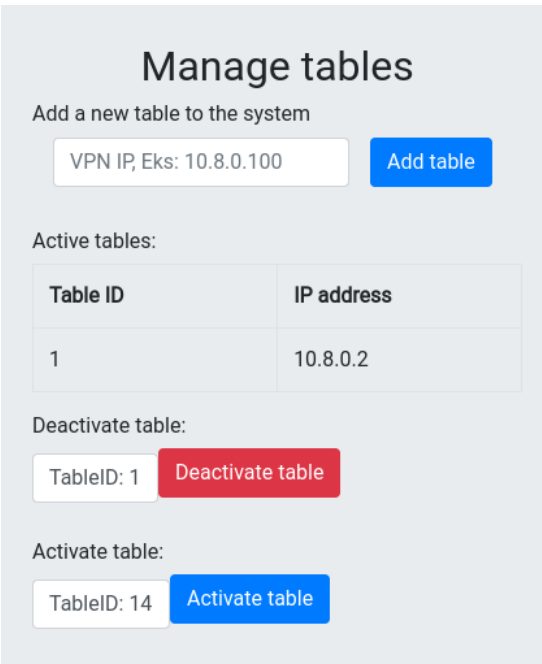


Figure 9-20: Manage tables menu



- The second section of the admin panel is for managing users. If the admin users wish to disable access to the web application for a specific user, they can select that user from the list and press the “Deactivate user” button. If the deactivated user tries to login, they will get an error informing that the account has been deactivated. This can be undone by selecting the disabled username from the list and press the “Activate user” button, as seen in Figure 9-21.

**Manage users**

Deactivate users:

Username: aleksander Deactivate user

Activate users:

Username: 232145 Activate user

Figure 9-21: Manage users’ menu

- The last section of the admin panel is for managing active games. Here, the admin user will have an overview of all the active games and can cancel these by pressing the “cancel” button. This can be useful if there are any issues with the software running on the table, or the players have left the table and forgot to finish or cancel the game. This panel can be seen in Figure 9-22.

**Manage active games**

GameID	Start Time	Create Time	TableID	Player 1	Player 2	
197		16:52:19	1	20		<span style="background-color: red; color: white; padding: 2px 10px;">Cancel</span>

Figure 9-22: Mange active games menu

### 9.4.9 Live gameplay page

To provide the possibility to watch live games from anywhere in the world, an animated version of the game can be accessed by entering the game-ID in the search field in the navigation bar. After providing a valid game-ID the live game page will reload and show the latest data of an ongoing game.

To display the balls, the latest ball locations are retrieved from the database, and a HTML canvas is created with a picture of a pool table [21]. After the table is added, each ball is drawn on a separate layer based on their location, colour, and type. When this is done, the image is converted to a byte64 string and displayed as an image for the users. The page can be seen in Figure 9-23.

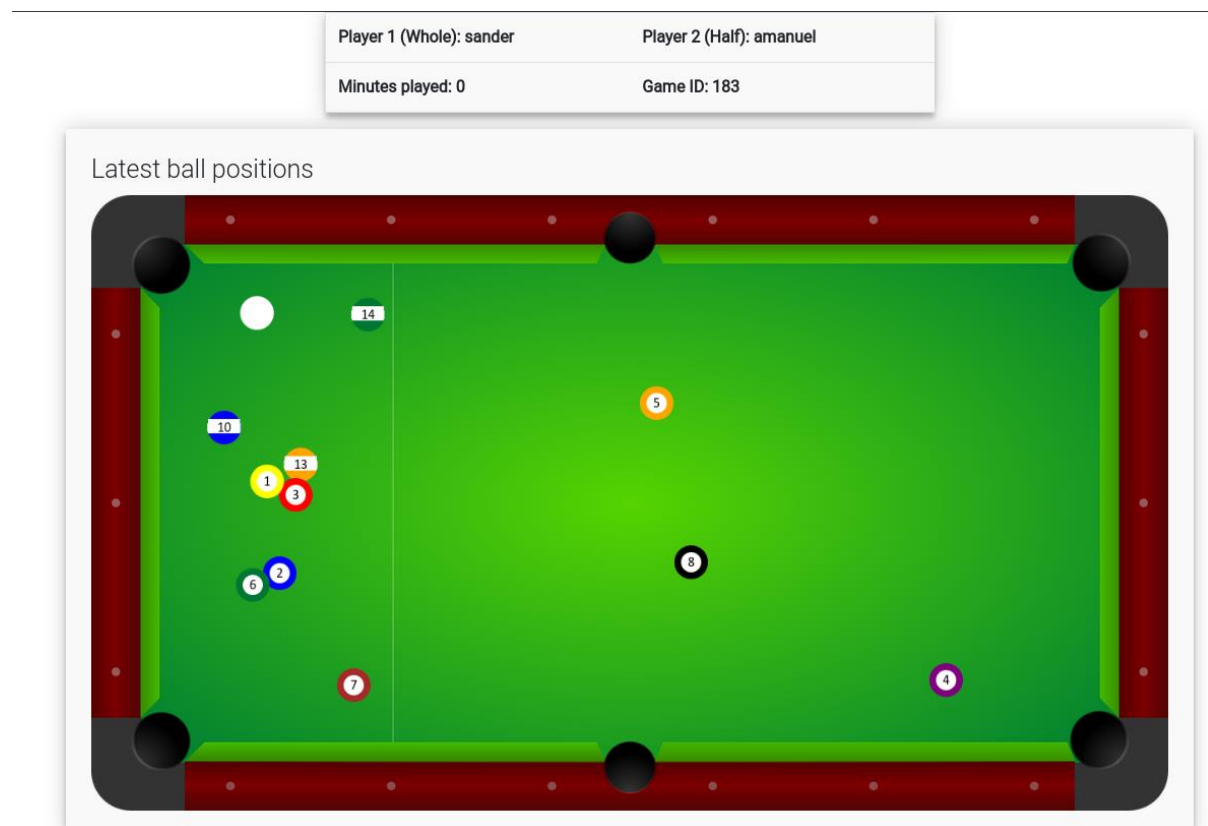


Figure 9-23: Graphical representation of the live game

### 9.4.10 Scoreboard page

To provide an overview of the players ranking on the page, a scoreboard was developed. The page calculates the win/loss score for all registered users and provides the data visualized in a bar chart. The page is not access restricted and can be accessed even without an account. The chart is generated using chart.JS and can be seen in Figure 9-24.

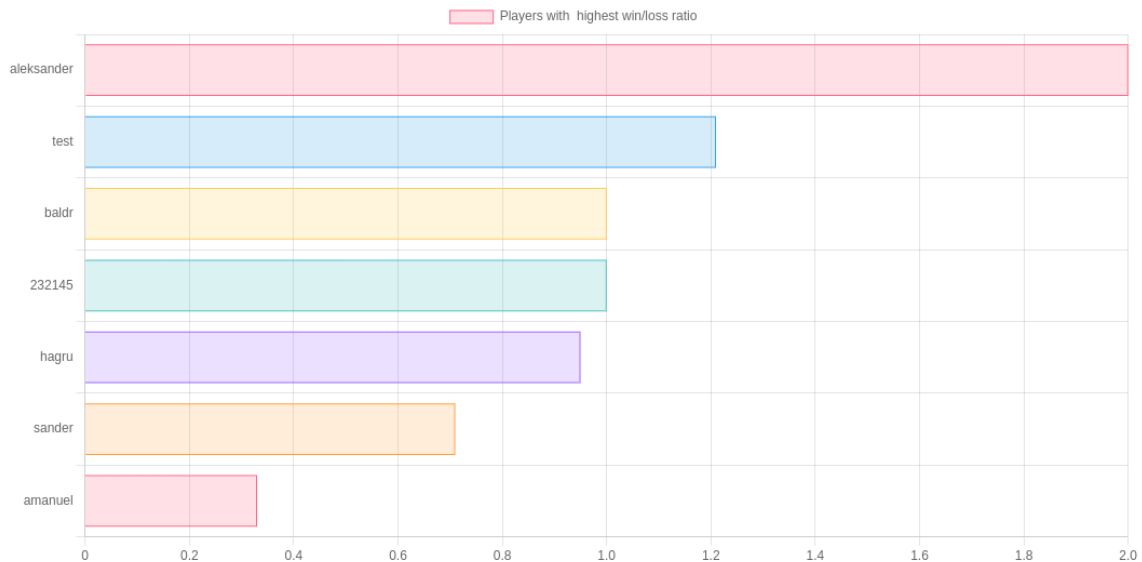


Figure 9-24: Scoreboard page

### 9.4.11 FAQ pages

There are three different pages accessible from the FAQ menu in the navigation bar. All the pages are delivering static content to provide the users with different information. The sites can be accessed by both users with and without an account.

- The first page is for displaying a list of the game rules. This was made to ensure that everyone uses the same ruleset as the vision system. The page is displaying a list of rules like what's described in chapter 6.4.
- The second page is providing user with information on how to get started playing. The page provides a short step by step guide on how to create a game and how to join a game.
- The last page is the "About" page that will inform the user about why the page was created and who created it.

## 9.5 Communication and security

With a publicly accessible web application, communication with both different clients over unknown networks, other applications and databases will accrue. It's therefore important to ensure a secure and reliable communication between all devices and services.

### 9.5.1 Communication with the vision system

On the same virtual server as the web application, an OpenVPN server has been installed to provide connection with the vision systems. For each new vision system that wants to communicate with the web application and the database, a new certificate and private key must be generated.

Using the certificate, key, and a client application, the vision system (client) can establish an encrypted tunnel with the web application (server). This will ensure that all the communication between the server and the client is encrypted and cannot be easily intercepted. The OpenVPN server can handle up to 100 clients. This is possible because each client generates a very low amount of traffic. If there is a need for more than 150 clients, the virtual server can be equipped with more ram and processing power to accommodate the extra clients. According to OpenVPN's documentation a recommended practice is to add 1GB extra ram for each 150 connected clients [22].

Using a VPN will also solve the issue with establishing connections with devices behind a NAT. Figure 9-25 provides an illustration on how multiple vision systems can communicate with the web application and the database.

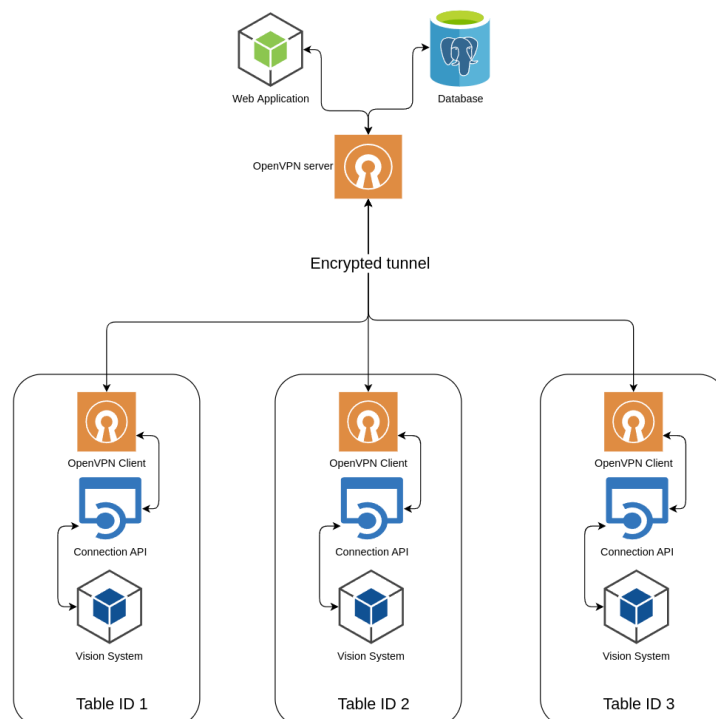


Figure 9-25: VPN connection between web application and multiple tables

### 9.5.2 Communication with clients

To ensure a secure connection between the clients and the web application, the use of the HTTPS protocol has been implemented in favour for HTTP. HTTPS will ensure that the communication is encrypted and cannot be easily intercepted and provides a higher level of trust. This is critical when exchanging passwords and personal information over the internet.

Let's Encrypt has been used to generate the required certificates, keys, and validate this towards a domain name (smartpool.no). This will result in a more secure page that is harder to impersonate, and more resilient to man-in-the-middle attacks [23] The processes of generating the required keys and certificates can be seen in Figure 9-26 provided by Let's Encrypt.

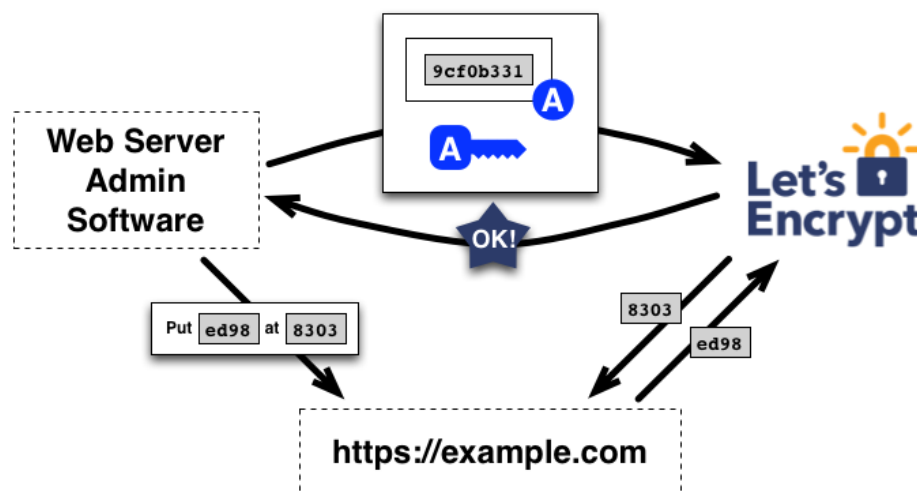


Figure 9-26: Let's Encrypt setup example

### 9.5.3 Communication with the database

To ensure limited access to the database, the database has been configured to only accept incoming connection from local services running on the same machine, like the web application, and the private subnet allocated to the OpenVPN clients (10.8.0.0/24). This will ensure that it's not publicly accessible and will therefor reduce the attack surface.

# 10 Testing

In this chapter, an evaluation will be given on how well the overall system works. This will be achieved by basing the tests in the requirements described in chapter 4.

Table 10-1 shows the test results for the vision system and GUI. Table 10-2 shows the test results for the web application. The testing of the system was also conducted with external students and their feedback is included in the overall evaluation.

Table 10-1: Tests for the vision system and GUI

Test case	Test description	OK	Failed	Comment
1	The vision system can establish connection with the connection-API	✓		
2	The vision system can establish connection with the database	✓		
3	The vision system can detect and classify billiard balls, using Azure's Custom Vision	✓		Fails to correctly classify the striped balls in some instances where the white part of the balls is positioned perpendicular to the camera.
4	Receive images and a live video feed of ongoing game from the camera	✓		
5	Implement game rules of 8-ball to interpret games	✓		Game rules are modified because of the limitation of having to process a still image
6	The vision system can transfer data and a live video feed of the ongoing game to the GUI	✓		
7	The GUI shows live video feed of ongoing game	✓		
8	The GUI shows scoreboard	✓		
9	The GUI indicates whose turn it is	✓		
10	The GUI can instruct the vision system to activate next play	✓		
11	The GUI can display which balls remains to be pocketed for each player	✓		

12	The GUI shows duration of the game	✓		
----	------------------------------------	---	--	--

Table 10-2: Test for the web application

Test case	Test description	OK	Failed	Comment
1	Users should be able to register an account	✓		
2	Users should be able to sign into their account	✓		
3	Users can create a game on any given available table. If not, they are provided with an appropriate flash message	✓		The flash messages could be improved with better description of the error.
4	Users can access previous games and the loading speed is not more than 10 sec	✓		Loading times can be above 7 seconds if there are more than 35-40 images.
5	Users can join a game. If not, they are provided with an appropriate flash message	✓		
6	Users should be able to cancel a game or leave a tournament	✓		This works, but if there is no connection with the connection API, the vision system will still be running on the local computer and the game will be changed to ended in the database.
7	Everyone on the page should be able to see the game rules (including the ones without an account)	✓		All informational pages are accessible without an account.

# 11 Deployment and distribution

In the later stages of this project, the need for a way to download and install these applications also had to be considered. What options were available and evaluated, and which way would be the most convenient for both developers and users? The following chapter goes through the deployment and distribution aspect of the applications.

## 11.1 Deploying the web application

After the development of the web application was completed, it had to be deployed somewhere so it could be accessed by clients over the internet. For this, two options were considered:

- The first option was to deploy it locally at the University. This could probably be provided free of charge by the University. A downside of this option is that it could be more time consuming since the students don't have access to the necessary equipment and infrastructure.
- The second option was to use a public cloud provider. This option would provide the students with full access to a virtual machine with a public IP, and a basic firewall for about 4 USD per month. This also provides great scalability and the possibility to get more features like VPN gateways, load balancer, etc if needed.

After evaluating the different options, the public cloud option was selected. This was selected since both Azure and Digital Ocean was providing a free 100 USD credit. The ability to also setup everything without having to interact multiple people was also a great add-on.

Digital Ocean was used since the free credits provided by Azure was going to be used for training the vision model as seen in chapter 6.2.

A virtual machine was deployed with 1GB ram, 1 CPU core and 25GB of storage with the Linux OS Ubuntu server 22.04. PostgreSQL was installed and configured to match the schema discussed in chapter 8.2. The code for the web application was cloned from the GitHub repository and dependencies were installed using the "NPM update" command as mentioned in chapter 9.1. The application pm2 was installed to run the web application and keep track of logs. After the installation of pm2, the application could be started by running the command "pm2 start server.js".

To access the web application remotely, incoming connections on port 443 and 80 had to be allowed in the firewall on the server's and on the firewall for the local network in Digital Ocean. The port 1194 was also allowed in the firewall to accommodate the VPN connections from the vision systems.

To allow communications with the vision systems, OpenVPN was installed and configured using an automated script [24]. After the installation was finished, a new client config was generated as described in chapter 9.5.1. The config file was imported into the local development computer and used to establish a connection with the server.

The installation process has been semi-automated using a bash script to make it easier to deploy the application. This script will install all the needed frameworks, clone the code repository, install dependencies, configure the environment file, and generate new SSL certificates. After the installation is done, the script will provide the user with information on how to install the



SSL certificates and start the application. The script can be downloaded from the code repository.

## **11.2 Installing the Smart Pool application**

Following the completion of the final iteration of the vision system, a way to access it for third party users had to be developed. The easiest way to distribute the application without extra cost of hardware was through GitHub. GitHub is free of cost, and lets developers distribute their work across the internet.

To package the necessary files, a “Setup Project” was made through Visual Studio. The tool allowed multiple files and projects to be added, so that the user can install the whole package at once, including the main application and the API. In order to set up a connection however, the user will have to install and set up OpenVPN. This is a third-party program that has not been developed in this project and is therefore not included in the installer.

After downloading setup.exe or setup.msi from GitHub [25], the user will be presented with a very simple install wizard. From there, the user can choose the save location. When the installation is complete, the user will find an icon for the application on their desktop. The application is now ready to use.

## 12 Discussion

This chapter contains the discussion of the results and solutions that has been presented in the later chapters. To solve the problem at hand, the group created a vision system application and a web application. Upon adding new features or changes, the system would be heavily tested in order to secure the performance of the system. The first subchapter discusses some of the observed flaws that the vision system has and presents some plausible solutions for future work. The second subchapter discusses some of the observed flaws that the web application has and presents some plausible solutions for future work.

### 12.1 Vision system

The vision system is mainly created to detect billiard balls on the billiard table, judge a game of 8-ball, display an ongoing game, and store and retrieve data from a cloud-based database. The vision system executes most of its tasks as expected, but it does come up short in some aspects of these tasks, meaning there's still room for improvement.

#### 12.1.1 Custom Vision

The model fails to correctly classify the striped balls in some instances where the white part of the balls is positioned perpendicular to the camera. This can be seen in Figure 12-1. Instead of correctly classifying the balls, the model classifies the detected balls as a “white” ball. This is a comprehensible prediction from the model. This is because the striped balls, in instances where the white part of the balls is positioned perpendicular to the camera, have similar visual characteristics to the white ball. For future work, installing additional cameras in different positions around the billiard table that produce images of the billiard balls with different angles, should be considered. These images would then be sent to the model for training and prediction. This could drastically increase the model's precision to correctly classify the striped balls. A disadvantage with a solution like this is that the cost will be higher. Another option which could solve or completely eradicate this problem, would be having 7 yellow and 7 red balls instead of the striped and solid balls. Implementing this change would mean the model would have to be trained using that set of balls, and certain other features in the applications would have to be changed compared to how they are set up currently. However, that would be a simple, cheap, and rewarding solution.



Figure 12-1: The white parts of the striped balls are positioned perpendicular to the camera

Figure 6-11 shows the final performance-metrics for the model. The model's precision is at 98%, the recall is at 94%, and the mAP is at 94.7%. But these performance-metrics can be misleading, as the Custom Vision model could face challenges if it was faced with untrained scenarios. Although there was a focus on feeding the model with training-images that varied in visual characteristics, as the documentation for Custom Vision recommends [18], the variety of the images were limited because of the resources that were available. The main limitation was the availability of billiard tables that had different surface colours. The model would face some level of challenge if the billiard table used in this project was replaced by a billiard table that had a different surface-colour. This is because the model was trained using images that only included the blue-surfaced table. Although, it is unclear how much the model would be affected. A solution for this in future work, can be to train the model with tables having different surface-colours. Another limitation to factor in, is the lighting options. Although there was some verity in the lighting of the training-images, the room that was utilized offered little in regard to lighting options. Therefore, the assumption is that untrained lighting scenarios would have some degree of effect on the model's performance.

The vision system utilizes the model by interacting with a prediction-API. This means that the vision system is completely dependent on an internet connection. This only becomes a problem if the user wants to play a game and there is no available internet connection. For future work, exporting the model as a docker container should be considered. This docker container would then be installed in the same machine that runs the vision system. This solution would mean that the vision system could interact with the Custom Vision model without the need of an internet connection, and consequently the prediction-API. Additionally, the response time from the model, would also be reduced and no longer reliant on the internet connection. Lastly, by exporting the model as a docker container, the prediction-cost would be eliminated.

### 12.1.2 Visual system GUI

During development, and all the way throughout the project, performance have been heavily weighted. A clean and good-looking design, combined with a good performance was an obvious target to have in sight. WinForms is in hindsight not the best solution for either of these

things, as the performance is not great, and the design of a full screen high-definition application is not where it thrives.

As stated earlier in the report, the GUI struggled to run a timer while displaying a full HD video. The video feed would be alright, but the timer would not tick for some odd reason. Another thing that seemed to bottleneck the performance was the resolution of the background image. The image had to be downscaled to 360p in order to keep the framerate of the video feed at an acceptable level. Issues regarding the loading of the application also presented itself later in the development. Double buffering was implemented in order to reduce flickering while loading the application, and upon loading/removing the pictures of the balls. Double buffering seemed to help a lot, however, there are still some flickering and flashing when initially loading the application. All these problems could be a hardware related issues, and a more powerful computer would most likely preform even better, based on testing done with other computers.

To further boost the performance of the application, a change from WinForms to WPF would be beneficial. Both frameworks are great in their own aspect to their purposes, but WPF beats WinForms in every aspect when it comes to performance. WinForms might be easier to get started with, but WPF is newer and more advanced, allowing for better scalability (as it is not pixel-based) and better overall performance. In hindsight, WPF would be the better option.

The application is also heavily reliant on a person operating it. This could be one of the players, or a set person that would press “SPACE” after each play. Initially, this was also the plan, as the experience with such advanced applications and automation was limited. So, pressing a button to send a picture to the model seemed like a realistic way to solve that part of the task. A way to improve the system in the future would be to implement a feature that removes this operator role, making the system automated.

## 12.2 Web application

The web application is the main interface for the users interacting with the different vision systems. It provides both general information and can provide personalized information if the user has signed up for an account. Since the application is accessible over the internet it’s important that it’s secure to use, personal data is stored securely, and data access is limited. In the current state of the application, all requirements for the applications have been implemented. Most of the implemented features are working flawlessly, others could be improved somewhat, and only one requirement is not working as expected. These subchapters will look more into how these issues affect the system, and how they could be fixed or improved.

### 12.2.1 Tournament mode

During the start of the project, the group decided to create a tournament mode to allow the different users to create their own tournaments, but the implementation of this did not work out as planned. The creation of the tournament was implemented and worked as expected but linking games to a tournament did not work as intended. To solve this issue, a change in the database structure is necessary. This could be achieved if more time was provided. A possible solution could be to create a link (foreign key) between the game and the tournament table. There is also a need to implement this change in the application code.

### 12.2.2 Creating games and tournaments

At the end of the development, when discussing different security related topics, a problem was discovered. With only access to the web application, users can create games without being present at the billiard saloon. This can result in users sitting at home, creating games with different accounts to occupy all the tables without playing.

To solve this issue, a small validation could be implemented. This validation could ask the users to input a daily code only visible in the physical locations of the billiard tables. This could be implemented by creating a random 6-character long code that the user will have to enter before creating a game or a tournament. To display this code a separate screen could be used, or it could be integrated into the Vision Systems GUI visible when a game is not active.

### 12.2.3 Live streaming

The ability to provide a live feed of the games has been a big goal since the beginning of the project. A semi live solution has been implemented which provides an animated image of the latest known position of the balls on the billiard table. The semi live solution works, but it does not show where balls are pocketed, or how the balls moved since the last update. Considering how the vision system is designed, it's not possible to provide this information at this time. If provided more time, the video feed from the vision system could be accessed by the web application through the connection API, and the web application could provide this stream to the users.

### 12.2.4 Connection with the vision systems

To communicate with the vision systems, a VPN solution has been implemented. How this has been implemented is that each vision system that needs its own certificate, key, and client software. The generation and revocation of these certificates must be done through the servers CLI, and this manual work could be automated and implemented into the admin panel if given more time. This would greatly increase the ease of use of the product.

Another solution to this problem would be to set up a site-to-site VPN connection. This would provide a local network access between the vision system at the billiard saloon, and the web application in the cloud. This would eliminate the need for the individual certificates for each vision system, but it would require more advanced networking hardware on the premises. This could increase the initial cost of implementing the solution if such hardware is not present. An example of the proposed site-to-site VPN solution can be seen in Figure 12-2.

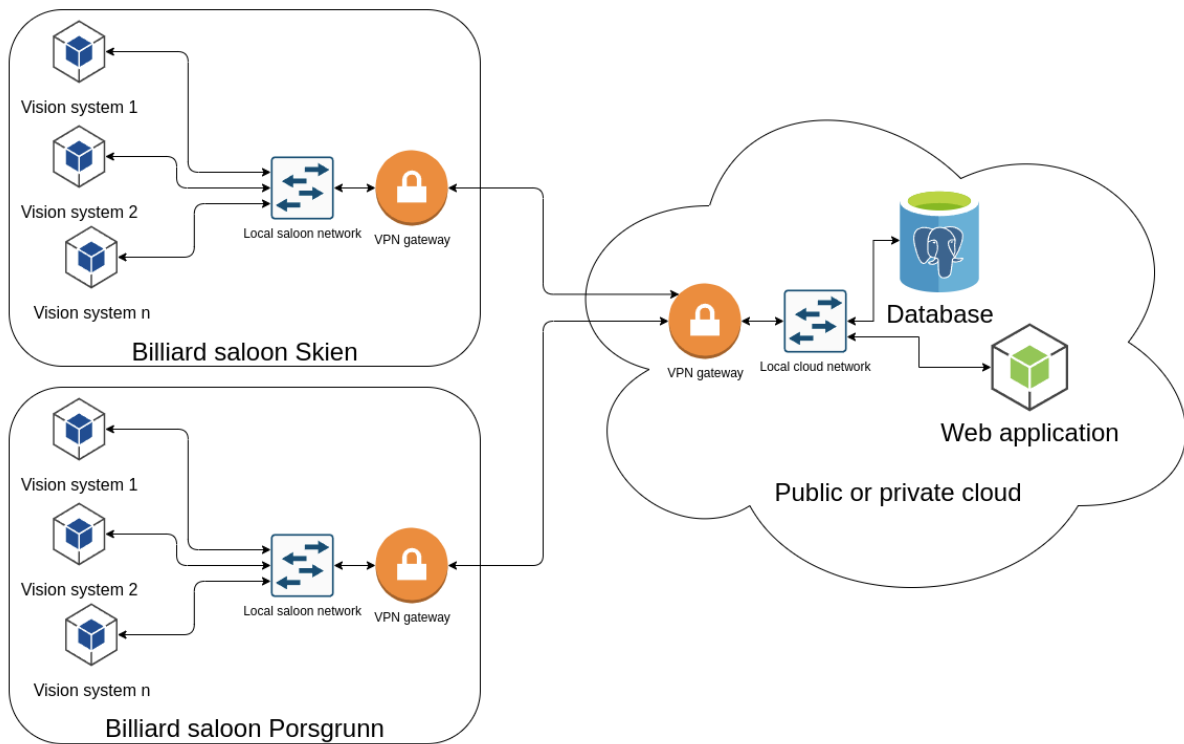


Figure 12-2 Proposed site to site VPN solution

## 13 Conclusion

The objective of this project was the creation of a system that can monitor and present the progression and outcome of a game of 8-ball. To achieve this, a vision system application that utilize a Custom Vision model and a GUI to present, configure and monitor a game of 8-ball. A web application was created to provide the users with an interface to interact with the vision system and present games, both live and archived. This is achieved by utilizing the database and the connection-API.

Azure Custom Vision was the service that was used to detect and classify the billiard balls, mainly because of its affordability and flexibility. Although the developed Custom Vision model works well in most instances, there were an observation made where it didn't perform as expected. The observation mentioned, was the model's inadequate ability to correctly classify the striped balls correctly in some instances. Because of the limited resources, the model may also face problems in new environments that introduce the model with untrained scenarios. The vision system's reliance on the internet will also restrict the usability of the system when there is no internet connection.

The vision system GUI was made in Visual Studio, using C# as the programming language, and WinForms as the base framework for design. This decision was made due to the group having experience using these tools throughout the education. Deciding to go with WinForms eventually proved to be a slight misstep, as it buckled a bit in terms of performance as the program got more complex. As a result of that, features such as the quality of pictures and video had to be compromised.

The web application was written in JavaScript using Visual Studio Code. The application provides the users with an interface to interact with the vision system from their phone, tablet, or computer. The user can create or join games, see statistics, and view playthroughs of previous games. The tournament mode was not fully implemented and are now in a semi working state. The possibility to create the tournament and assigning user to it is working but linking games to a tournament has not yet been implemented. This has resulted in a non-working tournament mode. The application also has implemented a live view page where users can enter a game ID to watch an ongoing game live. This was implemented by not streaming the video feed from the camera, but instead creating a static image representing the latest known ball positions. This resulted in just a static image being represented to the viewer, which occasionally updates, instead of an actual live feed where one can see the balls moving in real time.

To establish communication between the web application and the different vision systems, a VPN solution has been implemented. Each vision system will be identified using a unique certificate. This will allow the application to connect to multiple systems that can be in different billiard saloons all around the world. An improved solution for this would be to create a VPN tunnel between each location instead of having each individual vision system establish its own tunnel. This will result in fewer endpoints to manage, and each vision system does not need a VPN client installed on the machine.

As seen in chapter 10, the system works well under testing conditions. The group managed to fulfil most of the goals set for this system, however, because of time constraints and limited resources, the system is still in need of improvements. A more automated, optimized, and robust system could be possible in the future.

# 14 References

- [1] O. Kravchenko, M. Leshchenko, D. Marushchak, . Y. Vdovychenko and S. Boguslavska, “The digitalization as a global trend and growth factor of the,” 2019.
- [2] Digitalpool, “digitalpool.com,” 1 may 2022. [Online].  
Available:<https://digitalpool.com/>.
- [3] D. Abera. [Online].  
Available:<https://www.peerspot.com/products/microsoft-azure-devops-reviews?fbclid=IwAR1axts3S0rFrcsIHjAhfp04pch5G77sCmPbTj8ksiMhN3wA8l3UCnJHUXI>.
- [4] Logitech, “logitech.com,” 1 January 2022. [Online].  
Available:[1][https://resource.logitech.com/w\\_900,h\\_900,c\\_limit,q\\_auto,f\\_auto,dpr\\_1.0/d\\_transparent.gif/content/dam/logitech/en/products/webcams/streamcam/gallery/streamcam-gallery-1-graphite.png?v=1](https://resource.logitech.com/w_900,h_900,c_limit,q_auto,f_auto,dpr_1.0/d_transparent.gif/content/dam/logitech/en/products/webcams/streamcam/gallery/streamcam-gallery-1-graphite.png?v=1).
- [5] Logitech, “logitech.com,” Logitech, 1 January 2022. [Online].  
Available:[2]<https://cdn.cnetcontent.com/78/ad/78adbee3-e550-45bc-82f9-6b75b3e4c3b7.jpg>.
- [6] Amazon, “amazon.com,” [Online].  
Available:[1][https://m.media-amazon.com/images/I/71YWwJGP86L.\\_AC\\_SL1500\\_.jpg](https://m.media-amazon.com/images/I/71YWwJGP86L._AC_SL1500_.jpg) .
- [7] Billiards Colostate, “billiards.colostate.edu,” [Online].  
Available:<https://billiards.colostate.edu/faq/table/sizes/>.
- [8] “The Matcha Initiative,” [Online].  
Available:<https://www.thematchainitiative.com/find-a-solution/digital-footprint-it-green-hardware-and-e-waste>. [Accessed 19 05 2022].
- [9] National Instruments, “NI.com,” [Online].  
Available:<https://www.ni.com/en-no/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/select-license.html#>. [Accessed 7 may 2022].
- [10] OpenCV, “OpenCV.org,” [Online].  
Available:<https://docs.opencv.org/4.x/d1/dfb/intro.html>. [Accessed 7 may 2022].
- [11] Microsoft, “What is Computer Vision?,” [Online].  
Available:<https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/overview>. [Accessed 7 may 2022].



- [12] Microsoft, “What is Custom Vision?,” 3 February 2022. [Online].  
Available:<https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/overview>.
- [13] D. D. W. H. T. Mathew Salvaris, *Deep Learning with Azure*, Apress, 2018.
- [14] Custom vision, “Customvision.ai,” Azure, [Online].  
Available:<https://www.customvision.ai/projects/d2266057-b9f2-4d8e-a4a8-14fe78ac2edc#/performance>.
- [15] Microsoft, “microsoft.com,” 22 february 2022. [Online].  
Available:<https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/select-domain>. [Accessed 23 April 2022].
- [16] Microsoft, “microsoft.com,” [Online].  
Available:<https://azure.microsoft.com/nb-no/pricing/details/cognitive-services/custom-vision-service/#pricing>. [Accessed 11 05 2022].
- [17] O. D. Pedrayes , D. G. Lema, U. Rubén , D. F. García and A. Alonso, “Cost-Performance Evaluation of a Recognition Service of Livestock Activity Using Aerial Images,” MDPI, Spain, 2021.
- [18] Microsoft, “Quickstart: Build an object detector with the Custom Vision website,” 11 april 2022. [Online].  
Available:<https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/get-started-build-detector>.
- [19] RulesOfSport, “<https://www.rulesofsport.com>,” [Online].  
Available:<https://www.rulesofsport.com/sports/pool.html>.
- [20] “Piqsels,” [Online].  
Available:<https://www.piqsels.com/no/public-domain-photo-zyorh>. [Accessed 01 04 2022].
- [21] “Pixybay billiardtable 2d,” [Online].  
Available:<https://pixabay.com/no/vectors/basseng-biljardbord-biljard-snooker-4873047/>. [Accessed 02 02 2022].
- [22] “OpenVPN Access Server system requirements,” 12 05 2022. [Online].  
Available:<https://openvpn.net/vpn-server-resources/openvpn-access-server-system-requirements/>.
- [23] “Trust-tls-ssl-and-http,” medium, [Online].  
Available:<https://medium.com/mobile-development-group/trust-tls-ssl-and-https-b925ac9d59>. [Accessed 12 05 2022].
- [24] “OpenVPN install script,” 04 05 2022. [Online].  
Available:<https://github.com/angristan/openvpn-install>.

- [25] C. Hagrupsen, S. Blomvågnes and A. Isak, “Github,” [Online].  
Available:<https://github.com/hagru/PoolBachelor/releases/tag/Alpha>.
- [26] R. Barone, “idtech,” 11 September 2020. [Online].  
Available:[idtech](https://idtech.com).
- [27] S. Blomvågnes, “Webapplication repository,” 19 05 2022. [Online].  
Available:<https://github.com/SanderBlom/PoolFrontend>.

# Appendices

Appendix A – Project description

Appendix B – Camera settings

Appendix C – Log of training sessions done to Custom Vision model

Appendix D – Vision system GUI user manual

Appendix E – Gantt diagram

Appendix F – Class diagram of the concrete classes

Appendix G – Adding a new vision system to web application guide

Appendix H – Flowcharts for the web application

## Appendix A



# Bacheloroppgave

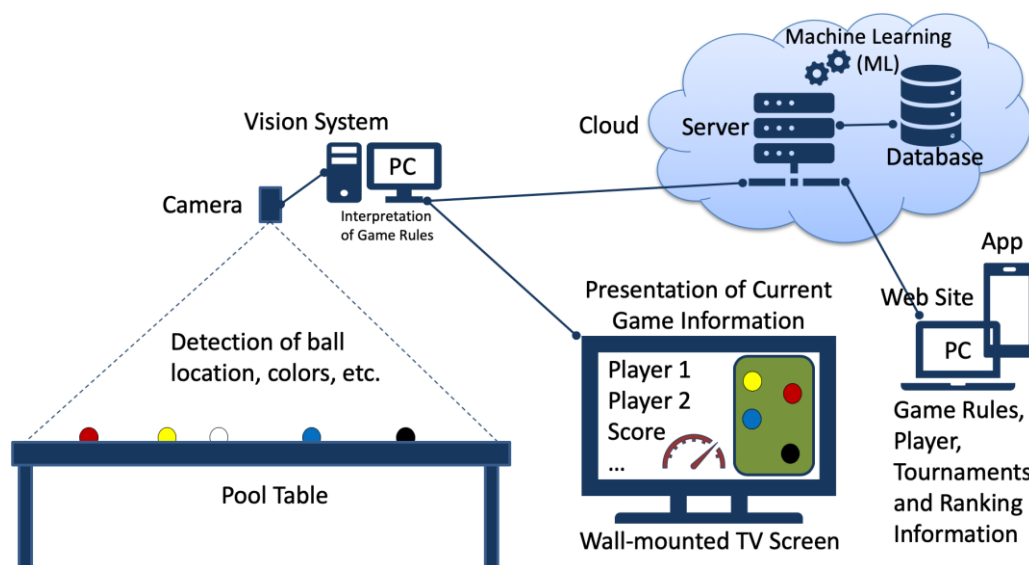
**Title:** Development of a camera-based system for 8-ball using Azure Custom Vision

**USN veileder:** Hans-Petter Halvorsen

**Ekstern partner:** Grenland Biljardklubb/Robert Immerstein

### Bakgrunn:

Utvikling av kamerabasert system for biljard. Figure 1 viser et eksempel på en mulig implementasjon av et slikt system.



**Figure 1: Systemoversikt**

### **Prosjektbeskrivelse:**

I dette prosjektet bør følgende aktiviteter utføres:

- Få en oversikt over visionsystemer
- Vision-programvare: Få en oversikt over tilgjengelige rammeverk, verktøy og programvare for utvikling av visionsystemer, f.eks. NI Vision Development Software for LabVIEW og C# rammeverk. Andre alternativer kan være Python.
- Se på mulig maskinvare for et slikt visionsystem, som f.eks. PC, Raspberry Pi, NUC, datamaskin, osv.
- Spesifikasjon og utvikling av en prototype for detektering av elementer (som f.eks. baller, farge, plassering, m.m.) innen biljard ved bruk av kamera.

- Utvikling av applikasjoner for presentasjon av resultater.
- Vurdere ulike skytjenester og implementering ifm. dette, f.eks. Microsoft Azure plattformen.
- Vurder aspekter ifm datasikkerhet
- Vurder bruk av maskinlæring
- Testing av systemet

**Studentkategori:**

IA studenter

**Praktisk informasjon:**

Biljardbord, kamera, mm. er tilgjengelig for bruk i prosjektet.

**Signatur:**

Veileder (dato and signatur):

Studenter (dato and signatur):

## Appendix B

# Camera settings

Applying custom camera settings was crucial to achieve a good result in the form of video and pictures from the camera. Figure 0-1 below shows the camera settings that worked best in our test environment.

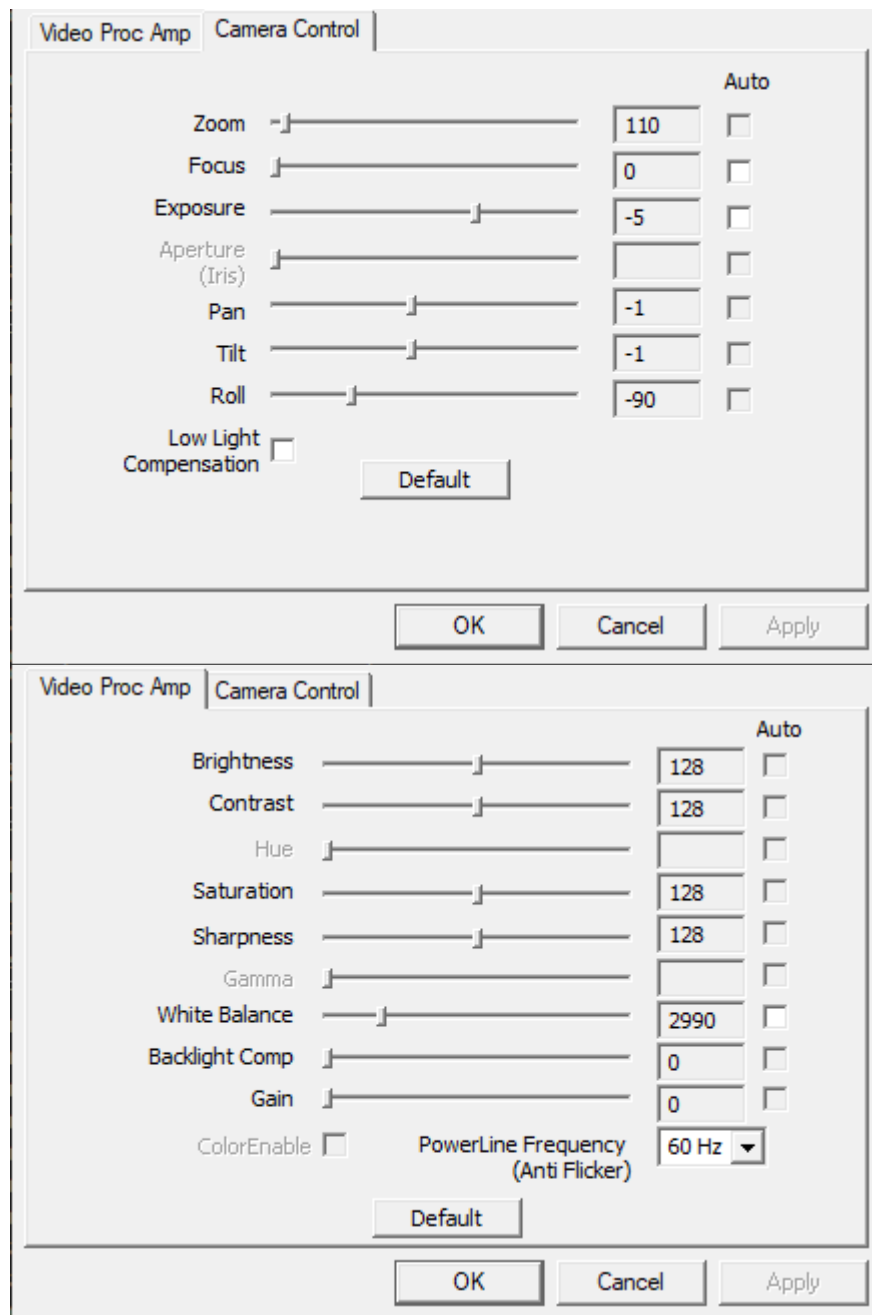


Figure 0-1: Camera settings menu and preferred values

## Appendix C

# Log of training sessions done to Custom Vision model

Table 0-1 Training log

Training-type	Date	Training-duration in hours	Number of Training-images
Quick training	12.1.2022	0.2	20
Quick training	13.1.2022	0.32	43
Quick training	14.1.2022	0.34	47
Quick training	14.1.2022	0.39	62
Quick training	17.1.2022	0.45	83
Advanced training	25.1.2022	2	88
Quick training	7.2.2022	0.49	98
Quick training	9.2.2022	1.25	176
Advanced training	24.2.2022	4 Hours	303
Quick training	7.4.2022	1.45	340

## Appendix D

# Vision system GUI user manual

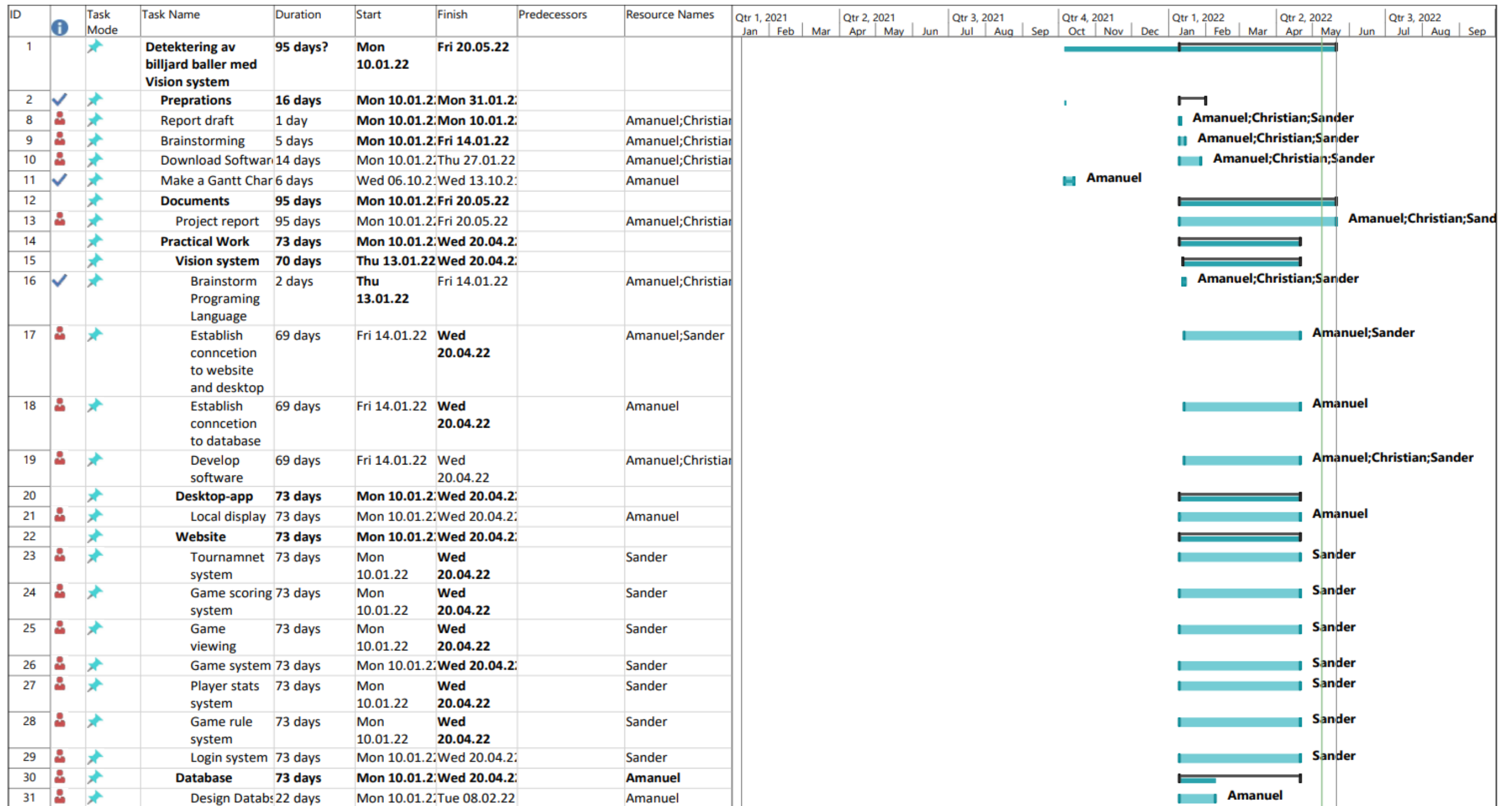
The following page will describe a short user manual. It is recommended that the user reads through this manual before using the system.

- Upon opening the program, a start page will be displayed.
- From the start page, the user is advised to select the correct camera from the dropdown menu.
- After selecting the camera, the user has three options:
  - If a game is being set up in the webpage, simply wait for the system to start automatically when the game is started from the webpage.
  - If the user wants to start a quick game without connection to the webpage, simply input the names desired, or leave them empty to automatically receive the names “Player 1” and “Player 2”. Press “**Start Quickgame**” to launch the game page
  - If the user wants to get a feel for the system without necessarily being near a camera or billiard table, the user can press “**Start Simulation**” to launch the simulation mode.
- With the game page now open, see that the camera is looking okay, and that the whole table is visible in the frame. If not, press “**TAB**” on the keyboard to open the camera settings and adjust them to achieve the optimal result. (See Appendix B for camera settings used in this project)
  - If the camera freezes after adjusting the settings, try pressing “**R**” to refresh the camera. If this does not work, try relaunching the application.
- With the camera ready, set the balls up in a triangle, and initialize the game by pressing “**SPACE**” on the keyboard.
- The game is now ready and have started. After each shot, press “**SPACE**” to advance.
- Once the black ball is pocketed and a winner is declared, the system will wait a few seconds before redirecting the user to the start page.
- Play again!



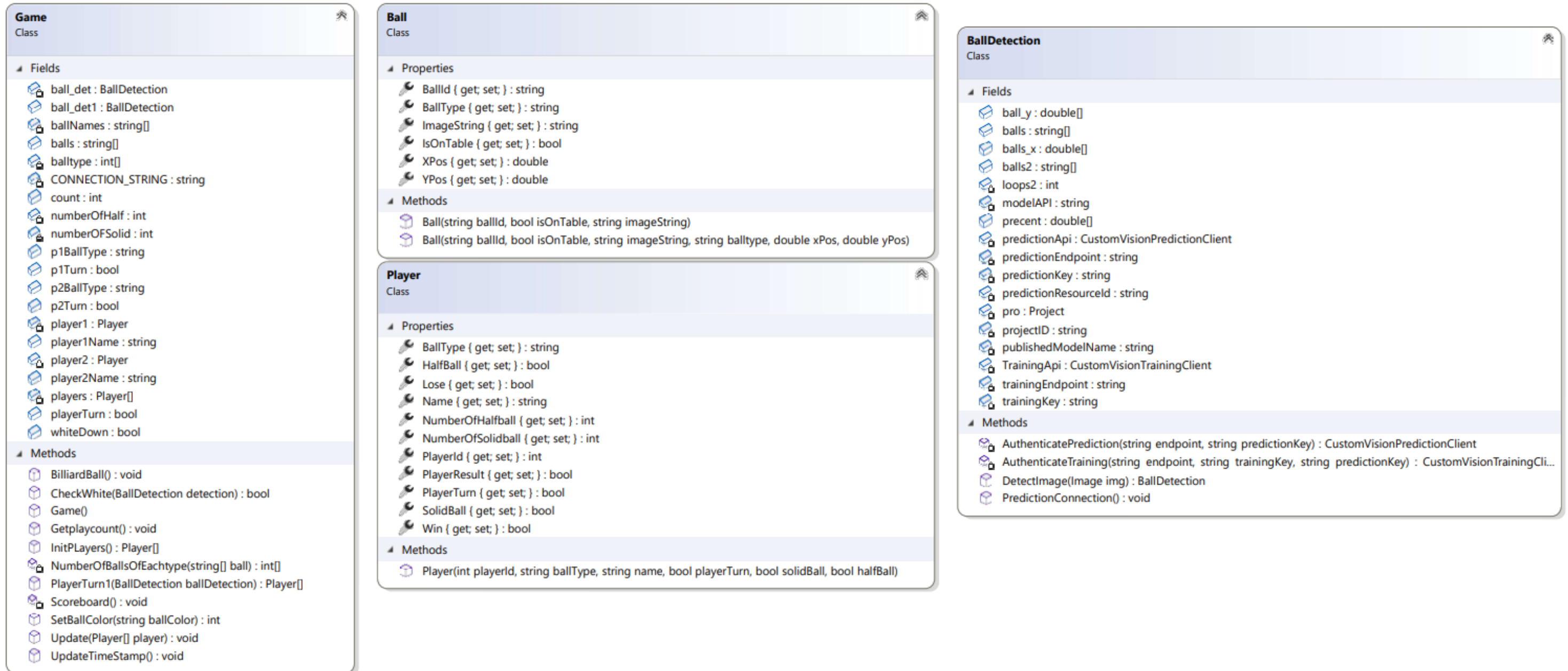
## Appendix E

### Gantt diagram



## Appendix F

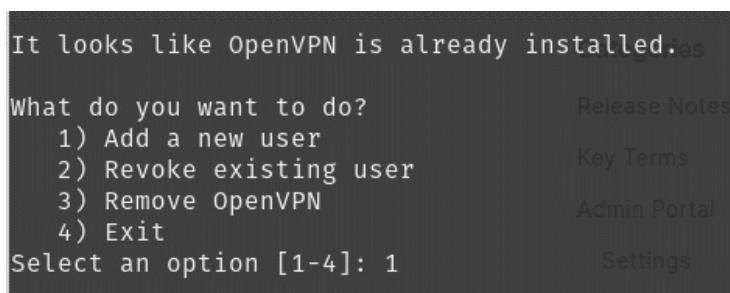
### Class diagram of the concrete classes



## Appendix G

# Adding a new vision system to web application guide

1. Login to the server where the OpenVPN server is installed and run the `openvpn-install.sh` file. This can be done by typing “`./openvpn-install.sh`” as the root user.
2. Select the first option to create a new user, see Figure 0-2

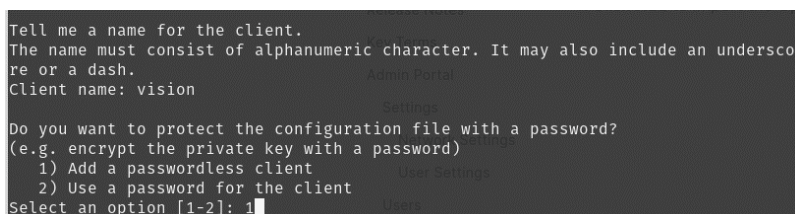


```

It looks like OpenVPN is already installed.es
What do you want to do?
1) Add a new user
2) Revoke existing user
3) Remove OpenVPN
4) Exit
Select an option [1-4]: 1
  
```

Figure 0-2 VPN menu adding new user

3. Then input a new username and press enter. After this, select option 1 to create a user without a password, see Figure 0-3



```

Tell me a name for the client.
The name must consist of alphanumeric character. It may also include an underscore or a dash.
Client name: vision
Do you want to protect the configuration file with a password?
(e.g. encrypt the private key with a password)
1) Add a passwordless client
2) Use a password for the client
Select an option [1-2]: 1
  
```

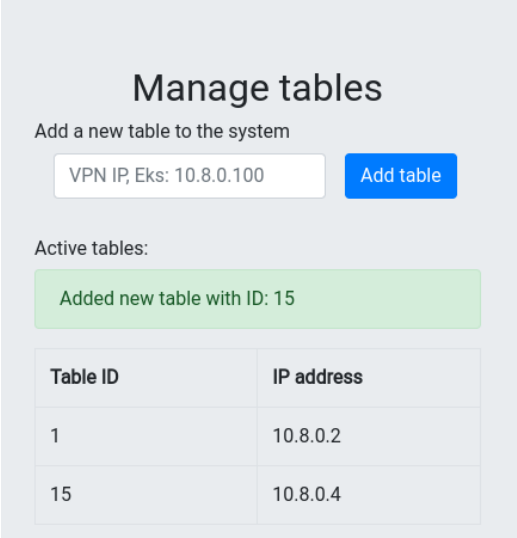
Figure 0-3 VPN menu select new username

4. Running the `ls` command will now show a new `.ovpn` file that contains all the necessary information for a client to connect to the OpenVPN server.
5. Transfer the new file to the vision system (this can be done with the `scp` command).
6. Import the new file in the OpenVPN client by pressing the `+` icon and selecting the `.ovpn` file. After importing the file, one should be able to connect to the server and communicate with the web application and database.

Opening a terminal and typing `ipconfig` should display your IP address (10.8.0.x). Take a note of this address because it is needed in the next step.

To test if the connection is established, one can try to ping the 10.8.0.1 address (OpenVPN server). If a response is received, the communication is working.

7. Log in with the administrator account on the web page and go to the admin panel. Under the manage tables section one can type in the IP-address from step 6 and press the “Add table” button. The new table should now be accessible to the users of the web page. See Figure 0-4.



The screenshot shows a web application interface for managing tables. At the top, the title 'Manage tables' is displayed. Below it, a subtitle reads 'Add a new table to the system'. There is a text input field containing 'VPN IP, Eks: 10.8.0.100' and a blue 'Add table' button. Below the form, a green notification box states 'Added new table with ID: 15'. Underneath, the section 'Active tables:' is followed by a table with two columns: 'Table ID' and 'IP address'. The table contains two rows: one with ID 1 and IP 10.8.0.2, and another with ID 15 and IP 10.8.0.4.

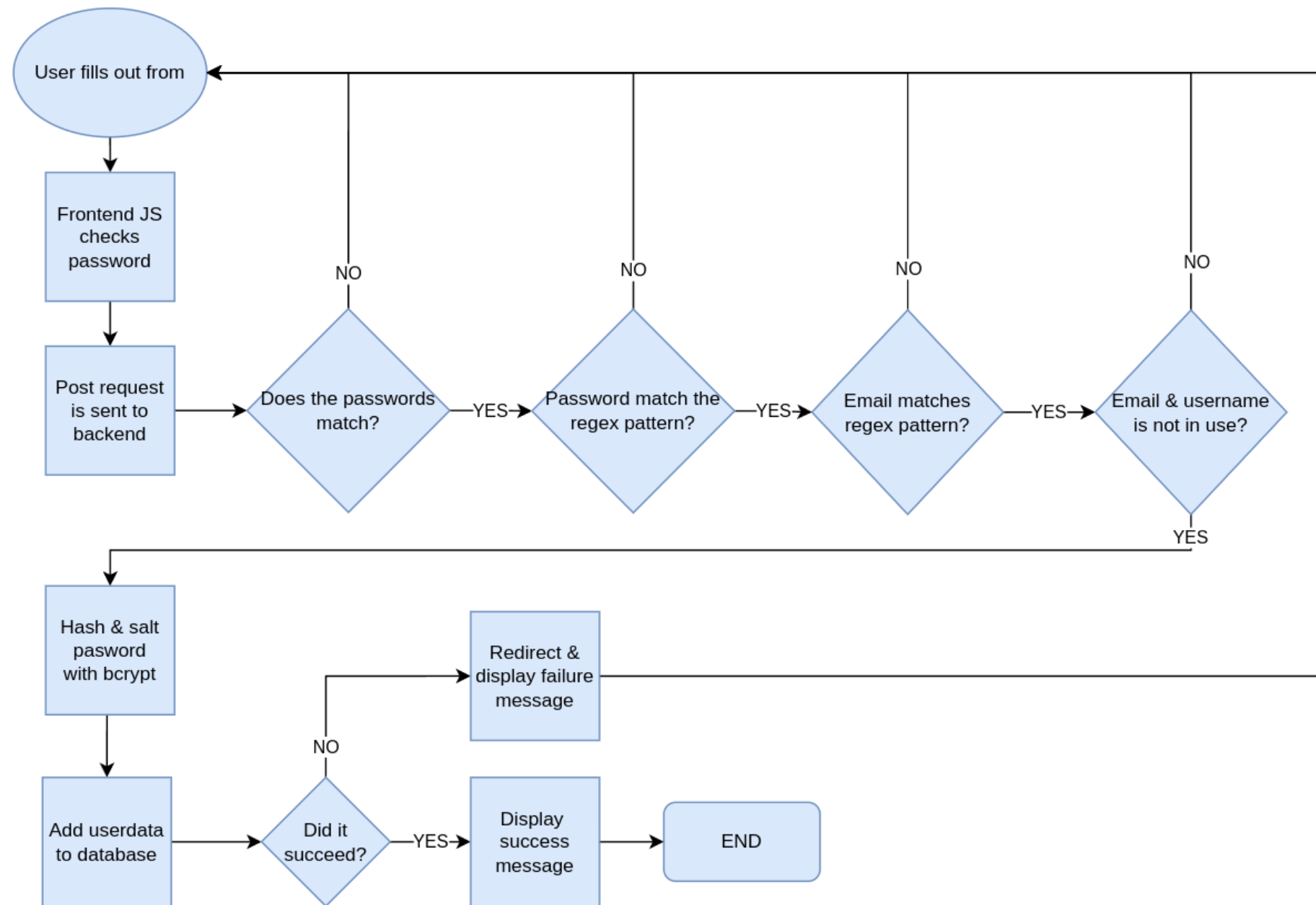
Table ID	IP address
1	10.8.0.2
15	10.8.0.4

Figure 0-4 Table menu in the admin panel, in the web application

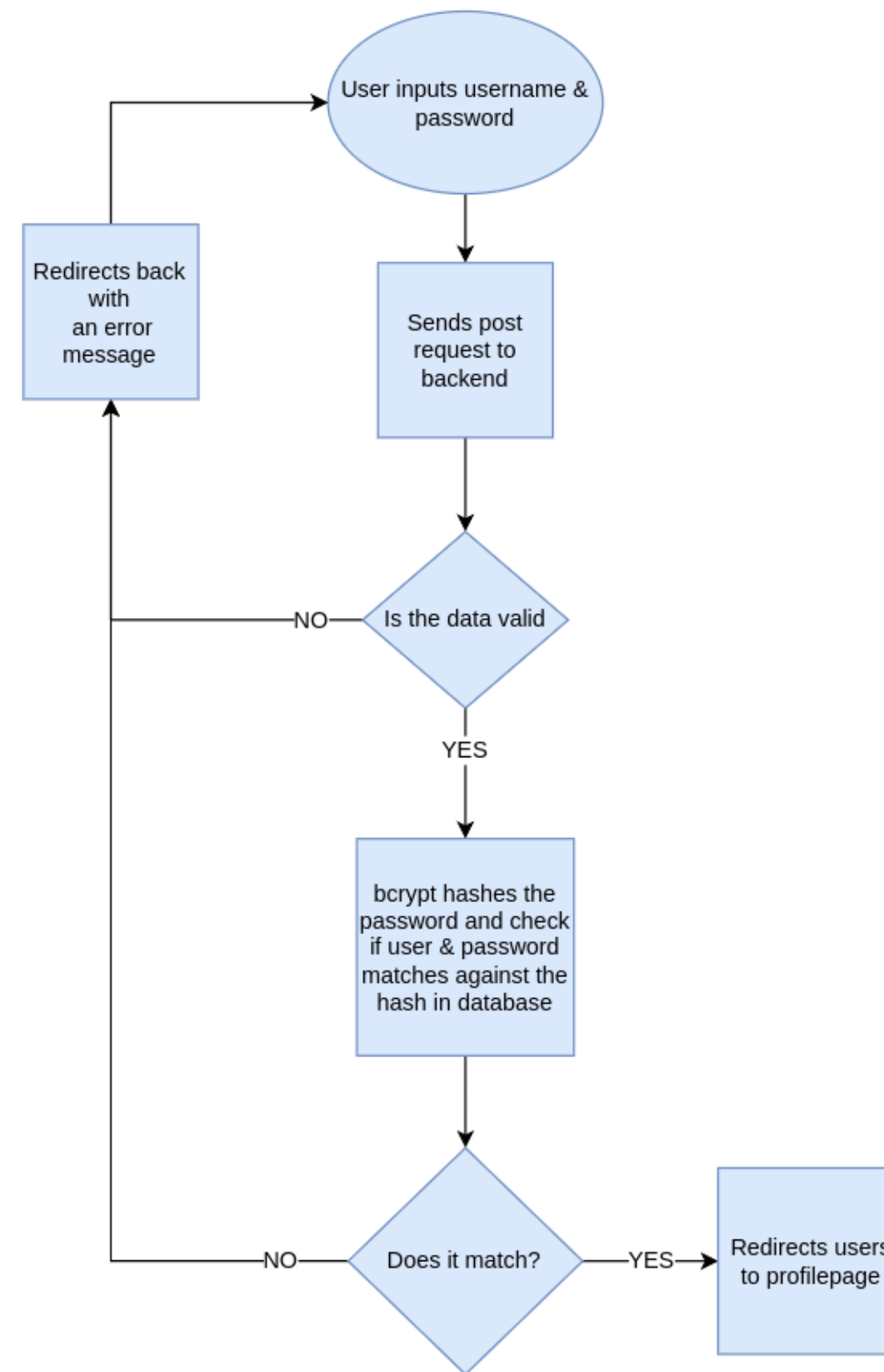
## Appendix H

## Flowcharts for the web application

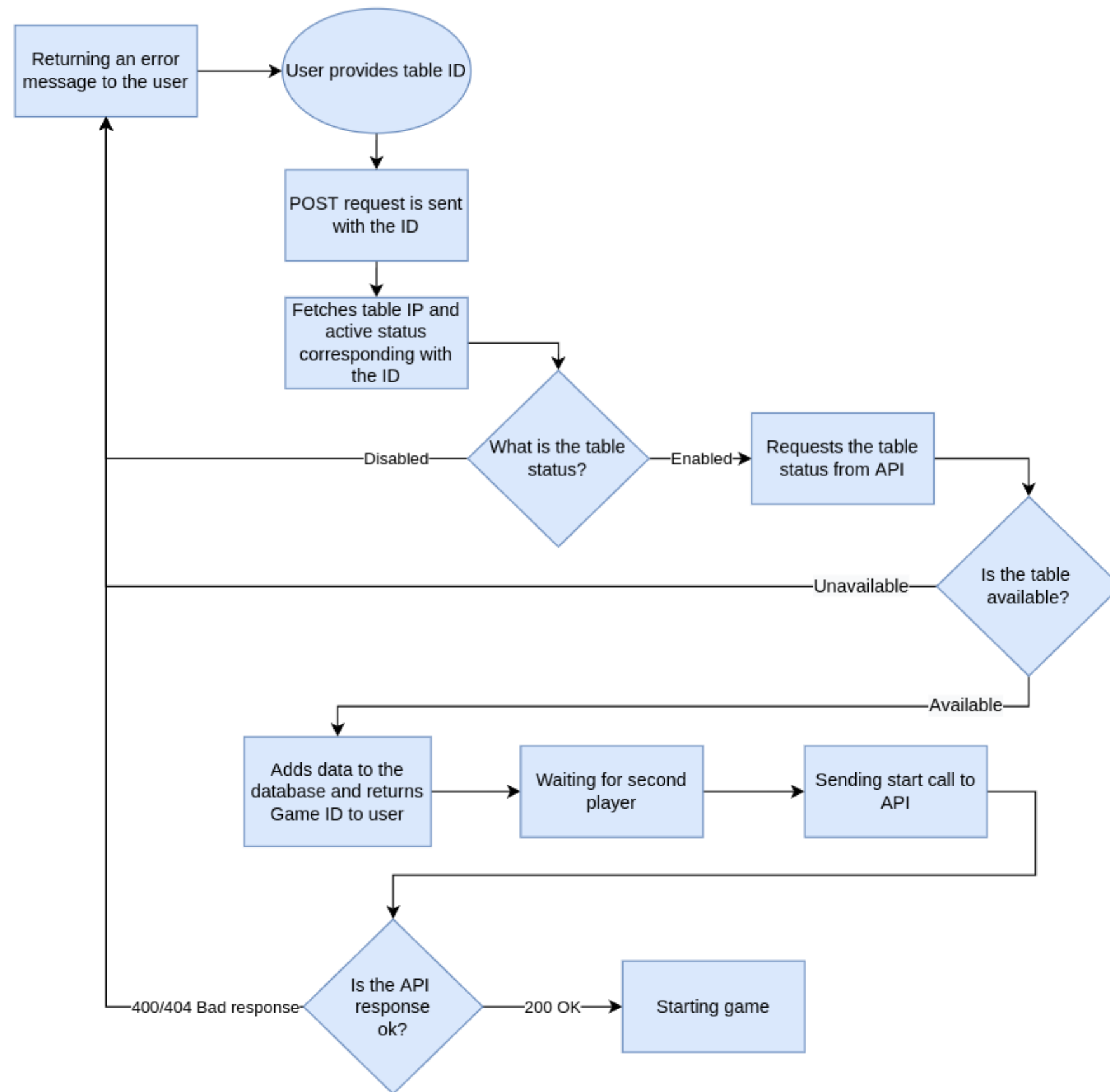
## Flowchart register page



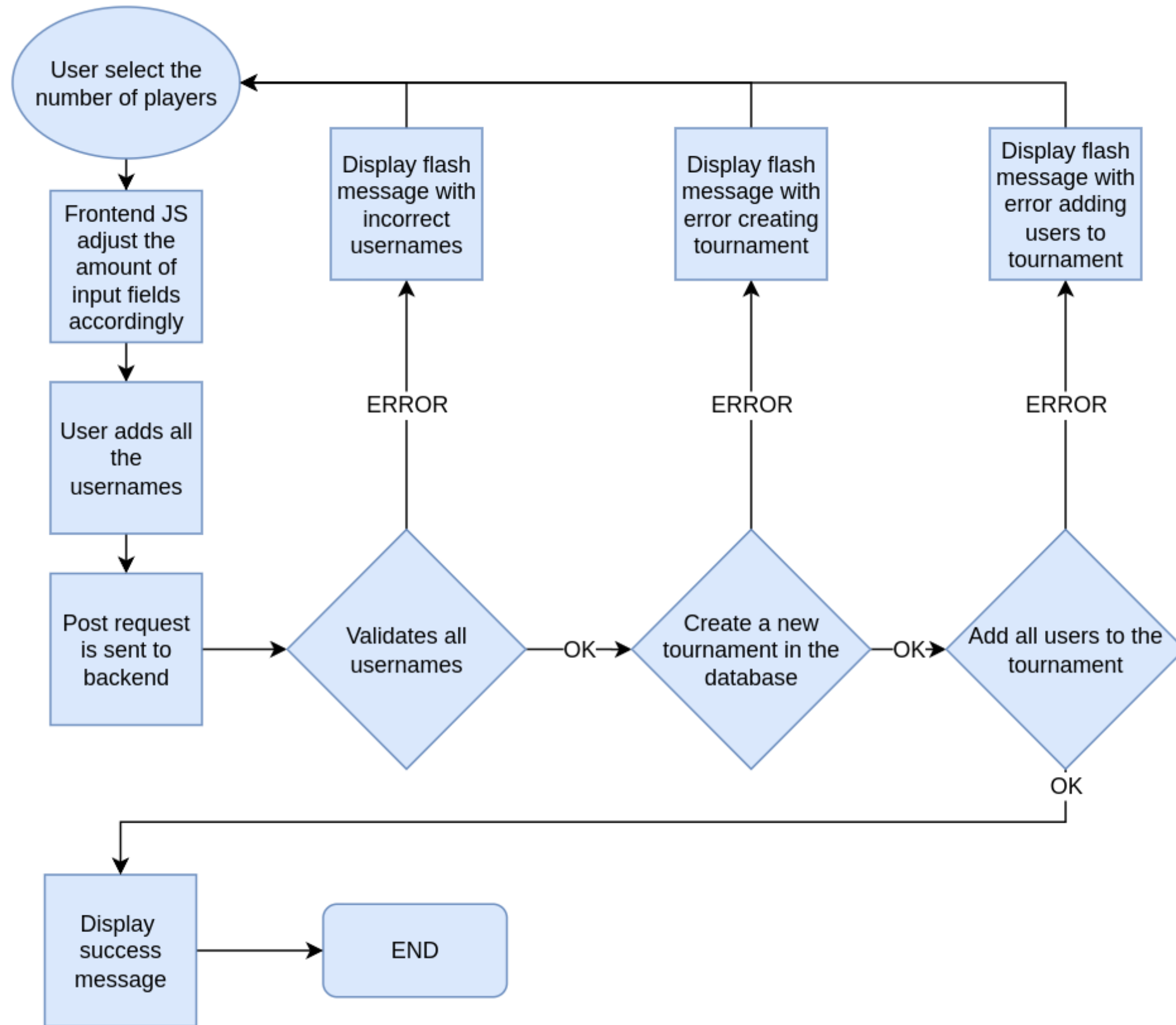
## Flowchart login page



## Flowchart game setup

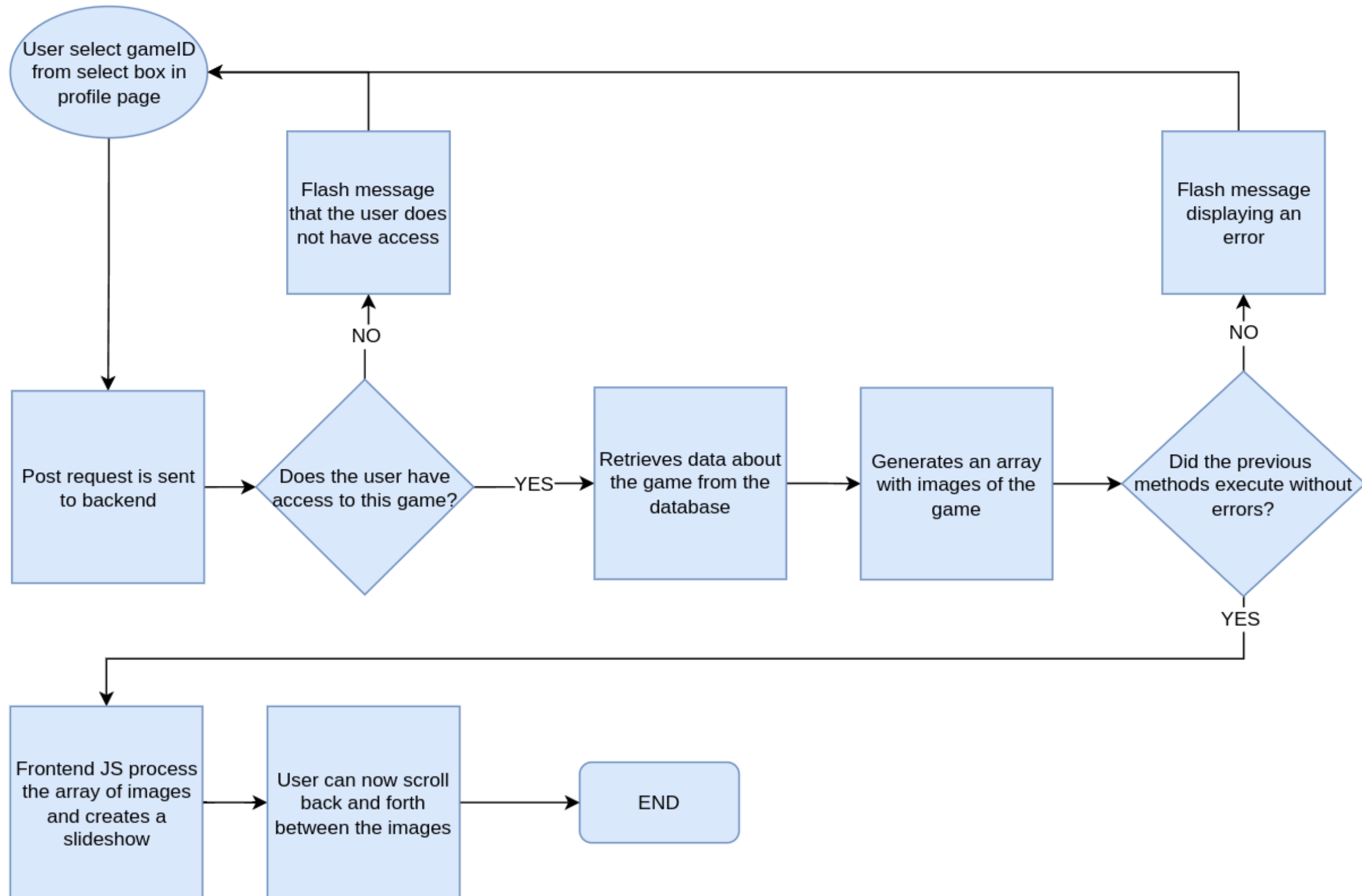


## Flowchart tournament creation





## Flowchart for displaying old games



## Flowchart for viewing live games

