Computer Vision for Identification and Measurement of Basketball Scoring

2024

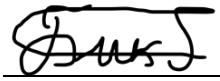
by

Gleb Bikushev
Under the supervision of
Dr. Kevin McDaid

School of Informatics and Creative Arts
Department of Computing Science and Mathematics

Declaration

“I hereby declare that the work described in this project is, except where otherwise stated, entirely my own work and has not been submitted as part of any degree at this or any other Institute/University”

Signed 

Name Gleb Bikushev

Date September 15, 2024

1 Abstract

In the rapidly advancing field of sports analytics, leveraging data-driven insights has become essential for optimizing athletic performance. This Master's dissertation presents the development of a web application designed to enhance basketball training by providing detailed feedback on shooting performance. The project utilizes computer vision techniques, specifically a custom-trained YOLOv8 model, to detect and analyze basketball shooting activities from user-uploaded training videos. The core aim is to automate the process of identifying and classifying basketball shots, providing players with immediate feedback and enabling them to track their performance over time.

The central research question focused on how computer vision could be adapted to recognize and analyze basketball shooting activities in a web-based platform. The dissertation addresses this by developing a system that detects players, basketballs, and hoops under various environmental conditions. Through careful dataset preparation and model training, the YOLOv8 model achieved high accuracy in detecting the necessary objects on the court. The framework for detecting throws and classifying successful shots was also developed and integrated into the web application, demonstrating reliable performance.

The project also explored the potential impact of quantitative feedback on training habits, though further research is needed to evaluate this effect. The web application, built with the FastAPI framework, allows athletes to upload videos, receive shot statistics, and track their progress, thus providing a scalable solution for basketball performance analytics.

In conclusion, the dissertation successfully demonstrates how computer vision can be incorporated into a web application for analyzing basketball shooting performance. The system offers a significant contribution to sports analytics by automating the detection and analysis of basketball shots, providing a foundation for future research and improvements in training efficiency and effectiveness.

2 Introduction

2.1 Brief introduction to problem

In the fast-evolving landscape of sports analytics, the integration of data-driven decisions has become crucial to advancing athletic performance. This Master's dissertation introduces a web application that leverages data analytics to improve basketball training experience and effectiveness by providing players with detailed feedback on their shooting performance. Utilizing computer vision techniques, this platform provides analysis of training session videos uploaded by the users, offering immediate feedback on their shooting performance, providing detailed statistics on their shots, including successful attempts and misses. Further, users can track the results of their training sessions, visually observing how their throwing accuracy changes over time.

2.2 Research questions

Main question:

1. How can computer vision techniques be adapted and incorporated in a web application to specifically recognize and analyze users' basketball shooting activities?

Supporting questions:

1. To what extent can the accuracy and reliability of a YOLOv8 model be improved through training to detect and track player, ball and basketball hoop objects on a basketball court for a camera positioned at specific points on the court?

2. Can an accurate and reliable framework for detecting throws and successful shots in the video be developed based on specifically trained YOLOv8?

3. Can a web application be developed with an integrated algorithm for detecting throws and goals that allows users to upload a basketball video and get statistics of shots from it?

4. How does quantitative feedback affect the training habits and score performance improvements of basketball players?

2.3 Problem definition

The traditional approach to tracking sports performance, namely counting the number of shots and goals in basketball, often relies on subjective observation and manual data recording, which are inherently limited by human error and scalability issues. This project proposes a solution to automate the process using a computer vision system that not only captures every detail but also provides quantitative feedback that is both accurate and immediate.

2.4 Project relevance in the field of Data Analytics

This project contributes significantly to the field of data analytics by applying deep learning and computer vision to the domain of performance monitoring in basketball, enhancing the precision and efficiency of sports training analytics. By transforming raw video data into actionable insights utilizing the YOLOv8 model and score detection algorithm, the project extends the application of data analytics to the basketball shots performance monitoring within training session. In addition, the decision of adding clear and concise visualization of the users score performance using the line graphs and bar charts makes the project relevant to the Data Analytics domain.

2.5 Core technology, architecture and research processes used

2.5.1 Data collection

The initial stage involved the collection of a diverse and robust dataset necessary for training an accurate YOLO model. This dataset includes various basketball game scenarios captured from YouTube Shorts videos, which provide a rich source of diverse environments and play styles. In order for the model to be able to identify the necessary objects on the basketball court from specific angles and specific height values, external participants were involved. They provided videos from their throwing practice sessions recorded with the phone located both on the ground and on some height using a tripod. Further, all the YouTube Shorts videos and videos from external participants were uploaded on the Roboflow, and frames were extracted at a frequency of around 10 frames per second to create a large dataset. Using the Roboflow labeling tool, each frame was thoroughly labeled to detect three objects on the basketball court: the player, the ball, and the basketball hoop.

2.5.2 YOLO Model training

The core of the technical implementation involves training a custom YOLOv8 model. This stage was approached by first using a large pre-trained YOLOv8 model from Ultralytics as a base to benefit from its existing learned features, which were then fine-tuned with our custom dataset through 100 training epochs. This approach ensured that the model not only learned the general characteristics of the objects of interest but also adapted to the specific nuances present in basketball training videos.

2.5.3 Throws and goals detection algorithms

Post-training, the focus shifted to developing a sophisticated algorithm capable of interpreting the video data to determine the number of shots, and distinguishing between successful shots and misses. Having the model that accurately detects all required objects on the basketball court, the software approach for throws and goals detection was developed, which is based on the coordinates of bounding boxes of a player, ball and basket from each video frame and the interactions between these objects and and special rectangular zones built near the basketball hoop to determine the presence of the ball in them.

2.5.4 Web application development

The final stage of the project involves creating a cloud-based web application. This app will allow athletes to upload training videos and receive detailed performance statistics. It will be powered by the FastAPI Python framework, processing videos and managing user data within a PostgreSQL database. SQLAlchemy will bridge Python and the database, allowing the use of Python classes over raw SQL. This approach supports heavy video processing demands and ensures secure and efficient data management.

2.6 Hypothesis for a solution

The hypothesis behind this study is that an accessible, easy-to-use web application powered by a custom-trained YOLOv8 model will provide accurate analytics on basketball shots. This system is expected to significantly improve the training quality by offering immediate feedback of player throwing accuracy, allowing players to clearly track their progress in improving their throwing skills. The camera setup's simple design, which requires minimal installation and utilizes common mobile devices, is anticipated to increase the adoption rate among basketball players of varying skill levels.

2.7 Structure of the report

Contents

3 Literature Review

3.1 Computer vision in different domains

Due to the rapid development of artificial intelligence technologies, specific areas including computer vision and computer image processing, have introduced innovations across a broad array of disciplines. Computer vision is a crucial aspect of artificial intelligence, aiming to empower computers in comprehending and interpreting visual data from images or videos [29]. It involves object classification, detection and segmentation. The following studies demonstrate the use of computer vision in various fields.

The paper [32] showcases how computer vision techniques can be applied to the study of animal ecology. Describing in detail the use of computer vision for observing, counting and identifying species in their natural habitat, it highlights the enormous potential of computer vision, bridging the gap between technology and ecological research.

Another example of implementing computer vision in different domains is paper [2]. By demonstrating how computer vision can enhance security systems through automatic face detection and movement recognition, this paper contributes valuable insights into practical applications of computer vision technologies.

3.2 Computer vision in sports analytics

In the field of sports and sports analytics, computer vision has also made a great contribution, offering considerable and in-depth insights that previously can not be obtained. This detailed level of analysis, that computer vision offers in the realm of sports analytics, helps coaches develop strategic game plans, helps players improve their skills by providing feedback on their moves [31], and increases audience engagement through advanced statistics and visualization [10].

For instance, the paper [26] discusses current commercial applications that are using computer vision for sports analysis. The authors consider team sports such as football (soccer), basketball, and American football, where player and ball tracking are crucial for tactical analysis and coaching. Also they involved racket sports including tennis and badminton, where ball tracking systems are used for officiating and enhancing broadcast coverage. In the context of individual sports, the authors mentioned motion capture for training professional athletes, where precise movements are analyzed for performance improvement.

The paper [19] provides a comprehensive review of computer vision techniques in sports (specifically in soccer, basketball, cricket, and badminton) by discussing and summarizing the variety of published works about application-specific sports-related problems. Broadly speaking, these studies focus on tasks like player detection and tracking, balls and players trajectory prediction in real-world sports scenarios, as well as identifying the tactics used by the team and categorizing different sporting events. For example, having the problem of detecting and tracking a player and a ball in soccer, the proposed methodology was using YOLOv3 and SORT algorithms (will be further discussed in section "YOLO"). This approach achieved a high tracking accuracy of 93.7% across various object tracking accuracy metrics. It operated at a detection speed of 23.7 frames per second (FPS) and a tracking speed of 11.3 FPS. Despite its effectiveness in handling partial occlusions and tracking players and the ball as they reappear after being momentarily out of frame, the methodology struggled with significant occlusions, where it failed to maintain accurate tracking. Additionally, the approach of combining such techniques as YOLO and Joy2019 was used to track ball movements and classify players in basketball games with the personal numbers on their jerseys. It achieved a precision of 74.3% for jersey number recognition and a recall of 89.8% for player recognition. However, the system faced issues when players overlapped, since YOLO mistakenly identified overlapping players as a single entity.

3.3 Computer vision in basketball

In basketball analytics, computer vision approaches have been used for ball trajectory analysis, players motion capture, object detection (player, ball, basketball hoop) and event categorization. [7]. Event categorization refers to the process of classifying various actions and occurrences during a basketball game into distinct categories, such as shots, passes, and fouls, using computer vision techniques. Better training techniques and tactical insights for teams and coaches have been made possible by the greater

understanding of the game and player performance that has resulted from advances in computer vision and artificial intelligence, in general.

For example, in the article [15] the authors utilized a position tracking algorithm based on action division in the field of artificial intelligence to determine the position of the basketball and identify classified actions of basketball players. These actions are decomposed into instantaneous and continuous operations, generating movements of upper (arm movements) and lower (leg movements) limb units based on limb angle changes in the movement process. The study reported high accuracy and recall rates, with the four different Machine Learning (ML) classification algorithms such as Decision Tree (DT), Naive Bayes (NB), Support Vector Machine (SVM) and Artificial Neural Network (ANN), demonstrating an average accuracy of movement recognition between 96.99% and 99.19% for lower limb movements, and between 84.89% and 92.19% for upper limb movements, including activities like dribbling, walking, running, and others. Therefore, this paper enables coaches to easily identify incorrect postures and allows for immediate correction, enhancing the training quality. However, the authors mentioned that there is no objective data to support the standards of basketball dribbling posture internationally. This suggests a future improvements of the research in the form of developing more standardized and objective criteria for evaluating basketball dribbling postures.

Another study [33] proposes a method that significantly improves the accuracy of basketball action recognition by addressing the limitations of the original C3D (Convolutional 3D) convolutional neural network, which struggled with focusing on keyframes and extracting sufficient feature information. The new method leads to an average accuracy of posture recognition of basketball players of more than 97%. Thus, by enabling precise control over athletes' sports postures, the authors significantly improve training effectiveness.

Based on the above-mentioned reasons, ball and basketball players detection and tracking in real time or through the analysis of provided video fragments have been the subject of different studies that have employed a variety of object detection techniques and algorithms.

3.4 OpenPose

One of the widely used techniques for basketball players detection and their pose estimation is **OpenPose** [20]. A special feature of this algorithm is that it recognizes multiple persons not just as bounding boxes; it is real-time human pose estimation algorithm, that capture 2D positions of a person's joints and skeleton wireframe of the body, and then draw lines between these keypoints, recreating person's exact pose (Figure 1).

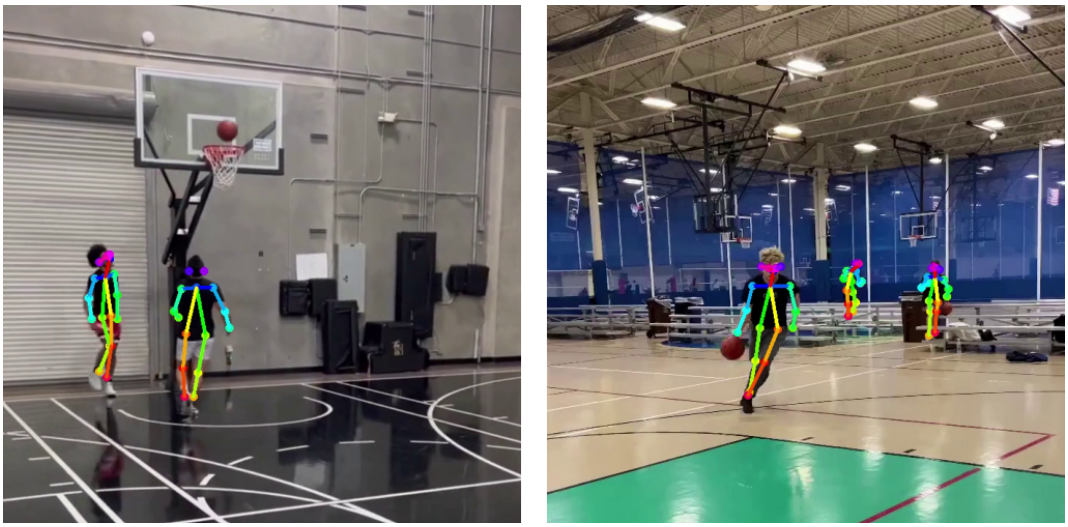


Figure 1: Examples of video frames processed by OpenPose

According to [5], the algorithm architecture is designed in such a way that the process of human pose estimation is divided into two branches that work concurrently: the first branch utilizes a feed-forward network to produce a series of 2D confidence maps that identify the locations of body parts of all individuals on the picture, and the second branch generates a series of 2D vector fields of body part

affinities, which represent the level of association between these parts. These confidence maps and affinity fields are then combined and analyzed through a greedy inference process, which ultimately outputs the 2D keypoints for every individual in the image.

Regarding the application of OpenPose technique in basketball, the study [6] can be a good example of that. The authors of this paper proposed a video-based basketball shooting prediction and posture suggestion system that leverages the OpenPose system for detecting human joints and, as a result, pose estimation, without the need for attaching sensors to the athletes. Concerning the experimental results, the authors evaluated the five curve similarity algorithms for predicting the basketball shooting outcomes based on the players postures for two different groups: general basketball class students and university team players. The authors reported that the highest accuracy rates were achieved when taking into account the overall differences in shooting postures among all players, rather than individually, and amounted to 87% for basketball class students dataset and 50% for university team players. Moreover, the study points out the potential for future improvements by incorporating more robust features like depth information or moving speed for better shooting outcomes predictions. Regarding the posture suggestion system, the results showed that it can effectively adjust incorrect poses and provide recommendations for players.

Additionally, the study [25] explores how OpenPose can help to avoid occlusion of human bodies and improve their detection efficiency, compared with using only color information, while detecting players on the basketball court. Specifically, the authors employed cameras placed at three different angles to capture comprehensive visual data, reducing the chances of player occlusion. The authors conclude that applying OpenPose with combination of these cameras clearly reduced the occlusion rate between basketball players and its use to identify players is very effective compared to using team uniform color alone. Particularly, traditional methods resulted in occlusion rates of 0.14% for three players, 0.26% for two players, and 0.42% for one player, with a 0.21% rate when no occlusion occurred. In contrast, the integration of OpenPose effectively reduced the occlusion rate to nearly zero, as it consistently allowed players to be detected from at least two different viewpoints, effectively avoiding occlusions in all tested instances. However, the effectiveness of the system relies heavily on the availability of multiple camera viewpoints to resolve occlusions. This could limit the method's applicability in environments where only one or two cameras are available.

3.5 YOLO

Another popular object detection algorithm in the field of computer vision, that is worth emphasizing, is **YOLO** (You Only Look Once). It marks objects in the picture using rectangular bounding boxes with corresponding class labelling and its confidence score (Figure 2).

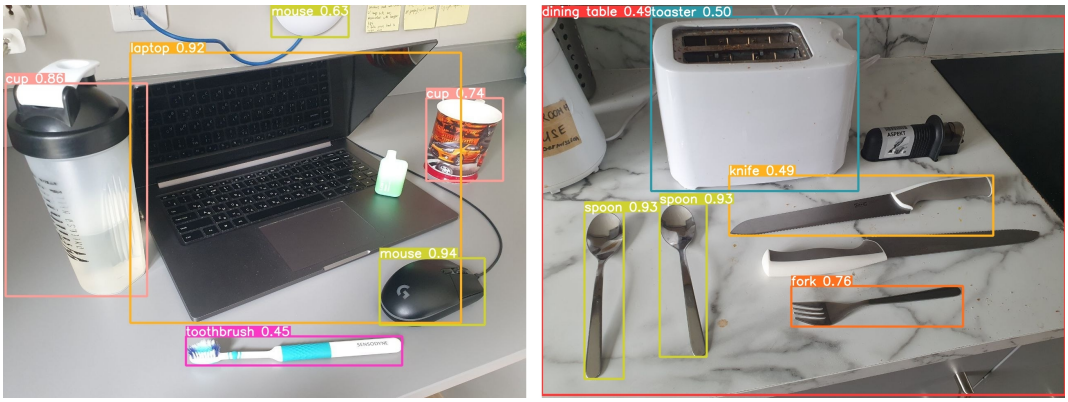


Figure 2: Examples of pictures processed by YOLOv8s model

The authors in the paper [23] first proposed this novel method. Generally speaking, contrary to other approaches, they frame detection as a single regression problem and, therefore, don't build a complex pipeline. More specifically about the YOLO method, the input image is first split into a grid where each cell predicts multiple bounding boxes and their confidence scores. Each bounding box includes the box center coordinates (x, y), its dimensions like height (h) and width (w), and a confidence score (c) that indicates the likelihood of an object being present and the accuracy of the

box. Simultaneously, each grid cell predicts conditional class probabilities ($\Pr(\text{Class}_i \mid \text{Object})$) based on the presence of an object. During detection, these probabilities are combined with the confidence scores to produce class-specific confidence scores (C) for each box, reflecting both the presence of a class within the box and the precision of the fit. Thus, each bounding box represent following vector:

$$\text{bounding box} = (x, y, h, w, C, \text{class})$$

where the categorical variable "class" reflects the specific class to which the object belongs. This method allows YOLO to predict multiple bounding boxes and class probabilities across the entire grid at once, simplifying object detection. This makes the proposed algorithm the fastest general-purpose object detection approach that stands out for its exceptional balance of speed and accuracy, compared to other classifier-based approaches. Moreover, since the model is supposed to be trained directly of full images, meaning it uses entire, unmodified images during its learning process, this method significantly simplified the task of training custom detection model. This reason has served as the impetus for many projects and studies in the field of object detection, particularly in such sports game with fast movements of the players and the ball as basketball.

Before discussing in detail some of the papers devoted to the topic of applying YOLO technology in basketball analytics, it is worth briefly mentioning the story of how YOLO has developed and improved over time:

- Introduced in 2015, YOLO quickly became popular for its impressive speed and precision.
- In 2016, YOLOv2 [21] was unveiled, enhancing the original with features like batch normalization, anchor boxes, and dimension clustering.
- YOLOv3 [22], released in 2018, advanced the architecture further by integrating a more robust backbone network, multiple anchor boxes, and spatial pyramid pooling.
- Launched in 2020, YOLOv4 [4] brought new features such as Mosaic data augmentation, an anchor-free detection head, and an innovative loss function.
- YOLOv5 [27] enhanced the model's capabilities with additions such as hyperparameter optimization, integrated experiment tracking, and automatic export to popular export formats
- In 2022, Meituan released YOLOv6 [16], which now powers many of its autonomous delivery robots.
- YOLOv7 [30] introduced additional capabilities like pose estimation for the COCO keypoints dataset.
- YOLOv8 [28] is the latest version of YOLO by Ultralytics, which continues to refine the series with significant advancements, enhancing performance, flexibility, and efficiency. YOLOv8 supports a broad array of vision AI tasks, such as detection, segmentation, pose estimation, tracking, and classification, making it adaptable to a wide range of applications and industries.

Let us now examine several studies regarding the application of the YOLO algorithm in basketball.

In the study [1], authors introduced a multi-detection and tracking framework capable of precisely detecting and tracking basketball players using only broadcast videos. This framework is based on YOLOv2, state-of-the-art real-time object detection system, and SORT [3], a simple but accurate object tracking algorithm. The framework was trained and evaluated on NCAA basketball dataset with 8787 images for training and 1000 images for testing set. The results demonstrates that the proposed approach achieved an Average Precision (AP) of 0.89 for the standard criteria (IOU0.5) and 0.63 for the harder criteria (IOU0.7). In other words, when using a threshold of 0.5 for the Intersection Over Union (a measure of overlap between the predicted bounding box and the ground truth), the model achieves an average precision of 89%, while using a IOU threshold of 0.7, a stricter criterion, the AP decreased to a value of 63%.

In [12] the authors consider multiple tracking of basketball players with basketball court monitoring,, aiming for good accuracy and real-time operational efficiency. The authors compare different baseline detectors such as Faster-RCNN [24] and YOLOv3 to assess their impact on tracking accuracy in the basketball court scenario. For this specific task, they developed a new basketball court dataset

that includes 15 labeled sequences of frames (3000 each) from 15 basketball videos with a duration of 2 minutes. Using various evaluation metrics, results demonstrates that Faster-RCNN surpasses YOLOv3 in terms of accuracy, but lacks in execution efficiency, since YOLOv3 achieves near real-time speed in players detection, while Faster-RCNN detection speed is just 3 frames per second.

The paper [34] focuses on automatically classifying players, tracking ball movements and analyzing pass relationships in basketball games using YOLO, as the core of the presented system. Pre-trained with Microsoft's COCO dataset [17], YOLO was trained on the 16,000 ground truth boxes of balls, players and jersey numbers from 1204 video frames. This made it possible to create a model that detects the above-mentioned objects with high efficiency. It classifies detected objects by identifying players and recognizing their jersey numbers, which is crucial for tracking individual player movements and ball possession across frames. Also, the study adapted YOLO to track players and the ball even when players move out of the frame or overlap each other. This adaptation involved using contextual information from previous or subsequent frames to maintain continuity of player tracking despite difficult conditions such as rapid camera shifts.

The study [11] went a step further and used YOLO for a camera-based basketball scoring detection method. The YOLO model was trained to detect the position of the basketball hoop within the video frame, which is a key element in determining the scoring condition. The dataset included 3500 images among which are the photos of the basketball court, the surveillance images of the basketball court and the screenshots of basketball matches. This basket detection is crucial because the system must accurately identify the hoop's location to determine if a score has occurred. After the hoop's location is identified by YOLO, the system used frame difference-based motion detection to determine if the basketball has passed through the hoop, thus confirming a scoring event. The paper reports that the YOLO-based hoop detection combined with motion detection runs in real-time and achieves satisfactory scoring detection accuracy (88.64% among 5 test videos). The experiments conducted on real-scenario basketball court videos validated the effectiveness of the proposed method.

3.6 ByteTrack

The advancements in tracking algorithms such as **ByteTrack** complement the capabilities of YOLO by enhancing tracking accuracy and efficiency in complex scenarios like sports. ByteTrack, which utilizes a simple yet highly effective association mechanism based on the intersection over union (IoU) metric, builds upon YOLO's detections to maintain robust track identities even in challenging conditions where occlusions and fast movements are common. This is particularly valuable in sports analytics, where accurately tracking the rapid movements of players and balls is crucial.

In the study detailed in [35], the authors first present ByteTrack as an extension of YOLO object detectors, specifically designed for addressing the limitations of Multi-object tracking (MOT). Compared to traditional tracking algorithms that basically discard low-confidence detections to avoid false positives, ByteTrack leverages these detections to improve tracking continuity and reduce identity switches. The core of ByteTrack's methodology is the association of detection boxes across frames. ByteTrack employs the Kalman filter to predict the state of tracked objects (e.g., position and velocity) and uses the Hungarian algorithm to associate detections with existing tracks based on the Intersection over Union (IoU) metric. This step primarily utilizes high-confidence detections. After associating high-confidence detections, ByteTrack uniquely integrates low-confidence detections into existing tracks. This is done by checking if these low-confidence detections have a sufficient IoU overlap with the predicted states of existing tracks. If a match is found, ByteTrack adds these detections to the respective tracks, which helps in maintaining track continuity especially in scenarios where objects are occluded or only partially visible.

As for the object tracking performance, the authors used widely recognized datasets in the multi-object tracking community, such as MOT17 [18] and MOT20 [9] from the MOTChallenge. These datasets include various video sequences with challenging scenarios, particularly with different levels of crowd density and complex occlusions like pedestrians on crowded streets captured with static and moving cameras. Thus, being tested on more densely crowded MOT20 dataset, the results showed that among all other object tracking methods ByteTrack leads with the highest scores in MOTA (77.8%), IDF1 (75.2%), and HOTA (61.3%), indicating its robustness in maintaining accurate track identities and associations.

The paper [8] can be an appropriate example of how ByteTrack helps in tracking fast-moving objects which is particularly common in sports scenes like football and basketball. The authors presented a

new dataset called SportsMOT, that is largely superior to the MOT17 dataset in terms of the number of frames and bounding boxes, and includes dynamic scenes from 3 sports categories, namely football, volleyball and basketball. Being tested on this proposed dataset, ByteTrack demonstrated the decent results of tracking multiple athletes on the scene with the values of 64.1% for HOTA, 95.9% for MOTA and 71.4% for IDF1.

3.7 Practical applications of basketball analytics

This section will be dedicated to actual commercial products in the field of basketball analytics.

For example, HomeCourt [13] is the interactive basketball mobile application, that captures the video of individuals throwing the ball into basketball hoop and gives them instant guided feedback of their shooting performance. This feedback includes the number of attempted and successful shots, their trajectories and positions on the court, providing users with the detailed map of the court with the marked points of shots positions. Another application feature for basketball enthusiasts that could help in improving ball handling skills is dribbling workout sessions. They work in such a way that the users, while dribbling with the ball, look at the screen of their device, which shows him an image from the front camera, where colored points appear in different positions on the screen, which the user has to trigger with his own hands. Each successful trigger adds score to the final result. At the end, the user receives feedback of his ball handling skills based on this exercise. Moreover, after capturing the statistics for every video from the user's practice throwing sessions, the application is capable of building the detailed visualisations of how users shooting accuracy performance changes over time, providing a complete picture of their progress. Despite the fact that the application does not use any physical sensors and does not require any specific, high-quality camera setups, expanding opportunities for basketball enthusiasts to start using this product, the app's creators inform about some limitations and specifications for the users. For example, it works only for hoops with the net and the color of the ball has to be close to orange. Despite the fact that the creators of the application do not disclose the specific technology by which they process video and get detailed shooting statistics from it, it can be assumed that computer vision technology lies at the core of their method.

The application Swish Hoop [14] uses another approach of solving the problem of successful basketball shots detection by integrating sensor technology directly into the basketball hoop. Swish Hoop's smart hoop system uses a series of sensors around the rim and base of the hoop to capture real-time data on every shot made. This data includes the speed of the ball, its angle of entry, and whether the shot was successful or not. Additionally, the system can detect the specific location from where each shot was taken, enabling it to provide players with precise feedback on their shooting patterns and accuracy from different court positions. One of the standout features of Swish Hoop is its interactive feedback mechanism. Once a session is completed, players receive instant feedback through the Swish Hoop mobile app, which displays detailed statistics and visualizations of their performance. These visualizations include heat maps of shooting accuracy across different zones of the court, allowing players to identify strengths and areas for improvement in their shooting technique. Despite its advanced capabilities, the creators of the product claim that the system primarily relies on a robust Wi-Fi connection to sync data between the hoop sensors and the mobile app, which may limit its use in areas with poor internet connectivity. In addition, the presence of a basketball hoop net is the mandatory condition, since the sensor is supposed to be installed exactly on the net.

3.8 Conclusion

Throughout this literature review, the huge impact of computer vision technologies across various domains has been thoroughly explored. In the realm of sports analytics, particularly basketball, computer vision techniques such as OpenPose, YOLO and ByteTrack has led to significant improvements in tasks like players detection, tracking and motion capture, a ball trajectory analysis and etc., providing previously unreachable analytical depth that enhances coaching strategies, more efficient players training and, subsequently, their general performance.

Innovative commercial applications like HomeCourt and Swish Hoop exemplify how these technologies are being successfully translated into user-friendly products that make high-level sports analytics available to a broader audience. This research work done sets a clear direction for this dissertation project, giving state-of-the-art approaches and solutions for the problem of shots and goals detection in basketball, based on the provided video.

This general problem is decomposed into several subtasks, starting with the accurate detection of such objects on the court as the player, the ball and the basketball hoop. The previously mentioned YOLO approach and its latest released and most relevant model YOLOv8, is perfectly suited for this purpose and will be used as the core technology of the shots and goals detection algorithm. Having the precise data on the objects location and size using the YOLO-generated bounding boxes of these objects, the task will be reduced to coming up with the logic for the algorithm for throws and goals detection using the coordinates of bounding boxes.

Since it is anticipated that we will process and analyze video recordings from individual basketball throwing training sessions, which means there won't be many players and a dynamic game on the video recordings, we exclude the task of handling objects occlusions. Therefore, we will not apply ByteTrack approach.

In addition, since OpenPose is mainly used for human pose recognition, and the players precise posture detection is not supposed to help in the task of scoring detection, its use is not of interest for this project.

4 Exploration of Data and Methods

This section will be divided into different stages of the project, and for each stage all methods will be thoroughly discussed and explained.

4.1 Dataset creation

The initial phase of my project focused on assembling a diverse and robust dataset essential for training an effective YOLO model for basketball analysis. The variety in the dataset is crucial as it impacts the model's ability to accurately detect key elements such as players, basketballs, and hoops.

4.1.1 Data collection

Speaking about dataset content, I sourced video data from two primary channels: YouTube Shorts, which provides a wide variety of basketball gameplay scenarios, and videos recorded by external participants from their basketball training sessions. All included YouTube Shorts videos (14 videos) were recorded with camera installed on some height using the tripod or cameraman, while all videos provided by external contributors (14 videos) were recorded with their phone located on the ground, from different positions on the court. Thus, a total of 28 videos have been uploaded to the Roboflow, where each video was cut with some frequency which depended on the video duration. This frequency was usually adjusted so that the number of frames extracted from the video was in the range from 50 to 400. The total number of extracted frames reached 4,900. Considering the fact that the extracted frames represented basketball scenes from different angles of the court, from different heights of the installed camera, with different degrees of dynamism and different environments (indoors and outdoors), it is expected that the final dataset will be diverse, comprehensive and suitable for building a highly accurate and efficient YOLO model capable of detecting desired objects in different environments. All detailed information about the video recordings that make up the final dataset is presented in the table 1. The values of "Video Num" column are clickable for YouTube Shorts videos, that lead to source videos.

4.1.2 Data labelling

All extracted frames were manually labelled with Roboflow labelling tool, drawing bounding box annotations for the objects on the frame of 3 classes: person, ball and basket. Thus, the total number of annotations for all frames reached 16,103 annotations.

4.1.3 Data splitting

For each video, after it has been cut into the frame and all the objects of classes "person", "ball", "basket" in the frame have been labelled, these frames were shuffled and split into 3 subsets: training set (70%), validation set (20%) and testing set (10%). This way of splitting the dataset was proposed by Roboflow by default. The training set is the largest subset and is crucial for the initial learning

Video Num	Video Category	Type of Action	Camera Location	Frames Extracted	Ball Color	Environment
1	YouTube Shorts	Dribbling, Layups	Height	205	Orange	Indoors
2	YouTube Shorts	Dunks	Height	66	Orange	Indoors
3	YouTube Shorts	Layups	Height	249	Orange	Indoors
4	YouTube Shorts	Layups	Height	111	Orange	Indoors
5	YouTube Shorts	Dribbling, Throws	Height	110	Orange	Indoors
6	YouTube Shorts	Throws, Layups	Height	298	White red blue	Outdoors
7	YouTube Shorts	Layups	Height	73	Black	Outdoors
8	YouTube Shorts	Dribbling, Throws	Height	171	Orange	Outdoors, Indoors
9	YouTube Shorts	Throws	Height	185	Orange	Outdoors, Indoors
10	YouTube Shorts	Throws	Height	129	Orange	Outdoors
11	YouTube Shorts	Dribbling, Layups	Height	83	Orange	Outdoors
12	YouTube Shorts	Dribbling, Throws, Dunks	Height	204	Orange, Red blue	Outdoors
13	YouTube Shorts	Dribbling, Throws, Dunks	Height	377	Orange	Outdoors
14	YouTube Shorts	Dribbling, Throws	Height	106	Orange	Indoors
15	External Contributor	Dribbling, Throws	Ground	154	Orange	Indoors
16	External Contributor	Throws	Ground	68	Orange	Indoors
17	External Contributor	Dribbling, Throws, Layups	Ground	123	Orange	Indoors
18	External Contributor	Throws	Ground	124	Orange	Indoors
19	External Contributor	Dribbling, Throws, Layups	Ground	232	Orange	Indoors
20	External Contributor	Dribbling, Throws, Layups	Ground	168	Orange	Indoors
21	External Contributor	Dribbling, Throws, Layups	Ground	189	Orange	Indoors
22	External Contributor	Dribbling, Throws, Layups	Ground	239	Orange	Indoors
23	External Contributor	Dribbling, Throws, Layups	Ground	244	Orange	Indoors
24	External Contributor	Dribbling, Throws, Layups	Ground	245	Orange	Indoors
25	External Contributor	Dribbling, Throws, Layups	Ground	236	Orange	Indoors
26	External Contributor	Dribbling, Throws, Layups	Ground	195	Orange	Indoors
27	External Contributor	Throws	Ground	157	Orange	Indoors
28	External Contributor	Throws	Ground	159	Orange	Indoors
Total	-	-	-	4900	-	-

Table 1: Dataset Overview of Basketball Video Sources

process of the model. A larger training set will provide more examples and diverse basketball game scenarios , which will help the YOLO model to learn better and, subsequently, have better detection performance. The validation set is a separate section of our dataset that we will use during training to get a sense of how well our model is doing on images that are not being used in training. By using 20% of the data, there is a substantial amount of information to validate the effectiveness of the model under different conditions and environments. The test set is crucial for assessing the final YOLO model's performance, and it was decided to leave 10% of the data for it. It acts as new, unseen data to simulate how the model will perform in real-world conditions.

The chart 3 represents a stacked bar chart that visualizes the distribution of annotated objects by class (ball, basket, person) across different splits of a dataset. Each bar represents a class of objects, and the segments within each bar indicate the number of instances of that class in each dataset split. The total height of each bar represents the total number of instances of that class across all dataset splits. As you can see from the graph, the 'person' class has the highest number of total instances (6193),

Figure 3: Counts of Instances by Class and Subset

the 'basket' class has the second-highest count (5530), while the 'ball' class has the fewest number of instances (4380) across all the images. Such a picture of the distribution of the total number of instances can be observed, since often a lot of people appear in video fragments taken from YouTube Shorts, at the same time the number of balls or basketball hoops rarely exceeds 2. Confirmation of this can be clearly seen in graph 4, describing in more detail how many objects of each class are present in the images. Approximately the same large number of instances per each class reduces the risk that the model will be biased, meaning it will not show less accurate detection in favor of a certain object class.

4.1.4 Data Preprocessing and Augmentation

Image preprocessing is the steps taken to format images before they are used by model training and inference. It can include image resizing, rotation, color corrections, etc. In our case we use Auto-Orient and resize transformation, stretching all images to 1280x1280 pixels. Auto-Orient ensures that all images are aligned in the same orientation, which is crucial for the YOLO model to learn correctly. Stretching the frames will provide the correct model training, since YOLO model architecture requires the input images to be the same size, while the resolution of collected frames may vary. Moreover, in the next stage we will train our YOLO model on the same image resolution of 1280x1280, since such high image resolution may benefit the resulting model's detection capabilities of such small objects on the frame as a ball, and a basket.

Image augmentation involves modifying images to generate varied versions of the same content,

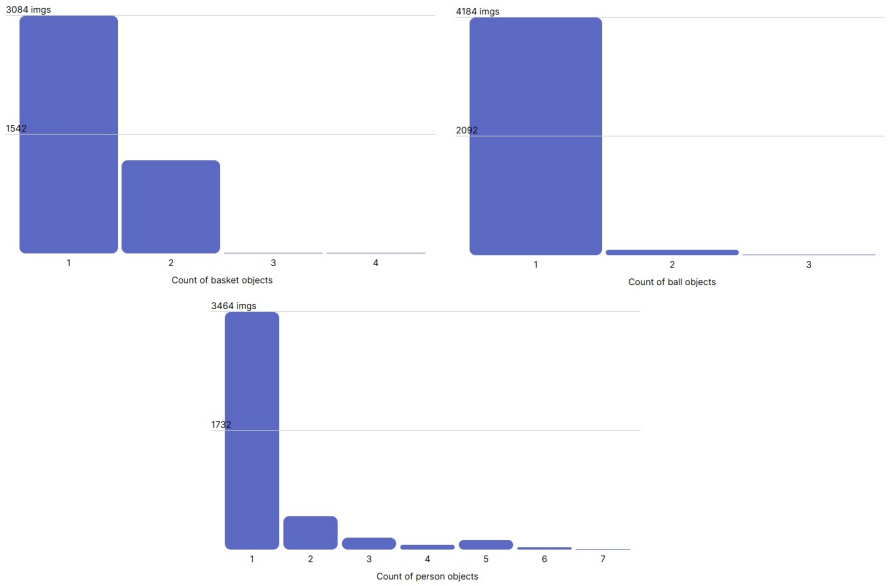


Figure 4: Counts of Objects by class in all the frames

thereby providing the model with a broader range of training examples. For instance, by randomly changing an image's rotation, brightness, or scale, the model is trained to recognize the subject of the image under diverse conditions. While image preprocessing steps are applied to training and test sets, image augmentation is only applied to the training data. Thus, I applied the horizontal flip manipulation to help the model be insensitive to subject orientation. At the same time, it increased the training set size by 1.75 times, from 3431 to 6017 images, which is good for training more accurate model.

After applying all mentioned image manipulation techniques to the dataset, I prepared the final version of the dataset with the distribution of the number of images across subsets as follows:

- **Training set (80%)**: 6017 images
- **Validation set (13%)**: 981 images
- **Testing set (7%)**: 488 images

4.2 Model Training and Evaluation

4.2.1 Model Training

This section will describe the process of training and evaluating the YOLO model based on the dataset we have prepared.

Since I develop a web application that will eventually be deployed on a cloud server, and all computing processes will also be performed on a cloud server using cloud computational resources like GPU, it was decided to use the latest released, highly efficient, yet fast and small in number of parameters YOLOv8l (large version), as a pre-trained model, that will be used as the starting point for training. The YOLOv8 models, provided by Ultralytics, were trained from scratch on famous COCO dataset, containing 200,000 labeled images with over 1.5 million object instances across 80 categories. Among all the classes, there are classes "Person" and "Ball", which intersects with the classes in our custom YOLO model, that will certainly benefit the resulting performance of our model and reduce training time requiring less computational resources. Moreover, training deep learning models from scratch generally requires large amounts of diverse data to perform well without overfitting, and since our dataset is small relative to the COCO dataset, it was decided to use the training approach using a pre-trained model.

In the training of the YOLO model, two critical settings were employed to optimize performance. The model was trained for 100 epochs, with each epoch representing a complete pass through the entire

training set, allowing the model to iteratively learn and adjust from the data over a total of 100 cycles. Additionally, the image size was set to 1280 pixels for both height and width during training. This uniform image size is crucial for maintaining consistency throughout the training process. Particularly, the choice of 1280 pixels is strategic for enhancing the model's capability to detect smaller objects, such as the "ball" in our dataset, which is relatively small compared to "person" class. This larger resolution helps in capturing finer details, crucial for accurate detection of small objects.

As for the computational resources, I utilized the powerful A100-SXM4-40GB GPU with 40514 MiB of memory, a resource provided by Google Colab Pro. This high-performance computing environment significantly enhanced and speed up the training process. During the entire training process, the GPU memory remained stable around 40GB, reflecting efficient utilization without overload.

Throughout the training, each of 100 epochs consistently evaluated the model performance on validation set (981 images and 3219 instances). The graph 5 illustrates the training and validation performance metrics of our custom YOLOv8l model over 100 epochs. Across all the 100 epochs, key

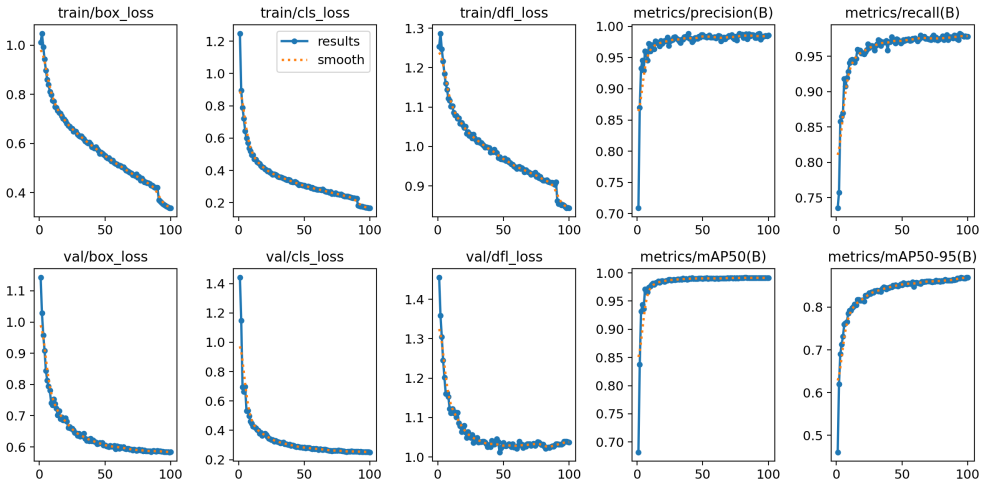


Figure 5: Training results

performance metrics, such as box loss, class loss, and distribution focal loss, all showed a downward trend, indicating enhanced accuracy in predicting both the locations and classifications of objects within images. The precision and recall graphs remain remarkably stable and high across the training epochs. High precision indicates that the model makes accurate predictions with a low rate of false positives, while the high recall indicates that the model successfully identifies a large proportion of relevant instances. The mean Average Precision (mAP) metrics, both mAP50 and the more stringent mAP50-95, demonstrate increasing accuracy, highlighting the model's growing ability to detect objects accurately across various Intersection over Union (IoU) thresholds. Overall, all mentioned evaluation metrics indicated a successful training progression, with the model becoming increasingly precise and reliable in its object detection capabilities. The entire training process took 5.902 hours of continuous training.

4.2.2 Model Evaluation

After completing the training, I evaluated the effectiveness of object detection by our model on a test dataset (488 images, 1612 instances) that the model had never seen before.

The graph 6 represents confusion matrices showing the results of evaluating a YOLO model's performance on a testing set, presented in both absolute counts and normalized forms, for object classifications into categories such as 'ball', 'basket', 'person', and 'background'. In the absolute counts matrix (left matrix), the model demonstrates high accuracy with predominant correct classifications for 'ball' (419 correct), 'basket' (546 correct), and 'person' (628 correct), with minimal misclassifications between these categories but some instances misclassified as 'background'. The normalized matrix further emphasizes this precision, highlighting nearly perfect identification of 'person' (1.00), excellent accuracy for 'basket' (0.99), and 'ball' (0.97). However, it also reveals minor issues with false positives,

where some objects are mistakenly identified as 'background'. This might be due to the fact that the dataset did not include background images, which do not contain objects and which are commonly used to minimize false positives. Overall, based on confusion matrices, the model exhibits strong object classification performance across classes "person", "basket" and "ball" with minimal error.

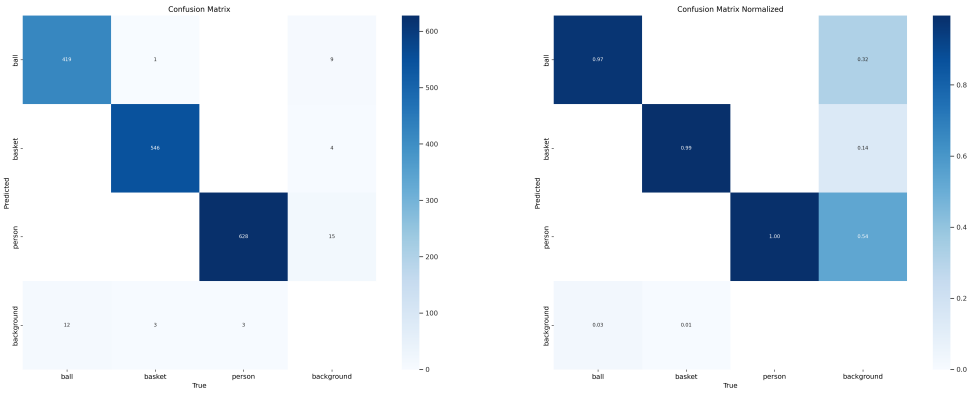


Figure 6: Object Detection Performance: Confusion Matrices Overview

The graphs in 7 represent various performance metrics in the form of curves for our YOLOv8l model evaluated on a testing set, specifically illustrating F1 score (a harmonic mean of precision and recall), precision, and recall against confidence levels for detecting objects categorized as 'ball', 'basket', and 'person'. The F1-Confidence curve indicates that the model achieves near-perfect F1 scores (0.99) at a confidence threshold of about 0.6, reflecting a strong balance between precision and recall. The Precision-Confidence curve shows high precision even from low confidence levels, suggesting minimal false positives, while the Recall-Confidence curve maintains high recall over almost the entire range of confidence level which indicates that the model consistently identifies most of the relevant objects across all categories with a high degree of certainty, regardless of the confidence setting, until it becomes excessively stringent (after 0.9 and up to 1). The Precision-Recall curve demonstrates that the model retains high precision and recall across the spectrum for all categories, underscoring its robust detection capabilities.

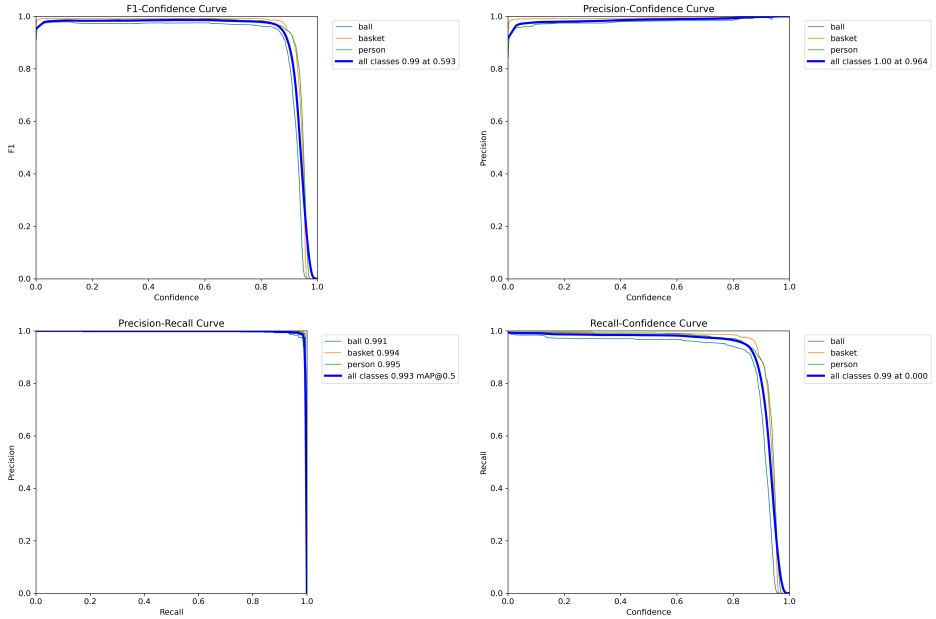


Figure 7: Object detection performance curves for YOLO model: F1, Precision, Recall

The table 2 displays the numerical results from evaluating our custom YOLOv8 model on a testing

set, providing metrics for overall performance as well as performance by individual classes: ball, basket, and person. Precision scores for the bounding boxes are consistently high, almost reaching 1.0, indicating that the model precisely identifies the bounding boxes containing the targeted objects. Recall metrics are similarly high, showcasing the model's ability to capture nearly all relevant objects. The mean Average Precision (mAP) at an IoU threshold of 0.50 is nearly perfect for all classes, indicating excellent model accuracy at this threshold. When the criteria are tightened across IoU thresholds from 0.50 to 0.95, the mAP scores are slightly lower but remain highly robust, confirming the model's reliable detection capabilities even under more stringent conditions.

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
all	488	1612	0.989	0.982	0.993	0.878
ball	488	431	0.983	0.968	0.991	0.807
basket	488	550	0.994	0.991	0.994	0.934
person	488	631	0.989	0.989	0.995	0.893

Table 2: Performance metrics of YOLO model on testing dataset across different classes

4.2.3 Model Inference

Picture 8 demonstrates objects detection capabilities on completely unseen data. As can be observed, in this case it perfectly detects actual objects on the provided frames: person, basket and ball.

Figure 8: Model inference on unseen data

4.3 Throws and goals detection algorithm

4.3.1 Building rectangular zones

In general terms, the logic of the algorithm for detecting throws and successful shots is based on obtaining and managing the coordinates of bounding boxes of the player, the ball, and the basket from each frame, as the output of our custom YOLOv8 model, and detecting the "ball" object in 3 certain rectangular zones built next to the basket. Before explaining the logic of the algorithm in detail, it is worth explaining what these zones are and how they are constructed based on the position of the basketball hoop in the frame.

For building these zones, I used PolygonZone from "supervision" Python library, a class for defining a polygon-shaped zone within a frame for detecting objects. Each of these 3 rectangular zones performs its own function and is constructed as follows:

- For successful shot detection:

  1. Zone above the basket (zone_above)
     - Zone width: the width of the "basket" bounding box increased on 10% (5% from both left and right sides).
     - Zone height: 60% of the height of the "basket" bounding box.
     - Zone location: higher on 10% of the "basket" bounding box height from the upper bound of "basket" bounding box.
  2. Zone below the basket (zone_below)
     - Zone width: the width of the "basket" bounding box decreased on 95% (47.5% from both left and right sides).
     - Zone height: 80% of the height of the "basket" bounding box.
     - Zone location: lower on 70% of the "basket" bounding box height from the lower bound of "basket" bounding box.

- For throw detection:

  1. More space coverage zone (zone_general)
     - Zone width: the width of the zone_above increased on 100% (50% from both left and right sides).
     - Zone height: 260% of the height of the zone_above.
     - Zone location: lower on 10% of the height of zone_above from the lower bound of zone_above.

It is worth saying that this way of constructing zones was derived experimentally, by processing a large number of videos and subsequent editing the way of constructing zones by minimizing the amount of false throws and goals detections. Figure 9 showcases how these zones look like at the frames with all other detected objects presented. As can be seen, 3 zones are built for each basket, regardless of the number of baskets in the frame.

Figure 9: Examples of building zone_above, zone_below, zone_general

4.3.2 The logic of the algorithm

The algorithm leverages the YOLO model to analyze each frame from the provided video for detection of such objects as a "person", "basket" and "ball", and then analyze the interaction of these objects and predefined rectangular zones located next to the basketball hoop.

The schema 10 represents how each frame is being processed by our algorithm. Initially, on the

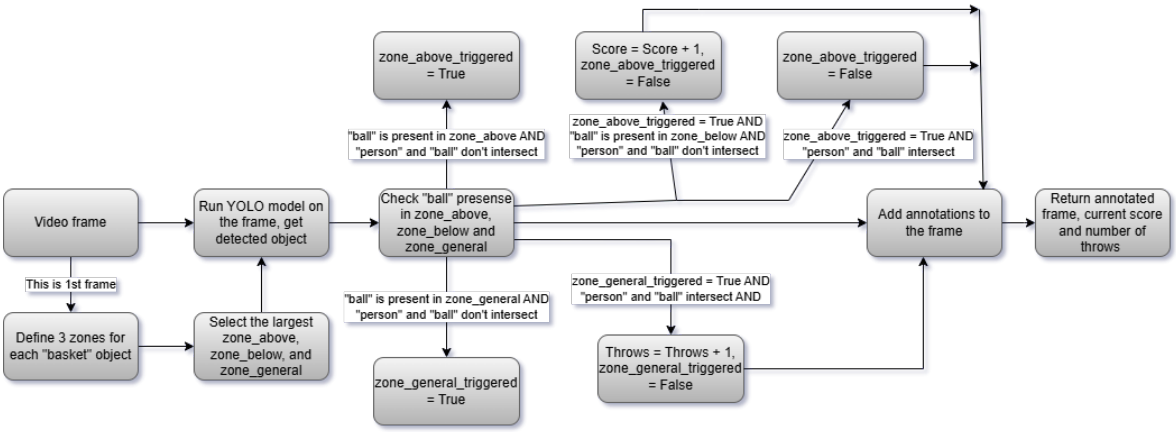


Figure 10: Pipeline diagram of how every frame is being processed by the algorithm

first frame with detected "basket" object present, the function sets up all 3 zones for each "basket" object, and then for each type of zone (zone_above, zone_below, zone_general) it selects the one with the largest area. Obviously, we start with the values of the number of shots and goals equal to 0. For every frame, the algorithm runs YOLO model and collects detected objects. Then, we check if the "ball" object intersect with any predefined zones without intersecting with the person. I use special variables "zone_above_triggered" and "zone_general_triggered" to truck whether zone_above and zone_general were triggered by the "ball". As soon as the ball hits these zones, these variables become true and will keep these values for all other frames until we set them as false.

Further, speaking about the logic of a scoring detection, if the variable "zone_above_triggered" is true, we are waiting for the moment when the ball appears in the zone_below. If this happens before the moment when the bounding boxes of the person and the ball intersect, we add the "score", meaning the number of successful shots, by one. Otherwise, if the outcome of intersection between a person and a ball occurs earlier, we do not change value of "score". As a result of both of these outcomes, we change the value of the variable "zone_above_triggered" to false, signifying the end of a circle of the throw.

The logic of a throw detection is similar; if the variable "zone_general_triggered" is true, we are waiting for the moment when bounding boxes of the person and the ball intersect. As soon as this happens, we add the value for throws by one, indicating the end of the circle for this throw. And this outcome is unambiguous, since a person needs to pick up the ball in order to make the next throw, then their bounding boxes will intersect.

After all the frames from the video have been processed, and the totals for the number of shots and goals were obtained, we can easily calculate the number of misses:

$$\text{misses} = \text{throws} - \text{score}$$

4.3.3 Video Requirements

Based on the capabilities of the YOLO trained model and the logic of the algorithm, to obtain the most accurate throws and goals detection results, it is anticipated that the provided video will match following requirements:

1. **Each shot must end with a person touching the ball on camera, so this is considered as the end of the throw**

The algorithm relies on detecting the position of the ball throughout the throw, and having the person touch the ball at the end of each shot provides a clear endpoint for the algorithm to determine the end of a throw. Without this, the algorithm may struggle to distinguish between different throws, especially if multiple consecutive shots occur. This step helps ensure that each throw is correctly counted and prevents confusion between throws in the detection process.

2. **During the throw, a person should not stand in front of the camera and obscure the visibility of the basketball hoop.**

   The YOLO model and the throw and score detection logic are heavily dependent on the clear visibility of the basketball hoop to accurately classify successful shots. If a person obstructs the view of the hoop, it can lead to errors in identifying whether the ball passed through the hoop, leading to false positives or missed shots. By ensuring that no one stands in front of the hoop, the algorithm can function properly, with unobstructed access to the critical visual cues it requires.

3. **If there are several basketball hoops in the video, shots and goals will be counted in the largest hoop in the frame.**

   The algorithm is designed to focus on one basketball hoop in the frame, and selecting the largest hoop helps to avoid confusion when there are multiple hoops in the video. The largest hoop is typically the closest to the camera, and by setting this as the default, the algorithm can consistently track the correct target. If multiple hoops were allowed without this constraint, the algorithm might struggle to distinguish between them, leading to incorrect goal classifications.

4. **There must be a net on the basketball hoops throughout the video.**

   This requirement is essential because my custom YOLO model used in the algorithm was trained on instances of basketball hoops that all had nets. The net is part of the "basket" object class in the training dataset, and the model relies on the presence of the net as a key feature for detecting the hoop. If the net is absent, the model may fail to detect the basketball hoop entirely, leading to significant errors in detecting throws and classifying successful shots. Therefore, the net is a critical element for proper goal detection.

5. **There must be only 1 ball throughout the video.**

   The algorithm is designed to track a single basketball in each video. Suppose an athlete is throwing a ball with one hand, while holding another ball in the second hand. Since the "person" and the "ball" bounding boxes intersect the entire time, as soon as thrown ball reaches the zone_general (highest rectangular zone in figure 9), the algorithm will start to increase the predicted number of throws for every frame where the bounding box of the thrown ball intersects the zone_general. This will lead to an almost instant increase in the counted number of throws by dozens and hundreds. This is just one of the possible scenarios, where the algorithm throws detection performance can dramatically fall. Therefore, to avoid it, we have to add this requirement of a single ball presence during the entire video.

6. **The color of the ball must be orange or at least close to it.**

   The YOLO model was trained on basketballs that are primarily orange, which is the standard color for basketballs. Deviations from this color could lead to poor detection performance, as the model may struggle to recognize non-standard ball colors. Ensuring the ball is orange or similar in hue helps the model maintain high detection accuracy across different lighting conditions and environments.

7. **The environment in the video can be both indoors and outdoors.**

   The YOLO model was trained on approximately the same number of images, both on the indoor and outdoor environment. This broadens the range of potential applications, making the system suitable for various basketball training and game scenarios.

8. **The camera must be stationary throughout the entire video recording.**

   As can be seen in the algorithm video processing pipeline in Figure 10, on the first frame with detected "basket" object present, the function sets up all 3 zones for this "basket" object, and the location of these zones in the frame remains stationary throughout the video, regardless of

subsequent “basket” detections. Therefore, if the camera moves, the position of the basketball hoop in the frame also moves, while the positions of the zones remain the same. This will lead to incorrect throws and score detection results, because in this case zones are not associated with the ”basket” instance anymore. Thus, a stationary camera ensures that the positions of the zones and the basketball hoop will be stationary throughout the video, avoiding such errors.

9. **The camera can be positioned on the ground or mounted at some height using a tripod.**

   The YOLO model was trained on approximately the same number of images, recorded with phone located on the ground and mounted on some height with the tripod. Therefore, the algorithm with core of YOLO model, is supposed to accurately handle any of these camera setups.

10. **The camera must be in positions on the court marked with red dots (Picture 11).**

    Placing the camera in predetermined positions ensures that the system has the best possible angle for detecting basketball throws and successful shots. These positions have been selected to provide clear visibility of the hoop and the ball’s trajectory, minimizing blind spots and maximizing detection accuracy.

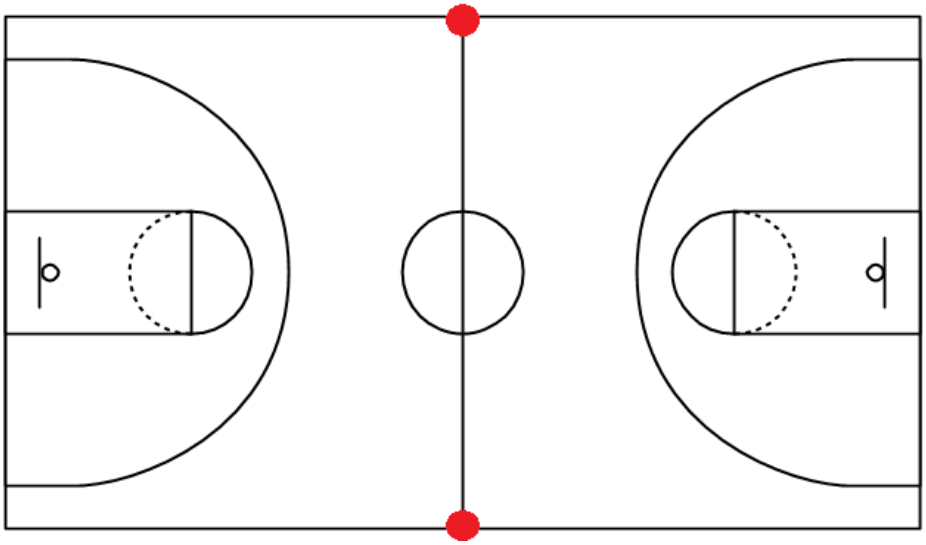


Figure 11: Required camera positions on the court marked with red dots

5 Results and Discussion

5.1 Testing Dataset

This section discusses one of the ways to objectively and accurately assess the effectiveness of the algorithm. I myself have recorded and prepared new 16 videos especially for algorithm evaluation, previously unseen by my YOLO model during its training and validation. These videos are 1-2 minute long, recorded both indoors and outdoors from both required camera heights (on the ground and with the tripod at a height of 1.5 meters) and required camera positions (red dots on figure 11) of an athlete throwing the ball into the basketball hoop from the free throw line and from three-point range, which the model has not seen before during its training and validation processes. These videos were recorded with iPhone 15 Pro with resolution 1080p (Full HD) and 60 fps. The recorded videos are shown in Table 3 with manually calculated shooting performance statistics for each video, that can be viewed by clicking on the video number and following the link.

Thus, total number of videos prepared to evaluate the accuracy of throws and goals detection algorithm is 16. As you can see from the table 3 each of these 1-2 minute videos includes 7-19 shots, depending on the video duration and the type of shot. The resulting dataset covers different court

Video Number	Court Location	Camera Height	Camera Position	Throw Type	Video Duration	Number Throws	Number Goals	Number Misses
1	Outdoors	Ground	Right Side From the Hoop	Free Throws	1:54	16	10	6
2	Outdoors	Ground	Right Side From the Hoop	3-Point Throws	1:39	12	4	8
3	Outdoors	Ground	Left Side From the Hoop	Free Throws	1:41	12	9	3
4	Outdoors	Ground	Left Side From the Hoop	3-Point Throws	1:43	13	1	12
5	Outdoors	Tripod	Right Side From the Hoop	Free Throws	1:30	13	8	5
6	Outdoors	Tripod	Right Side From the Hoop	3-Point Throws	2:02	11	4	7
7	Outdoors	Tripod	Left Side From the Hoop	Free Throws	1:48	16	6	11
8	Outdoors	Tripod	Left Side From the Hoop	3-Point Throws	1:22	8	2	6
9	Indoors	Ground	Right Side From the Hoop	Free Throws	1:56	15	7	8
10	Indoors	Ground	Right Side From the Hoop	3-Point Throws	1:19	10	5	5
11	Indoors	Ground	Left Side From the Hoop	Free Throws	1:58	14	4	10
12	Indoors	Ground	Left Side From the Hoop	3-Point Throws	1:00	7	1	6
13	Indoors	Tripod	Right Side From the Hoop	Free Throws	1:32	19	10	9
14	Indoors	Tripod	Right Side From the Hoop	3-Point Throws	1:43	13	5	8
15	Indoors	Tripod	Left Side From the Hoop	Free Throws	1:55	16	8	8
16	Indoors	Tripod	Left Side From the Hoop	3-Point Throws	1:00	7	2	5

Table 3: Videos prepared for evaluating the algorithm

locations, different camera positions on the court and its height, as well as different types of throws from near and far distances, which makes this dataset sufficiently representative for an objective evaluation of the algorithm.

5.2 Evaluation of Throws and Score Detection algorithm

5.2.1 Actual vs Algorithm-predicted results of shooting accuracy

After preparing a testing dataset, I have run my algorithm for each of collected videos, obtaining number of throws, successful shots and misses, as a result. The results are presented in the table 4, where the actual and algorithm-calculated results are explicitly compared. For clarity, the table has a "Difference" column, which shows the difference between the actual and predicted numbers of throws and goals. The "Video Number" column has clickable values which lead to links to videos processed by the algorithm. There you can clearly see how the shots and goals scored were counted, as well as what mistakes and accidental detections the algorithm made during video processing.

This table 4, which compares actual and algorithm-predicted shooting accuracy, must be considered in more detail for a more nuanced understanding of the model's performance. Intuitively, it may seem appropriate to focus on the total number of actual and predicted shots and goals, but sometimes this can hide important details of the algorithm's behavior. Specifically, we need to examine not only the overall totals but also how well the algorithm correctly identifies successful shots (goals) and misses.

For instance, there can be situations where the difference between actual and predicted throws or successful shots appears to be zero, suggesting that the model performed well. However, in reality, this might not be the case. Consider a scenario where there are 10 actual successful shots in a video: if 9 of them were correctly classified as goals, but 1 was misclassified as a miss, and the algorithm also falsely identified 1 miss as a goal, this could result in the same total number of predicted goals as actual goals. However, despite the total being correct, the model would have made two classification errors: one false negative (a goal classified as a miss) and one false positive (a miss classified as a goal). These misclassifications can lead to a misleading impression of accuracy if only the totals are considered.

Video Number	Court Location	Camera Height	Camera Position	Throws Type	Predicted		Actual		Difference	
					Throws	Score	Throws	Score	Throws	Score
1	Outdoors	Ground	Right Side From the Hoop	Free Throws	16	10	16	10	0	0
2	Outdoors	Ground	Right Side From the Hoop	3-Point Throws	12	6	12	4	0	2
3	Outdoors	Ground	Left Side From the Hoop	Free Throws	14	5	12	9	2	4
4	Outdoors	Ground	Left Side From the Hoop	3-Point Throws	16	1	13	1	3	0
5	Outdoors	Tripod	Right Side From the Hoop	Free Throws	15	4	13	8	2	4
6	Outdoors	Tripod	Right Side From the Hoop	3-Point Throws	11	3	11	4	0	1
7	Outdoors	Tripod	Left Side From the Hoop	Free Throws	16	4	16	6	0	2
8	Outdoors	Tripod	Left Side From the Hoop	3-Point Throws	8	3	8	2	0	1
9	Indoors	Ground	Right Side From the Hoop	Free Throws	16	5	15	7	1	2
10	Indoors	Ground	Right Side From the Hoop	3-Point Throws	10	6	10	5	0	1
11	Indoors	Ground	Left Side From the Hoop	Free Throws	14	5	14	4	0	1
12	Indoors	Ground	Left Side From the Hoop	3-Point Throws	7	2	7	1	0	1
13	Indoors	Tripod	Right Side From the Hoop	Free Throws	19	11	19	10	0	1
14	Indoors	Tripod	Right Side From the Hoop	3-Point Throws	13	6	13	5	0	1
15	Indoors	Tripod	Left Side From the Hoop	Free Throws	16	9	16	8	0	1
16	Indoors	Tripod	Left Side From the Hoop	3-Point Throws	7	2	7	2	0	0

Table 4: Comparison of actual and algorithm-predicted results of shooting accuracy

Thus, to truly understand the algorithm's performance, it's essential to compare not just the total predicted values but also to consider how accurately individual throws were classified as either goals or misses. This deeper analysis will reveal how well the model distinguishes between successful and unsuccessful throws, and whether there are any patterns of misclassification that need improvement. Therefore, the next section will consider specifically the correctly identified throws.

5.2.2 Detailed overview of throws identification results

The table 5 provides a detailed analysis of the algorithm's performance in identifying basketball throws, focusing on two key metrics: Identification Accuracy and Identification Precision. Identification Accuracy reflects the proportion of correctly identified throws relative to the total number of actual throws, and was calculated, using the formula:

$$\text{Identification Accuracy} = \frac{\text{Identified Correctly}}{\text{Actual}}$$

while Identification Precision measures how many of the predicted throws were truly correct, and was calculated with formula:

$$\text{Identification Precision} = \frac{\text{Identified Correctly}}{\text{Identified Correctly} + \text{Identified Incorrectly}}$$

Together, these metrics offer a comprehensive understanding of the algorithm's effectiveness across 16 videos.

The algorithm demonstrates 100% identification accuracy across all videos, meaning that it correctly identified all actual throws in each case. This indicates that the algorithm is highly effective at detecting real throws, with no missed throws in any of the videos. Achieving perfect accuracy in all instances shows that the model is capable of capturing every true throw in the dataset, which is an essential measure of its detection capabilities.

However, the identification precision provides a more nuanced view of the algorithm's performance. While precision is also 100% in most videos, indicating no false positives, there are a few videos where precision drops. For instance, in Video 3, precision falls to 85.71%, and in Videos 4 and 5, precision

Video_num	Throws					
	Actual	Predicted	Identified Correctly	Identified Incorrectly	Identification Accuracy	Identification Precision
1	16	16	16	0	1.0000	1.0000
2	12	12	12	0	1.0000	1.0000
3	12	14	12	2	1.0000	0.8571
4	13	16	13	3	1.0000	0.8125
5	13	15	13	2	1.0000	0.8667
6	11	11	11	0	1.0000	1.0000
7	16	16	16	0	1.0000	1.0000
8	8	8	8	0	1.0000	1.0000
9	15	16	15	1	1.0000	0.9375
10	10	10	10	0	1.0000	1.0000
11	14	14	14	0	1.0000	1.0000
12	7	7	7	0	1.0000	1.0000
13	19	19	19	0	1.0000	1.0000
14	13	13	13	0	1.0000	1.0000
15	16	16	16	0	1.0000	1.0000
16	7	7	7	0	1.0000	1.0000
Total	202	210	202	8	1.0000	0.9619

Table 5: Detailed overview of throws identification results

decreases to 81.25% and 86.67%, respectively. In these cases, the algorithm overpredicted, identifying some false throws that did not actually occur. This shows that while the algorithm captures all true throws, it sometimes struggles with false positives, reducing precision in those particular instances.

Basically, the observed pattern occurs as a result of outdoor area being more overloaded with objects that can look like the ball from a certain angle (e.g. YOLO model detects trash can as a ball in the video [5]) or other uncontrolled moving objects that can be caught on camera, which also affect the results of the algorithm (e.g. passing cyclists or randomly walking people in the videos [2, 5, 7]).

The biggest influence on such a noticeable decrease in the precision of identifying throws in outdoors videos was the fact that the cap of the athlete throwing the ball in the video was very similar in color and shape to the ball, especially when he was standing with his back to the camera. This fact greatly confused the YOLO model during its object detections. In the videos [3, 4, 7] you can clearly see how ball detection is constantly changing between the cap and the ball, which can lead to incorrect results. The fact is that during the training and validation processes of the YOLO model, it has never seen a basketball player in a cap or hat on the court standing with his back to the camera, which led to such results.

Overall, the algorithm achieves an impressive 96.19% precision across all videos, indicating that only 8 out of 210 predicted throws were false positives. This high precision, combined with the perfect accuracy, demonstrates the algorithm's strong performance in identifying true throws while making only occasional misclassifications.

In conclusion of throws identification analysis, the algorithm is highly reliable in detecting basketball throws, achieving perfect accuracy in all cases. Although it occasionally overpredicts and produces false positives in a few videos, the overall precision remains very high at 96.19%, reflecting the model's overall effectiveness.

The next section will be specifically devoted to evaluating the accuracy of the classification of successful shots.

5.2.3 Detailed overview of score detection results

The table 6 presents the algorithm's performance in classifying successful shots, focusing on the "Score Classification Accuracy" column, which values were obtained with formula:

$$\text{Score Classification Accuracy} = \frac{\text{Classified Correctly as Success (TP)}}{\text{Actual}}$$

This metric reflects how well the algorithm correctly identifies successful throws (true positives) and avoids false classifications of unsuccessful throws as successful (false negatives). In many videos,

Video Number	Successful throws				
	Actual	Predicted	Classified Correctly as Success (TP)	Classified Incorrectly as Unsuccess (FN)	Score Classification Accuracy
1	10	10	10	0	1.0000
2	4	6	4	0	1.0000
3	9	5	4	5	0.4444
4	1	1	0	1	0.0000
5	8	4	4	4	0.5000
6	4	3	2	2	0.5000
7	6	4	4	2	0.6667
8	2	3	2	0	1.0000
9	7	5	5	2	0.7143
10	5	6	5	0	1.0000
11	4	5	4	0	1.0000
12	1	2	1	0	1.0000
13	10	11	10	0	1.0000
14	5	6	5	0	1.0000
15	8	9	8	0	1.0000
16	2	2	2	0	1.0000
Total	86	82	70	16	0.8016

Table 6: Detailed overview of score detection results

the algorithm performs exceptionally well, achieving 100% score classification accuracy in 10 out of 16 videos. This demonstrates that, in typical scenarios, the model is highly effective at distinguishing between successful and unsuccessful throws.

However, there are several instances where the algorithm's accuracy drops significantly. For example, in Videos 3-7 and 9, the algorithm struggles to maintain high accuracy. In Video 3, only 4 out of 9 successful throws were correctly classified, and 5 were incorrectly identified as successful, resulting in a classification accuracy of 44.44%. Similarly, in Videos 5 and 6, the accuracy drops to 50%, with half of the successful throws misclassified. Again, these lower accuracies of score classification are mostly seen in videos recorded on an outdoor court, that can be related to the outdoor setting, which contains more objects that, from certain angles, resemble the ball, like a cap on an athlete's head or a round trash can in the background.

A particularly notable case is Video 4, where the algorithm failed to correctly classify a single successful shot made in the video, leading to a 0% score classification accuracy.

Overall, the algorithm achieved a total score classification accuracy of 80.16%, correctly identifying 70 out of 86 successful throws across all videos. While the overall performance is solid, the variability in accuracy across different videos suggests that the algorithm is not consistently effective in all conditions. Improving its ability to handle more complex or ambiguous situations, as seen in lower-performing videos, could significantly enhance the algorithm's robustness and accuracy in detecting successful throws, which will be described in more detail in section "Possible improvements of the algorithm".

5.2.4 Detailed overview of misses detection results

The table 7 provides an assessment of the algorithm's performance in detecting unsuccessful throws (misses), focusing on the "Misses Classification Accuracy" column, calculated as:

$$\text{Misses Classification Accuracy} = \frac{\text{Classified Correctly as Unsuccess (TN)}}{\text{Actual}}$$

This metric measures how accurately the algorithm classifies missed shots as unsuccessful (true negatives) without incorrectly identifying them as successful (false positives). The results reveal that the algorithm performs well in many cases, achieving 100% accuracy in 8 out of the 16 videos, such as in Videos 1, 5, 7, and 16. In these cases, the algorithm correctly classified all missed shots, showing a high level of precision in identifying unsuccessful throws under certain conditions. The algorithm's performance slightly decreasing in a few videos, such as Videos 8 and 12, where the misses classification accuracy was 83.33%. In this videos, 1 missed shot was incorrectly classified as successful, indicating that the algorithm has some difficulty in distinguishing misses from successful throws in certain scenarios.

Video Number	Unsuccessful throws				
	Actual	Predicted	Classified Correctly as Unsuccess (TN)	Classified Incorrectly as Success (FP)	Misses Classification Accuracy
1	6	6	6	0	1.0000
2	8	6	6	2	0.7500
3	3	7	2	1	0.6667
4	12	12	11	1	0.9167
5	5	9	5	0	1.0000
6	7	8	6	1	0.8571
7	10	12	10	0	1.0000
8	6	5	5	1	0.8333
9	8	10	8	0	1.0000
10	5	4	4	1	0.8000
11	10	9	9	1	0.9000
12	6	5	5	1	0.8333
13	9	8	8	1	0.8889
14	8	7	7	1	0.8750
15	8	7	7	1	0.8750
16	5	5	5	0	1.0000
Total	116	120	104	12	0.8872

Table 7: Detailed overview of misses detection results

In several other videos, such as Video 2, 3, and 4, the algorithm performs even less accurately, although it still demonstrates relatively strong results. For example, in Video 2, the algorithm achieved a misses classification accuracy of 75%, misclassifying 2 out of 8 missed shots as successful. Similarly, Video 3 shows an accuracy of 66.67%, with 2 out of 6 missed throws misclassified. These cases suggest that while the algorithm is generally effective at detecting missed shots, it can sometimes struggle with certain conditions, leading to misclassification.

Overall, the algorithm's performance is strong, with an overall misses classification accuracy of 88.72% across all 16 videos. While it generally classifies missed throws accurately, there are occasional misclassifications that highlight areas where the algorithm can be refined, especially in outdoor videos, where the conditions are generally more challenging. Again, all possible ways of improvements of algorithm's capabilities in classifying successful and unsuccessful shots are presented in section "Possible improvements of the algorithm".

5.2.5 Successful and unsuccessful throws classification

The table 8 provides a detailed assessment of the algorithm's ability to classify both successful and unsuccessful basketball shots using four key metrics: accuracy, precision, recall, and F1 score. These metrics offer a comprehensive evaluation of how well the algorithm performs across 16 different videos. It is worth noting that for calculating these metrics we used columns "Classified Correctly as Success (TP)" and "Classified Incorrectly as Unsuccess (FN)" from the table 6 and columns "Classified Correctly as Unsuccess (TN)" and "Classified Incorrectly as Success (FP)" from the table 7, as well as following formulas:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F1} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

The accuracy indicates the overall proportion of correctly classified throws, while precision measures how many of the predicted successful throws were actually successful. Recall evaluates the algorithm's ability to detect all successful throws, and the F1 score combines precision and recall into a balanced measure of performance.

In several videos, such as Video 1 and Video 16, the algorithm achieves 100% accuracy, precision, recall, and F1 score, demonstrating flawless performance in distinguishing between successful

Successful and Unsuccessful Throws Classification				
Video Number	Accuracy	Recall	Precision	F1
1	1.0000	1.0000	1.0000	1.0000
2	0.8333	1.0000	0.6667	0.8000
3	0.5000	0.4444	0.8000	0.5714
4	0.8461	0.0000	0.0000	0.0000
5	0.6923	0.5000	1.0000	0.6667
6	0.7273	0.5000	0.6667	0.5714
7	0.8750	0.6667	1.0000	0.8000
8	0.8750	1.0000	0.6667	0.8000
9	0.8667	0.7143	1.0000	0.8333
10	0.9000	1.0000	0.8333	0.9091
11	0.9286	1.0000	0.8000	0.8889
12	0.8571	1.0000	0.5000	0.6667
13	0.9474	1.0000	0.9091	0.9524
14	0.9231	1.0000	0.8333	0.9091
15	0.9375	1.0000	0.8889	0.9412
16	1.0000	1.0000	1.0000	1.0000
Total	0.8614	0.8140	0.8537	0.8333

Table 8: Successful and unsuccessful throws classification

and unsuccessful throws. These results highlight the algorithm's effectiveness in scenarios where the differences between the two types of throws are clear and the classification task is straightforward, without any challenging circumstances.

However, while the algorithm generally performs well, with accuracy levels above 80% in most cases, there are some instances where its performance drops. For example, in Videos 2, 4, 7-15, the algorithm maintains a solid accuracy range between 83.33% and 94.74%, with high precision and recall. These results suggest that, in most cases, the algorithm can reliably identify both successful and unsuccessful shots, although it may occasionally misclassify a few throws.

Notably, the algorithm struggles in some videos, particularly in Videos 3 and 5, where the classification accuracy is significantly lower. In Video 3, accuracy falls to 50%, with low recall and an F1 score of 0.5714, indicating that the algorithm has difficulty accurately classifying throws in this video. Similarly, in Video 4, the precision drops to 0.0000, meaning the algorithm failed to classify any successful throws correctly, which severely impacts its overall performance. These cases highlight areas where the algorithm faces challenges, likely due to more complex video conditions, such as varying throw types or environmental factors. These are the very cases where inaccurate ball detection prevented the algorithm from working correctly to determine successful and unsuccessful throws. Since the whole logic of the algorithm is based on accurately determining the position of the ball in the frame and its passage through the zones above and below the basketball hoop, any misdetections of the ball could lead to inaccurate results of score and misses detection.

Looking at the overall performance in the last row, the algorithm achieves a total accuracy of 86.14%, indicating that it correctly classifies throws most of the time. The overall precision of 85.37% shows that the algorithm is fairly reliable when it predicts a throw as successful, while the recall of 81.40% suggests it is generally good at detecting successful throws. With an F1 score of 83.33%, the algorithm demonstrates strong, balanced performance, although there is room for improvement in consistently handling more difficult cases.

5.2.6 Effect of different conditions on the algorithm performance

This section analyzes how different video characteristics affect the algorithm's performance in identifying throws and classifying successful and unsuccessful shots, based on the tables 5 and 8.

– **Effect of Court Location (Indoors vs. Outdoors)**.

Starting with the impact of court location (indoors vs. outdoors), from Table 4, the identification accuracy for both indoor and outdoor environments remains at 100%. However, the throws

identification precision shows noticeable differences between indoor and outdoor videos. The outdoor precision calculated is 0.935, while the indoor precision is higher, standing at 0.990. This suggests that the algorithm performs more consistently and accurately in indoor environments when identifying throws, which is likely due to fewer distractions and more controlled lighting conditions indoors.

For successful and unsuccessful shot classification, as shown in Table 7, the average accuracy, precision, and recall were calculated for indoor and outdoor environments. The mean accuracy for outdoor shots is 0.795, while for indoor shots, it is 0.915. Similarly, the mean precision for outdoor shots is 0.725, whereas for indoor shots, it rises to 0.846. This indicates that the algorithm performs better in indoor environments, with fewer false positives and misclassifications of shots compared to outdoor courts. This could be attributed to outdoor conditions having more moving objects or elements like shadows, wind, or other external factors that affect the algorithm's ability to track and classify the ball's trajectory.

– **Effect of Camera Height (Ground vs. Tripod)**.

When assessing the effect of camera height on the algorithm's performance, we observe that both ground and tripod-mounted cameras show consistently high throw identification accuracy and precision, as seen in Table 4. Ground cameras produce near-perfect precision in most videos, though some slight deviations occur, such as in Video 3 (0.857) and Video 9 (0.9375), which may result from specific environmental factors. Tripod-mounted cameras exhibit similarly strong performance in identifying throws, with no significant variations in precision. However, when it comes to classifying successful and unsuccessful shots (Table 7), the stability provided by tripod-mounted cameras seems to offer a slight advantage. While classification accuracy in ground camera setups varies, tripod setups generally exhibit better accuracy, particularly for free throws, where accuracy often approaches perfection (e.g., 0.9375 in Video 15 and 0.9474 in Video 13). This suggests that tripod-mounted cameras might provide more stability, helping the algorithm to improve shot classification performance.

– **Effect of Camera Position (Right vs. Left Side from the hoop)**.

The effect of camera position, whether on the right or left side of the hoop, reveals notable differences in both throw identification and shot classification. From Table 4, cameras positioned on the right side generally achieve higher identification precision and accuracy, often maintaining a perfect precision of 1.00 across several videos. Some minor deviations occur, such as in Video 5 (0.8667 precision), but these are exceptions. In contrast, left-side camera positions show more variation in precision, especially in outdoor environments, as evidenced by Videos 3 and 4, where the precision drops to 0.8571 and 0.8125, respectively. This could indicate that the left-side perspective introduces more challenges for the algorithm, perhaps due to differing visual angles or environmental factors. In terms of shot classification, the right-side cameras also perform better, particularly in free throw classifications, which maintain high accuracy across most videos. Meanwhile, left-side positions show more inconsistency, such as in Video 3, where free throw classification accuracy falls to 0.5. This suggests that the algorithm benefits from the camera being on the right side of the hoop, likely because it provides a clearer and more informative visual perspective for shot analysis.

– **Effect of Throw Type (Free Throw vs. 3-Point Throw)**.

Regarding the effect of throw type, free throws are generally easier for the algorithm to identify and classify compared to 3-point throws. From Table 4, free throws consistently demonstrate perfect or near-perfect identification accuracy and precision, indicating that the algorithm is highly reliable in recognizing these types of shots. On the other hand, 3-point throws are more challenging to identify, especially in outdoor environments. Videos 3 and 4 show a decline in precision for 3-point throws, with values of 0.8571 and 0.8125, respectively. This suggests that the increased distance and movement associated with 3-point throws make them harder to detect. Similarly, classification accuracy for 3-point throws also tends to be lower than for free throws, as seen in Table 7. For example, Video 3 (outdoors, left side) exhibits an accuracy of only 0.5 for classifying 3-point throws, whereas free throws generally maintain much higher classification accuracies. In more controlled indoor environments, 3-point throw classification improves significantly, often reaching 0.9 or higher, indicating that external factors in outdoor environments,

such as lighting and background distractions, may impair the algorithm's performance with more complex shot types.

Final summary:

- **Camera Height:** Tripod cameras provide slightly better classification performance than ground-based cameras.
- **Camera Position:** Right side cameras produce more accurate results for both throw identification and classification.
- **Throw Type:** Free throws are easier for the algorithm to both identify and classify, while 3-point throws pose more of a challenge, particularly in outdoor settings.

5.2.7 Difference in detection between throw and score

Overall, the algorithm shows the result in detecting shots much better than in detecting score. This conclusion can be drawn from the comparison of throw identification accuracy in the table 5 and score classification accuracy in the table 6, as well as from the column "Difference" in the table 4.

This pattern can be observed due to the fact that the algorithm for score detection is more complicated than just throw detection, as in this case we make sure the ball goes through 2 zones (above and below the basketball hoop) to count this as a goal. Thus, in case of accidental and incorrect detection of the ball on the video, which happened quite often, especially on the outdoor court, the accuracy of the detection of the successful shot is most damaged.

Additionally, the throws detection method generally outperforms the goals detection method even in case of no false detections. The thing is that there are often cases when the ball crosses both zones (above and below the basket) not hitting the basketball hoop, but passing right in front of it. The examples of such situations can be seen in videos 13 (1:12), 14 (0:33), 15 (0:56), etc. The elimination of such errors requires further improvement of the algorithm, all possible ways of which will be discussed in the next section.

5.3 Possible improvements of the algorithm detections capabilities

There are several possible improvements that could enhance the algorithm's accuracy, particularly in detecting successful goals and reducing the larger prediction errors observed. First, refining the ball detection process is crucial. One of the primary reasons for errors in goal detection is accidental and incorrect identification of the ball, especially in outdoor environments where factors such as lighting and external objects interfere with the detection process. Therefor, expanding the training dataset would greatly contribute to this. By incorporating more variations of basketball plays—including different types of courts, player positions, shot types, and environmental factors—the algorithm would be able to generalize better to real-world scenarios, reducing detections errors and improving algorithm's accuracy.

Another possible way to improve the algorithm's accuracy is to add a second camera from a different angle. By using a combination of two cameras, each positioned at distinct viewpoints, the system could capture more comprehensive information about the ball's trajectory and eliminate blind spots. This would provide a fuller picture of the court and the shot, covering multiple angles that a single camera cannot capture. For instance, while one camera may struggle to accurately detect a ball crossing in front of the hoop without entering it, the second camera could provide a different perspective, making it easier to determine whether the ball has actually gone through the hoop. This dual-camera setup would reduce the chances of misidentifying false goals, as the algorithm would cross-check the ball's position from both angles before making a final decision on whether a shot was successful. The combined data from both cameras would result in more precise detection of goals and throws, as it mitigates the risk of occlusion or misdetection that often arises from a single-camera setup.

Incorporating a Support Vector Machine (SVM) algorithm could significantly enhance the accuracy of throws and goals detection by improving classification and decision-making processes. SVM is highly effective at handling complex scenarios and distinguishing between true and false detections, which is especially important for reducing false positives—such as identifying a ball that passes in front of the hoop but does not actually enter. By analyzing high-dimensional data like ball trajectory, speed,

and position, SVM can classify whether a detected shot is likely to result in a successful goal or not. Additionally, SVM could enhance trajectory prediction, learning from historical patterns to better identify valid goals. It could also be integrated into our existing detection pipeline as an extra layer of verification, working in tandem with YOLO, to further reduce errors and boost the robustness of the overall system. This combination would lead to more precise and reliable detection of both throws and goals.

The last method to improve the accuracy of goal detection is by adding an additional class to the YOLO model, such as "ball in the hoop," which could serve as an extra verification layer alongside the existing zones-crossing method. This new class would allow the algorithm to explicitly detect when the ball is inside the hoop, reducing false positives where the ball crosses in front of the hoop but does not actually enter. By training YOLO to recognize this specific event, the algorithm would have a more robust way of confirming successful goals, improving accuracy overall. However, implementing this method would require a large and diverse dataset of videos for training the YOLO model. Since a typical 2-minute video contains around 15 shots and approximately 7 of them are successful on average, the number of instances for the "ball in the hoop" class would be relatively low compared to other classes like "ball" or "person". To ensure the YOLO model can accurately detect this new class, a substantial dataset with hundreds of videos would be needed, capturing various angles, lighting conditions, and court setups. This would provide the model with enough examples to generalize well and reliably detect the "ball in the hoop" event, thereby reducing errors and enhancing the overall goal verification process.

By addressing these areas, the algorithm can be made more robust, accurate, and efficient in detecting throws, successful shots and misses, which would in turn reduce the prediction errors highlighted in tables 5, 6, 7 and 8.

5.4 Algorithm Evaluation Conclusion

In this evaluation, the performance of the algorithm for detecting basketball throws and classifying successful versus unsuccessful shots was analyzed in depth across several metrics, including throw identification accuracy, throw identification precision, score classification accuracy, and misses classification accuracy.

The algorithm demonstrated perfect identification accuracy across all videos, successfully detecting every true throw in all 16 video samples. The throw identification precision was also very high at 96.19%, with only 8 out of 210 throws incorrectly identified as false positives. This result indicates that the algorithm is highly effective at detecting basketball throws, although it occasionally over-predicts, especially in outdoor environments where there are more objects and distractions resembling a ball. Outdoor videos posed challenges due to environmental factors, such as moving objects or players wearing similar colors to the ball, which led to misclassifications. Despite this, the high accuracy and precision show that the algorithm reliably identifies real throws with minimal errors.

However, the algorithm's performance in detecting successful shots (goals) was more variable. The score classification accuracy averaged 80.16%, meaning the algorithm correctly classified 70 out of 86 actual successful throws. While the algorithm performed flawlessly in 10 out of the 16 videos, achieving 100% accuracy, it struggled in others, such as Videos 3, 4, and 5. Video 4, in particular, showed a 0% classification accuracy, failing to classify a single successful shot correctly. This discrepancy highlights that although the algorithm generally performs well, it can falter in complex or ambiguous scenarios, especially in outdoor environments where additional moving objects or environmental conditions may affect performance.

In detecting missed shots (unsuccessful throws), the algorithm performed well, achieving an overall misses classification accuracy of 88.72%. Similar to score detection, the algorithm reached 100% accuracy in 8 out of the 16 videos but showed lower accuracy in others. In Videos 2 and 3, the misses classification accuracy dropped to 75% and 66.67%, respectively, indicating some difficulty in distinguishing between successful and unsuccessful shots. This suggests that while the algorithm generally classifies misses correctly, certain dynamic outdoor environments pose occasional challenges that result in misclassifications.

Overall classification performance, considering both successful and unsuccessful throws, was evaluated using accuracy, precision, recall, and F1 score. The algorithm achieved an average accuracy of 86.14%, with a precision of 85.37%, a recall of 81.40%, and an F1 score of 83.33%. These metrics reflect a balanced performance in distinguishing between successful and unsuccessful throws. However,

there were videos where the algorithm struggled more, such as Videos 3 and 4, where the performance dropped significantly. For instance, Video 3 had a classification accuracy of only 50%, and in Video 4, precision dropped to 0% for successful throws. These inconsistencies highlight the need for further improvements, especially under complex or ambiguous conditions, such as occlusions, varying angles, or challenging lighting.

The algorithm's performance was also analyzed under different conditions, such as indoor vs. outdoor courts, camera height (ground vs. tripod), camera position (right vs. left), and throw type (free throws vs. 3-point throws). Indoor environments resulted in higher accuracy and precision for both throw and score detection compared to outdoor settings, likely due to fewer distractions and more controlled lighting. Camera height also influenced performance, with better accuracy when the camera was mounted on a tripod, as this provided a clearer view of the court compared to ground-level cameras. The camera's right-side position consistently outperformed the left side, possibly due to better visibility of the ball's trajectory and fewer obstructions. Additionally, free throws were slightly easier for the algorithm to classify accurately than 3-point throws, which are more challenging due to their longer and faster trajectories.

In conclusion, the algorithm shows strong performance in detecting basketball throws, with near-perfect identification accuracy and high precision in most cases. However, there is room for improvement in classifying successful and unsuccessful shots, particularly in outdoor environments or under challenging conditions. The overall classification performance is solid, but further improvements, such as refining ball detection, improving shot classification, and addressing false positives, will help the algorithm achieve more consistent and robust results. Additionally, enhancements like incorporating multiple camera angles or expanding the training dataset to include more diverse scenarios could significantly improve the algorithm's performance, especially in detecting goals and misses.

6 Web Application

The final phase of the project centered on the development and deployment of a cloud-based web application. This application provides a user-friendly interface where athletes can upload their training videos and receive the clear statistics of their shooting performance. Users benefit from interactive visualizations, such as line graphs showing changes in shooting accuracy over time and bar charts showing the distribution between successful and unsuccessful throws. The backend, that is built with the FastAPI Python web-framework, manages video processing and store personal and basketball statistics in a PostgreSQL database. SQLAlchemy facilitates interaction between Python and the database through an optional high-level Object-Relational Mapping (ORM) interface, allowing the use of Python classes instead of raw SQL. This architecture supports the intensive computational demands of video processing and ensures secure, efficient management of user data.

The application was successfully deployed on the remote server using Render service, that can be accessed through the link basketball-application. Since I used free subscription on Render service, a very small amount of resources are available to keep my application running. As a result, it is impossible to upload videos to the platform, since the available memory on the remote server is limited to several megabytes. However, you can still access the application, register, log in, and try to use its functionality. Also, you can watch the 10-minutes screencast, recorded by myself, where I demonstrate in detail all the functionality of the application by running it on my local computer.

6.1 Main Technologies

Following technologies have been used in creating this web application:

FastAPI: a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. In my application it has been used as the framework to develop the backend of my web application, handling HTTP requests and routing them to the appropriate functions. Also, it automatically generates the interactive API documentation of web application which can be accessed with "/docs".

FastAPI-Users: a user management system for FastAPI, which provides authentication and user management functionalities out-of-the-box. In my application this library is used to handle user registration, login, logout, and secure user authentication using JWT (JSON Web Tokens).

JWT (JSON Web Tokens): a compact, URL-safe means of representing claims to be transferred between two parties. JWTs are used in my web application for secure authentication, as a token that clients send to the server with each request after logging in. This token is verified by the server to ensure that the request is made by an authenticated user, enabling stateless authentication.

Uvicorn: an exceptionally rapid server implementation. Uvicorn is used to serve my FastAPI application, handling HTTP requests and providing high-performance capabilities, especially for asynchronous web applications.

Asyncio: asyncio is a Python library that allows you to write programs that perform multiple tasks at the same time without getting stuck waiting for one task to complete before starting another. This enables my application to be used by multiple users simultaneously.

SQLAlchemy: a SQL toolkit and Object-Relational Mapping (ORM) library for Python. SQLAlchemy has been used to interact with my PostgreSQL database in an efficient and Pythonic way, allowing to define and manage my database schema in Python classes without writing raw SQL queries.

Pydantic: a data validation and settings management library for Python that ensures that the data the application processes meets the expected formats and types. In my application, Pydantic is used to validate data that is sent to the server after users registration. Thus, if the user's email does not contain single sign while registration, the server issues an error message and does not register that user.

PostgreSQL: a powerful, open-source relational database management system. PostgreSQL is used as the primary database in my web application to store and manage data about users and videos they uploaded with detailed statistics about their basketball shooting accuracy, providing robust data storage and querying capabilities.

HTML: the standard markup language for documents designed to be displayed in a web browser. HTML is used in my application as the frame of the frontend allowing to structure the content and layout of my web pages, defining elements such as headers, paragraphs, forms, and buttons.

CSS: a stylesheet language used to describe the presentation of a document written in HTML. CSS is used in my web application to style and layout my web pages, allowing to create well-designed and user-friendly interface.

JavaScript: JavaScript is used in my web application to add interactivity to web pages such as adding specific logic for different buttons (e.g. buttons "next" and "back" in Developers web page allows to switch between pages of text) and handling user inputs on the client side (e.g. a check is enabled to ensure that the user can upload only an mp4 file on the platform)

6.2 Features and Buttons

6.2.1 Authentication

– Users can register on the web site using their username, email and password (Figure 12).

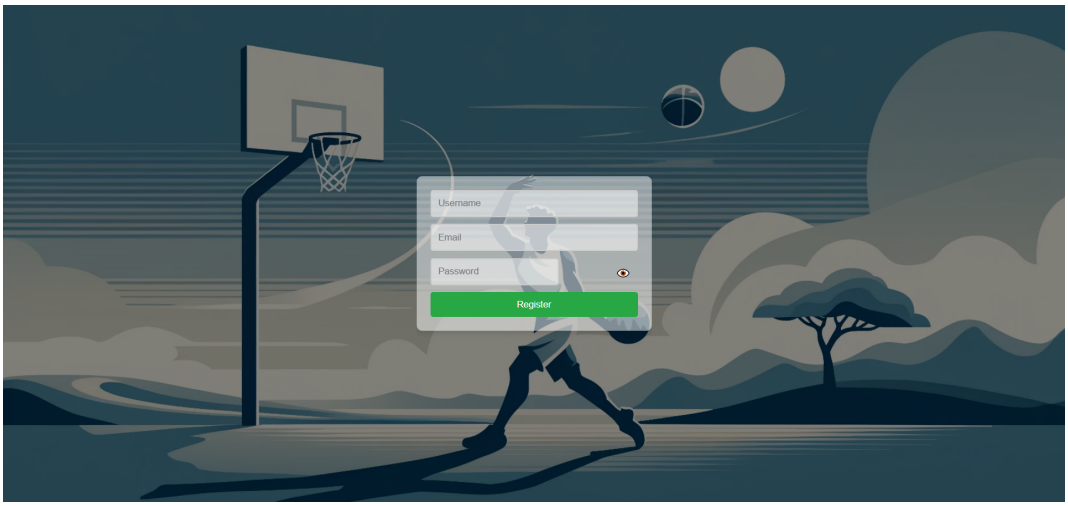


Figure 12: "Register" page

– Users can log in to their account on the website using email and password (Figure 13).

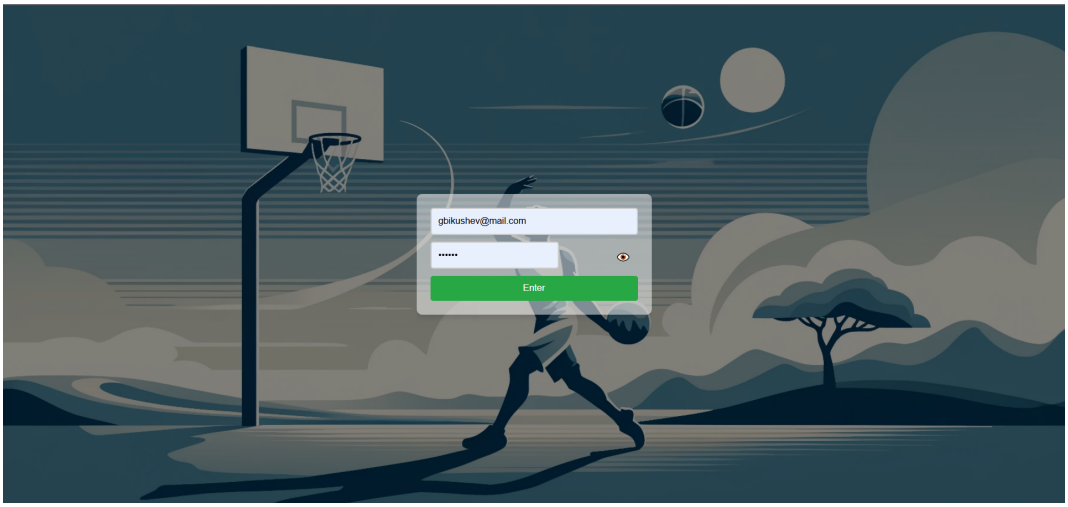


Figure 13: "Login" page

6.2.2 Header

There are 4 buttons on the left side of the header that help users to navigate through the web app and describe in detail about its capabilities:

– **Home Page**: takes you to the main page of the application where all features are presented (Figures 14 and 15).

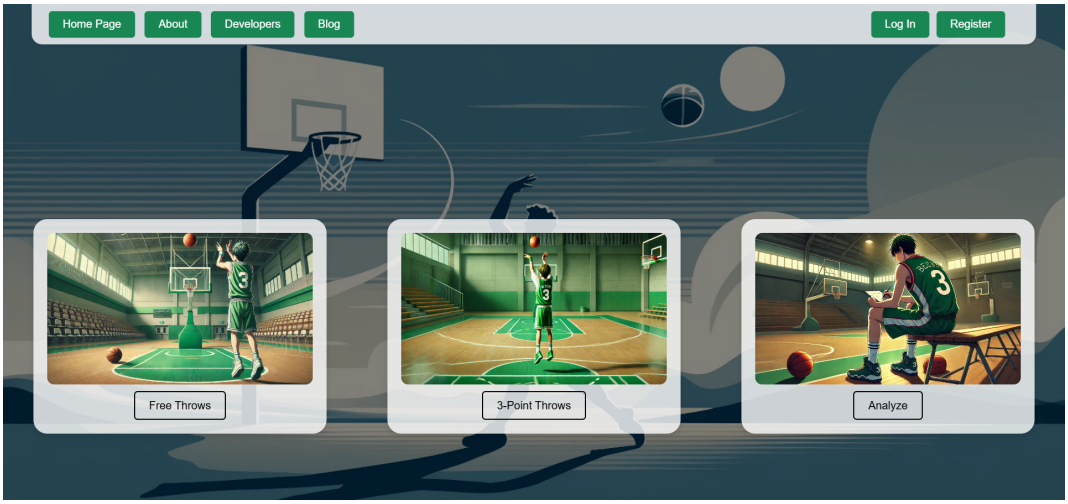


Figure 14: Homepage when user is not logged in

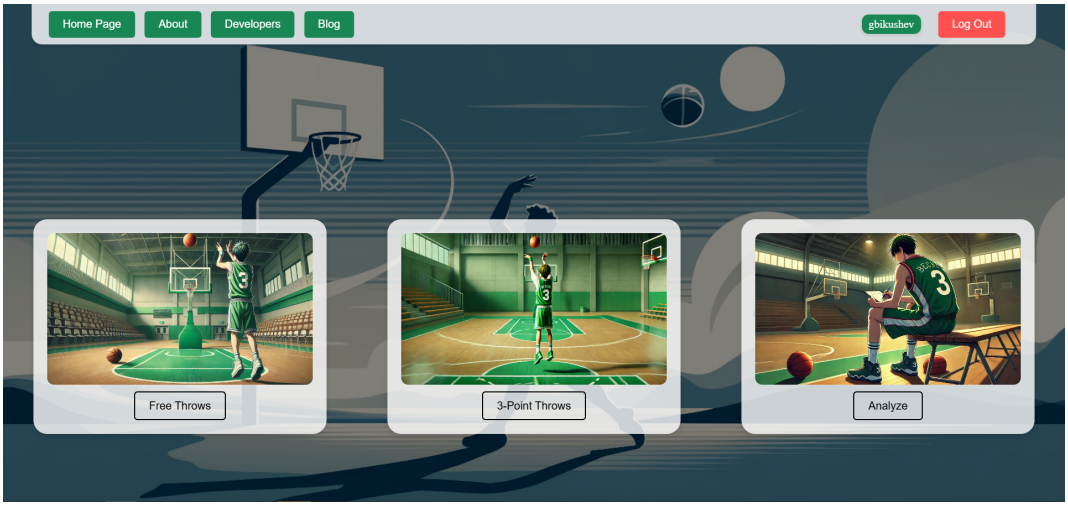


Figure 15: Homepage when users are logged in (their username is shown in the header)

– **About**: takes you to the page that describes in detail the functionality of the web app, what are the different buttons responsible for and video requirements for users (Figure 16).

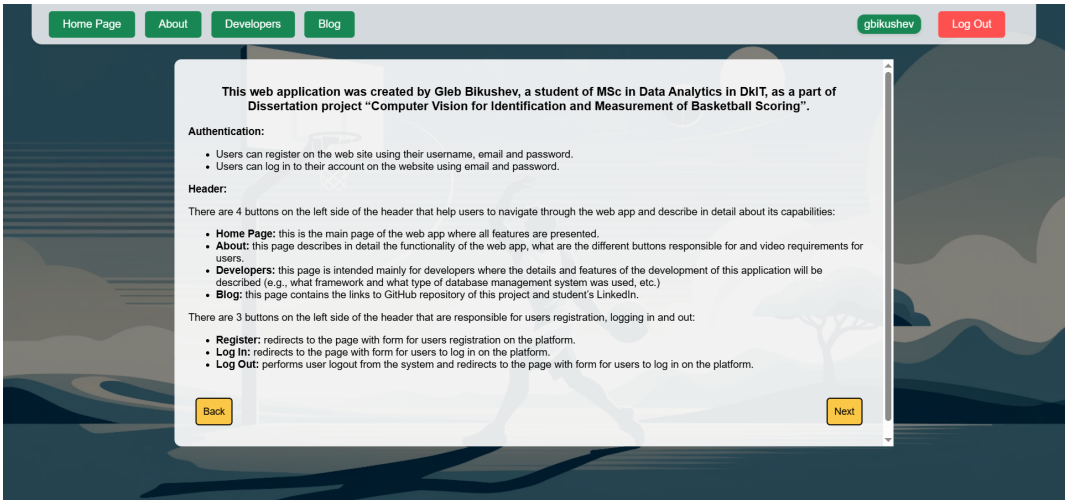


Figure 16: "About" page

– **Developers**: takes you to the page that is intended mainly for developers where the details and features of the development of this application will be described (e.g. API documentation, database setup, etc.) (Figure 17).

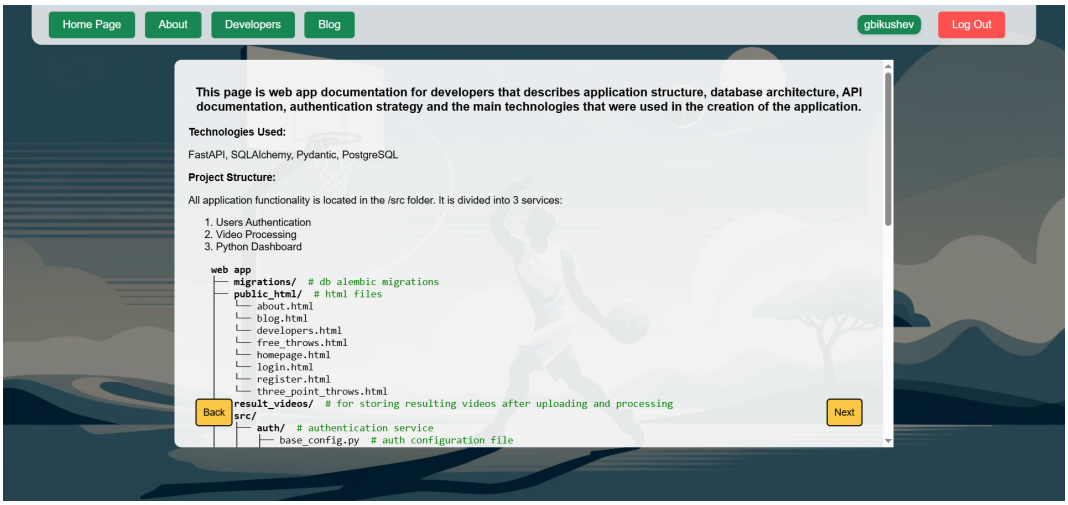


Figure 17: "Developers" page

– **Blog**: takes you to the page that contains the links to GitHub repository of this project and student's LinkedIn (Figure 18).

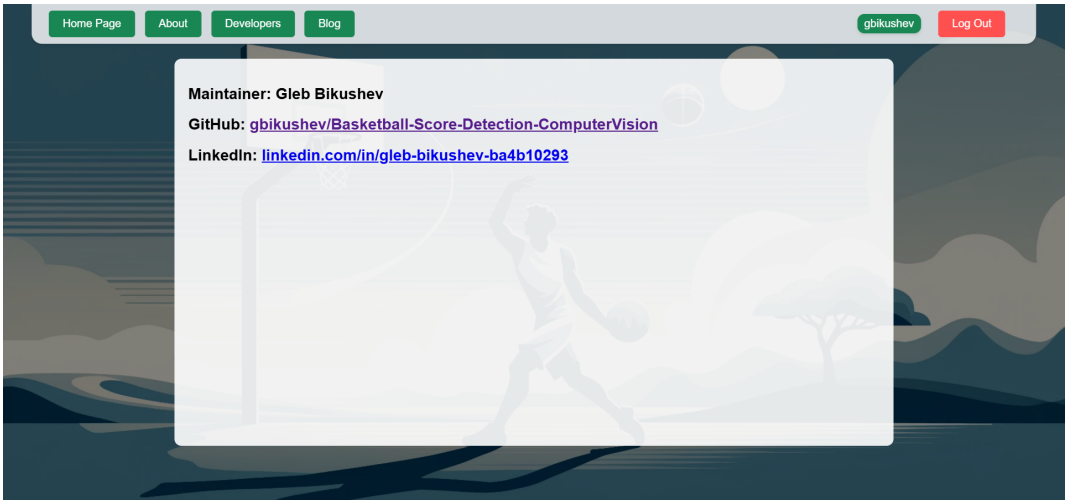


Figure 18: "Blog" page

There are 3 buttons on the left side of the header that are responsible for users registration, logging in and out:

– **Register**: redirects to the page with form for users registration on the platform.

– **Log In**: redirects to the page with form for users to log in on the platform.

– **Log Out**: performs user logout from the system and redirects to the page with form for users to log in on the platform.

6.2.3 Functionality

There are 3 tabs on the homepage users can access after logging in to their account:

– **Free Throws**: on this page users can upload their videos of taking free throw shots to the server, where they are processed by custom YOLOv8 deep learning model in combination with unique algorithm for identifying the successful shots. While the video is being uploaded and processed, users can track its progress (in %) on an informative progress bar. Users can also open the table with statistics of all uploaded free throws videos and for each of them users can observe following information:

Video Name	Number of Throws	Number of Successful Shots	Number of Misses	Shooting Accuracy	Upload Date

The ability is enabled for users to open the uploaded video after processing by algorithms and see exactly how the calculating the number of goals and shots is going. Also, users can delete uploaded videos from the list, and they will also be deleted on the server. (Figures 19, 20, 21)

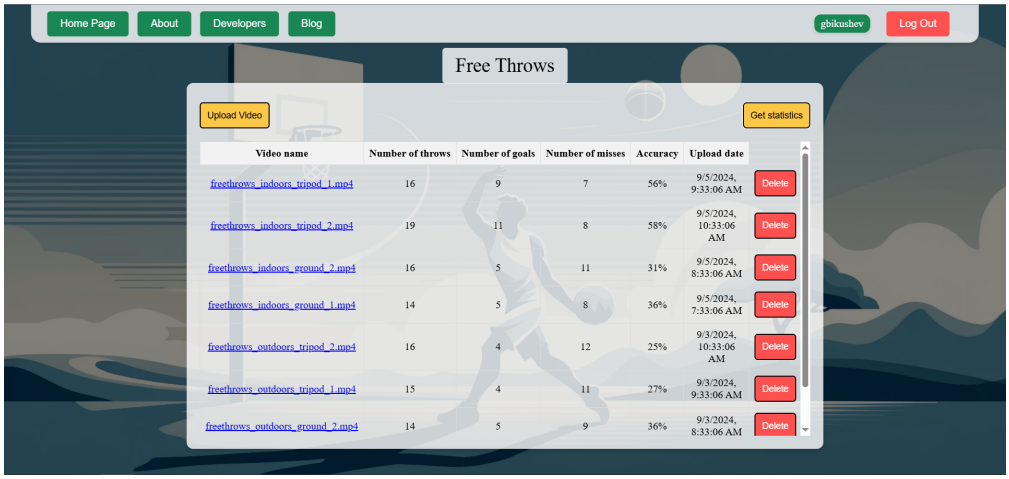


Figure 19: "Free Throws" page

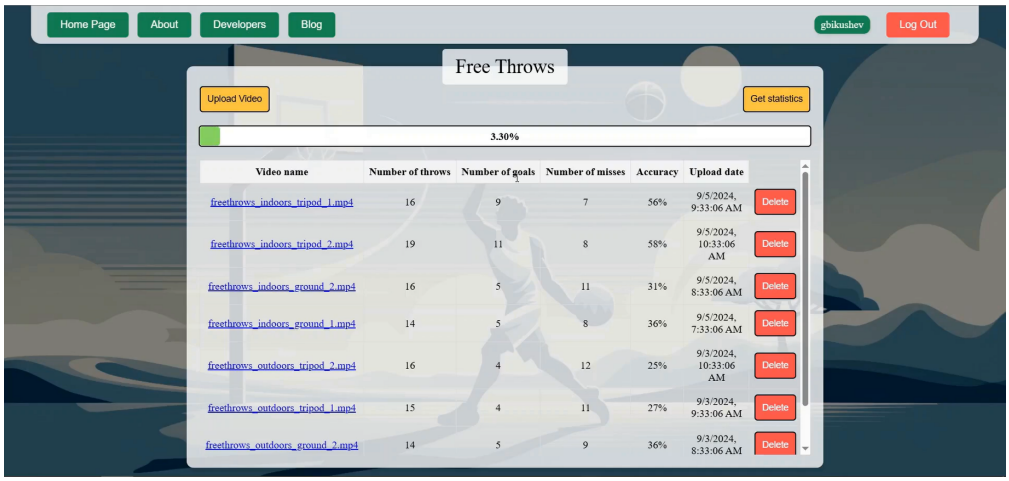


Figure 20: "Free Throws" page (video uploading stage)

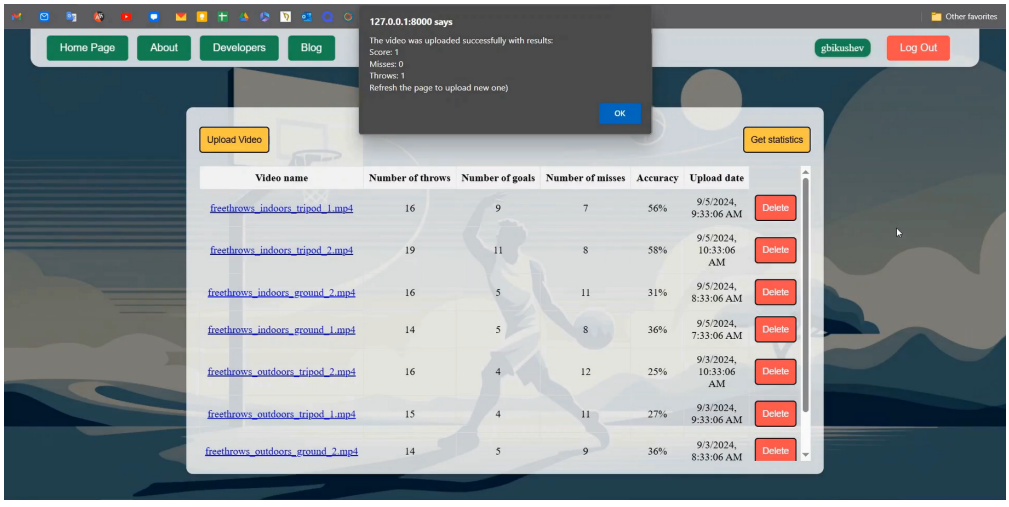


Figure 21: Message with performance stats after the video was uploaded

– **3-Point Throws**: this page is almost the same as for free throws, but it is designed to upload videos with three-point shots. It was decided to separate these 2 types of shots since the difficulty of successful shot of these 2 types of throws is very different: scoring a 3-point shot is much more difficult. So, it's more convenient to track shooting accuracy of them separately. (Figure 22)

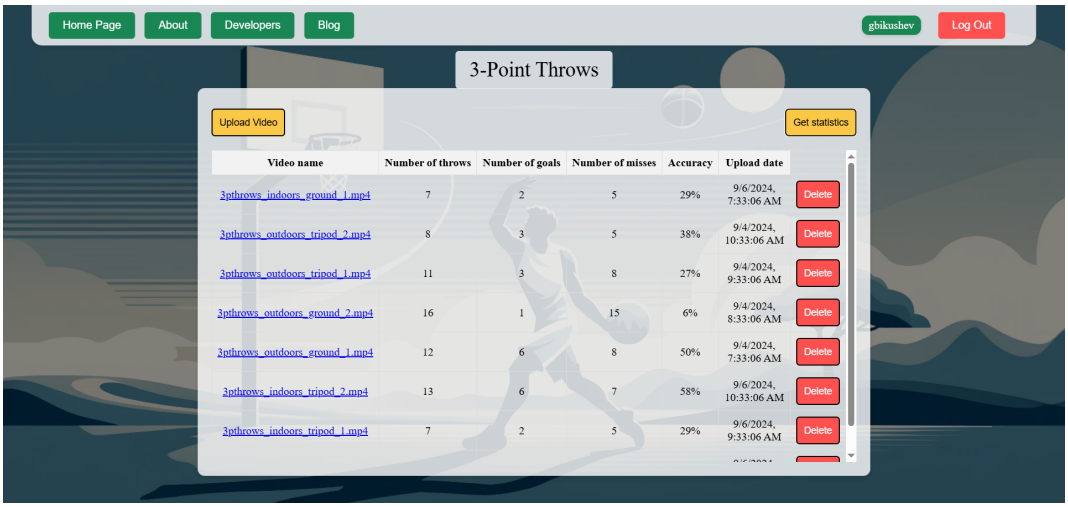


Figure 22: "3-Point Throws" page

– **Analyze**: this page is an information panel that was created to help users better track their basketball shooting performance over a specified period through interactive visualization. The user can filter and view data based on different types of shots (Free Throws, 3-Point Throws, or All) and can choose the method of filtering data either by date range or number of recent videos. The visualization consists of the following:

1. Bar chart (Total Goals and Misses): this bar chart displays the distribution of successful goals versus missed shots over the selected time period.
2. Line Chart (Shooting accuracy over time): this line chart tracks the accuracy of shots over the selected date range.

Dashboard screenshots are presented in Figures 23, 24, 25

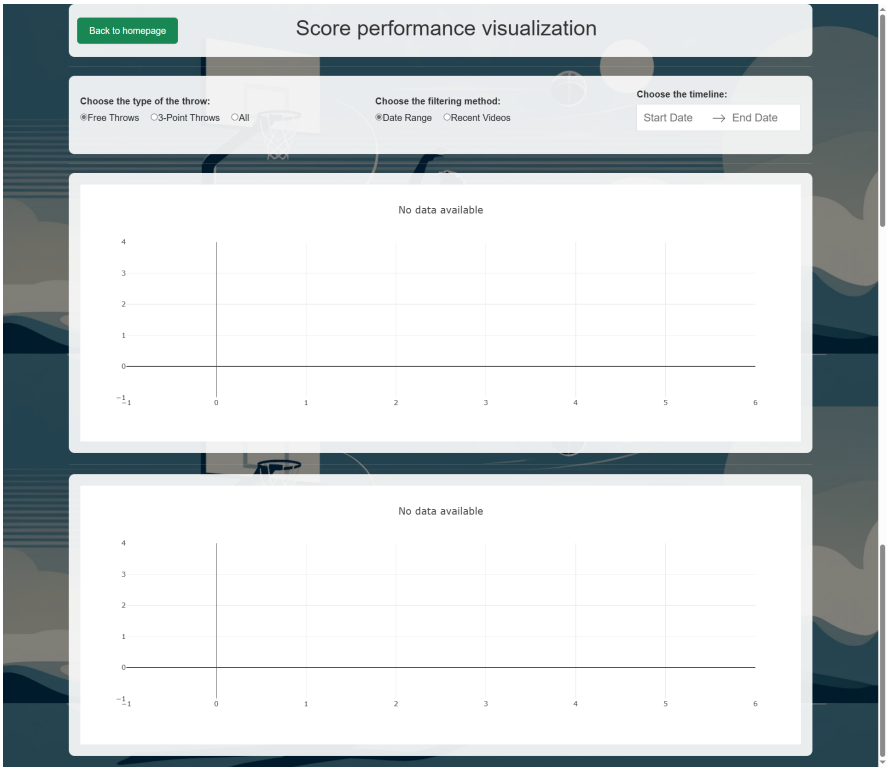


Figure 23: Dashboard while no videos uploaded

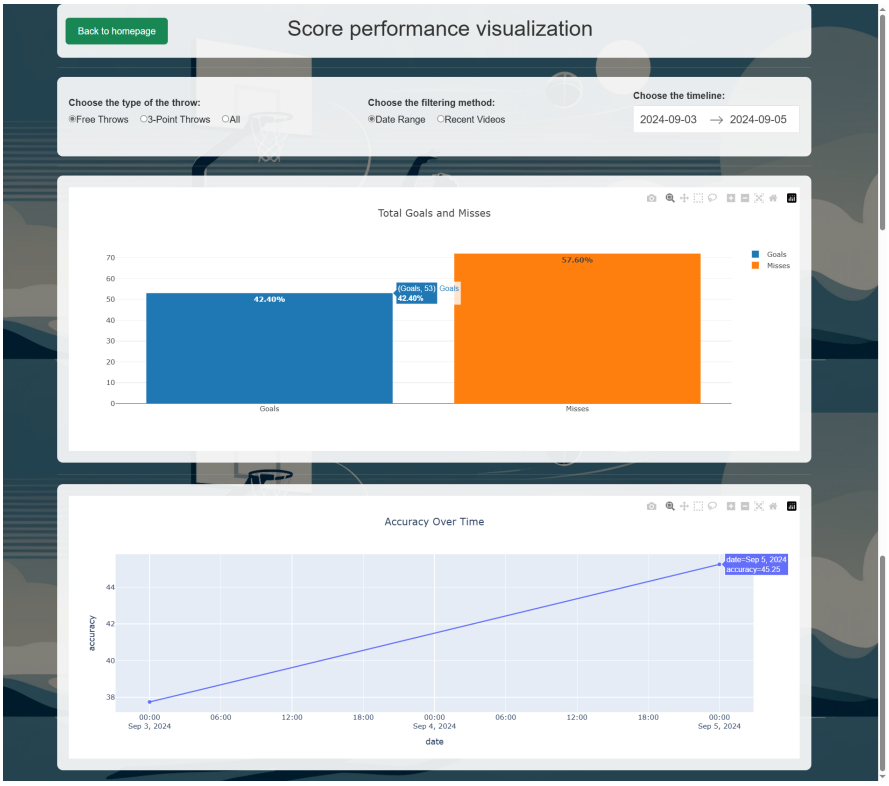


Figure 24: Dashboard for free throws with "Date Range" filtering method

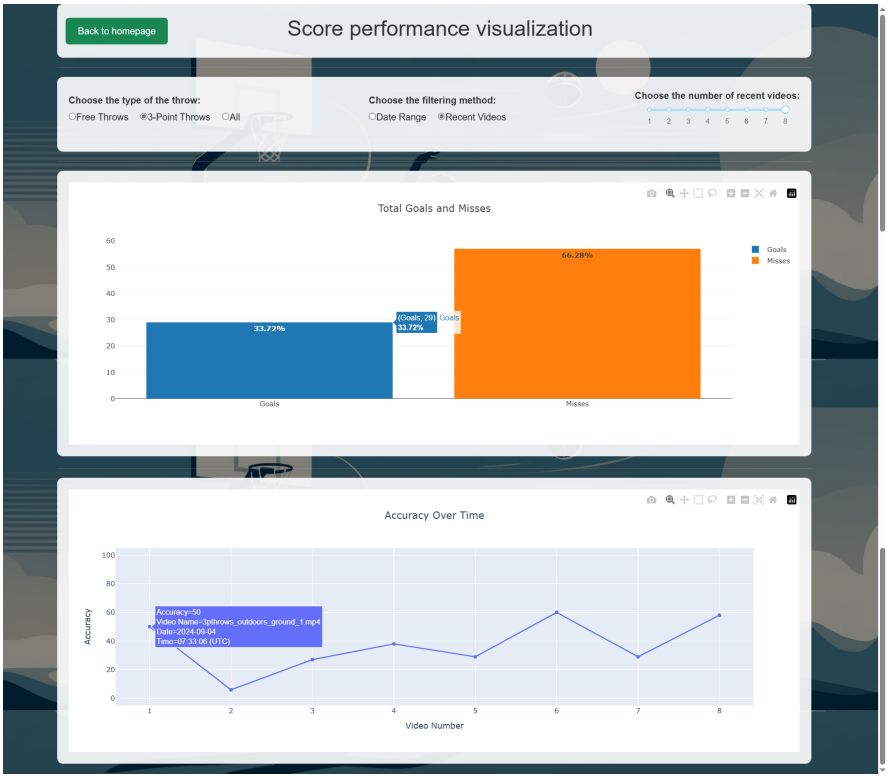


Figure 25: Dashboard for 3-point throws with "Recent Videos" filtering method

6.3 Database

I use PostgreSQL database management system in combination with Alembic (the database migration tool for Python) and SQLAlchemy (the Python SQL toolkit and Object-Relational Mapping (ORM) library).

Alembic helps manage changes to the database schema over time, allowing developers to apply, track, and undo schema changes in a systematic and version-controlled manner.

SQLAlchemy provides tools for interacting with databases in a more Pythonic way, abstracting much of the template code that would otherwise be required for executing SQL queries and managing database schemas. Thus, in this project SQLAlchemy is used for such database operations as creating and modifying tables (adding new entries or deleting them)

6.3.1 Tables

Tables:

1. **alembic_version**: this table is used to store the current version of the database schema. This version corresponds to the most recent Alembic migration that has been applied to the database.

   Columns:

   – *version_num* (Primary Key, type: varchar (32)): column stores a unique identifier (usually a string of characters) that represents the specific migration version.

2. **role**: this table defines the different types of roles available in the role-based access control system of users for the web application, such as 'user' and 'admin'. Each role corresponds to a specific set of permissions or access levels.

   Columns:

   – *id* (Primary Key, type: integer): column uniquely identifies each role, serving as the primary key.

- *name* (type: varchar): the column stores the name of the role, such as 'user' or 'admin'. The *name* is used to label the role and define what permissions or level of access a user associated with that role has within the system.
- *permission* (type: json): this column is meant to store a JSON object that could define specific permissions associated with the role.

3. **user**: this table manages user information for the web application.

   Columns:

   - *id* (Primary Key, type: integer): this is a unique identifier for each user in the table, serving as the primary key.
   - *email* (type: varchar): stores the email address of the user.
   - *username* (type: varchar): stores the username chosen by the user.
   - *registered_at* (type: timestamp without time zone): records the date and time when the user registered on the platform (UTC time zone).
   - *role_id* (Foreign Key, type: integer): this column references the *id* column in the "role" table, linking each user to a specific role (e.g., user, admin)
   - *hashed_password* (type: varchar): stores the hashed version of the user's password. Hashing is used to secure passwords so that they aren't stored in plain text.
   - *is_active* (type: boolean): indicates whether the user's account is active. An active account means the user can log in and use the application.
   - *is_superuser* (type: boolean): determines whether the user has superuser privileges, which includes administrative access to the entire system.
   - *is_verified* (type: boolean): indicates whether the user's email has been verified.

4. **video**: the table stores information about uploaded videos from all users including score performance statistics captured after processing the video by web application.

   Columns:

   - *id* (Primary Key, type: integer): this is a unique identifier for each video in the table, serving as the primary key.
   - *name_video* (type: varchar): the name of the video that user has uploaded.
   - *video_size* (type: integer): the size of the video in bytes that user has uploaded.
   - *number_throws* (type: integer): the number of the basketball shots made in the video calculated by "Throws and Goals Detection" algorithm.
   - *number_goals* (type: integer): the number of the successful basketball shots made in the video calculated by "Throws and Goals Detection" algorithm.
   - *number_misses* (type: integer): the number of the missed basketball shots made in the video calculated by "Throws and Goals Detection" algorithm.
   - *accuracy* (type: integer): the total basketball shooting accuracy within the uploaded video. Calculated as:
     $$\text{accuracy} = round(\frac{\text{number\_goals}}{\text{number\_throws}} * 100)$$
     If the number_throws = 0, accuracy is 0.
   - *throw_type* (type: integer): the certain type of basketball shots throughout the video. This column can only take the values "free-throws" and "three-point-throws".
   - *date_upload* (type: type: timestamp without time zone): records the date and time when the video was uploaded on the platform (UTC time zone).
   - *user_id* (Foreign Key, type: integer): this column references the *id* column in the "user" table, linking each video to a specific user.
   - *video_path* (type: varchar): the path to a directory within the project root folder where all the uploaded videos are stored.

7 Conclusion

The central research question asked how computer vision techniques could be adapted and incorporated into a web application to recognize and analyze basketball shooting activities. Through the integration of a custom-trained YOLOv8 model into a web-based framework, your project demonstrates how computer vision can effectively detect players, basketballs, and hoops, even under varying environmental conditions such as different court locations and video setups. The successful development of this web application, which processes user-uploaded training videos and provides statistical feedback on shot accuracy, confirms that the primary research question has been answered. The system's ability to automatically recognize basketball-related activities marks a significant achievement in applying computer vision to basketball training analytics.

One of the supporting questions addressed the extent to which the accuracy and reliability of a YOLOv8 model could be improved to detect and track players, balls, and hoops. Through careful dataset preparation, including diverse videos from YouTube Shorts and contributions from external participants, the model was trained to handle various court setups, camera positions, and dynamic basketball scenarios. The results indicated high precision and recall in object detection, particularly for basketballs and hoops, although some challenges remain, such as detecting objects under occlusion. Nevertheless, the training process successfully improved model performance, confirming that this supporting question has been answered.

The framework developed for detecting basketball throws and goals, based on YOLOv8's output and the algorithm's logic, demonstrated reliable results. The methodology involved defining rectangular zones around the hoop and analyzing the interactions between the ball and these zones. The evaluation of the algorithm on the testing dataset revealed satisfactory performance, with the system achieving high accuracy in detecting throws and goals. However, some areas for improvement, such as handling cases where the player obscures the hoop or improving detection under challenging conditions, were noted. Nonetheless, the framework effectively achieves its purpose, addressing this supporting question.

The web application was successfully developed using the FastAPI framework, allowing users to upload their training videos and receive detailed shot statistics. The application processes videos efficiently and securely, offering a user-friendly platform for athletes to track their performance over time. This aspect of the project fully meets the research objective of building an integrated solution for basketball performance analytics.

Although the project hypothesized that quantitative feedback would positively impact basketball training habits, the evaluation of this impact was not a primary focus in the current work. The web application's ability to track performance over time allows athletes to monitor improvements, but specific research into how this feedback influences training behaviors and outcomes remains to be explored. As such, this supporting question may benefit from further investigation in future research phases.

In conclusion, this Master's dissertation successfully addressed the core research question by demonstrating the adaptation of computer vision techniques for analyzing basketball shooting activities through a web application. The supporting questions were also largely addressed, particularly concerning the improvement of model accuracy, the development of the detection framework, and the successful integration into a user-friendly web application. Although some areas, such as the quantitative impact of feedback on training, warrant further exploration, the project lays a solid foundation for future advancements in sports analytics. Overall, the dissertation represents a significant contribution to the intersection of computer vision, data analytics, and basketball training.

8 Future Work

The current version of the algorithm demonstrates promising performance in detecting and classifying basketball throws and successful shots, but there are several areas for improvement to further enhance its accuracy, robustness, and applicability. Below are potential directions for future work, which aim to address existing limitations and expand the capabilities of the system.

8.1 Enhancing YOLO Model Capabilities for Ball Detection

One key area of improvement is the accuracy of the YOLO model in detecting and tracking the basketball. The performance of the overall algorithm in detecting throws and classifying successful and unsuccessful shots relies heavily on the accurate detection of the ball's position within the frame. To improve YOLO's ability to detect the ball across different conditions, it is crucial to increase the size and diversity of the training dataset. This can be achieved by including more instances of basketballs in varying colors, sizes, and textures, which will enable the model to generalize better to different ball appearances.

Additionally, the dataset should incorporate more challenging scenarios, such as outdoor environments where objects like cones, players, or background elements could be mistaken for the ball. The inclusion of these diverse scenarios will allow YOLO to distinguish the basketball more reliably under difficult conditions. Improving the model's accuracy in these diverse conditions will directly enhance the overall performance of the algorithm in both throw detection and the classification of successful and unsuccessful shots. Importantly, this would also enable the system to handle videos featuring basketballs of different colors, making it more flexible for use by different users who may not always have standard basketballs.

8.2 Improving Throws and Score Detection Accuracy

To improve the overall accuracy of throw and score detection, there are several enhancements that can be made to the system:

One approach is to add a second camera positioned at a different angle. A dual-camera system would provide a more comprehensive view of the court, allowing the algorithm to capture the ball's trajectory from multiple perspectives. This setup would reduce the risk of false positives, such as situations where the ball crosses in front of the hoop but does not go through. By combining data from both cameras, the system could more accurately assess the ball's path and make better decisions about whether a throw resulted in a successful goal. For instance, while one camera may struggle with certain angles, the second camera would provide a different viewpoint, improving the accuracy of detection.

Another method to improve accuracy is by integrating a Support Vector Machine (SVM) into the detection pipeline. SVMs are known for their effectiveness in classification tasks, especially when dealing with high-dimensional data. In the context of basketball detection, SVM could be used to classify whether a detected shot is successful based on features such as ball trajectory, speed, and position. This would add an extra layer of verification, reducing false positives and improving overall robustness. By working in conjunction with YOLO, SVM could enhance the system's ability to make more accurate predictions, particularly in borderline cases where the ball may have crossed near the hoop without entering.

A final way to improve the accuracy of goal detection is by adding a new class to the YOLO model specifically for detecting when the ball is inside the hoop. This would serve as an additional verification layer, improving the accuracy of goal detection by explicitly recognizing the event of a successful goal. Implementing this feature, however, would require a large, well-curated dataset with examples of the ball in various positions relative to the hoop. Given the relatively low frequency of successful shots in a typical basketball game, collecting enough data for training this class would be a challenge. Nevertheless, if successfully implemented, this feature could drastically reduce the number of false positives where the ball passes in front of the hoop without actually entering, providing a more reliable goal detection system.

8.3 Detecting Multiple Balls for Enhanced Training Efficiency

A useful feature for basketball training would be the ability to detect multiple balls in a video. In real training scenarios, athletes often shoot with several basketballs to avoid retrieving the ball after every shot. By allowing the system to detect two or more balls simultaneously, training sessions could become more productive, as the athlete could focus on shooting without interruptions. To implement this feature, the algorithm of throws and score detection needs to be adjusted to detect and track multiple instances of the "ball" class, ensuring that each ball is accurately identified and tracked in real-time.

8.4 Handling Situations Where the Athlete Obscures the Hoop

Another challenge that the algorithm currently faces is when the athlete obstructs the camera's view of the basketball hoop, making it difficult for the system to accurately assess the outcome of a throw. To address this, future work could explore methods such as leveraging temporal data or incorporating occlusion-aware models that predict the position of the ball and hoop even when part of the scene is blocked. Advanced tracking techniques that use previous frames to infer the position of the ball relative to the hoop could help mitigate this issue. Additionally, employing machine learning models that can handle partial occlusion of key objects in the scene could improve the algorithm's ability to classify shots accurately, even when visibility is compromised.

8.5 Impact of Quantitative Feedback on Basketball Training

Another promising direction for future work is to investigate the impact of providing quantitative feedback on basketball training habits and performance improvements. The current system offers real-time statistics on shot accuracy and outcomes, but further research is needed to evaluate how this feedback influences athletes' training behaviors. By tracking performance over time and offering detailed insights into shooting trends, the system could help players identify areas of improvement, such as shot consistency or accuracy from different court locations.

To fully understand the impact, future iterations of the system could include user studies or experiments where athletes use the system regularly, and their performance improvements are monitored over time. Feedback could be tailored to provide actionable insights, such as personalized drills based on the player's weak spots or motivational feedback to encourage continued practice. Studying how players engage with these feedback mechanisms and whether they lead to measurable improvements in performance would provide valuable insights into the effectiveness of integrating technology into basketball training routines.

8.6 Future Work Conclusion

By improving the ball detection capabilities of the YOLO model, incorporating new algorithms like SVM, adding new features such as dual-camera setups and multi-ball detection, and evaluating the impact of quantitative feedback on training, the algorithm's accuracy, reliability, and applicability can be significantly enhanced. These improvements would make the system more robust in a variety of conditions, leading to better performance in identifying throws and classifying successful and unsuccessful shots. Furthermore, addressing the challenge of occlusion, integrating more diverse datasets, and understanding how feedback influences training behaviors will ensure the system's effectiveness in real-world applications, making it more valuable for athletes, coaches, and other users.

9 GitHub

Here is the link to GitHub repository of this project: GitHub

10 References

References

[1] David Acuna. "Towards real-time detection and tracking of basketball players using deep neural networks". In: *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA*. 2017, pp. 4–9.

[2] Ilhan Aydin and Nashwan Adnan Othman. "A new IoT combined face detection of people by using computer vision for security application". In: *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE. 2017, pp. 1–6.

[3] Alex Bewley et al. "Simple online and realtime tracking". In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 3464–3468.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[5] Zhe Cao et al. "Realtime multi-person 2d pose estimation using part affinity fields". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7291–7299.

[6] Chien-Chang Chen et al. "Video based basketball shooting prediction and pose suggestion system". In: *Multimedia Tools and Applications* 82.18 (2023), pp. 27551–27570.

[7] Ivan Ćirić et al. "INTELLIGENT COMPUTER VISION SYSTEM FOR SCORE DETECTION IN BASKETBALL". In: *Facta Universitatis, Series: Automatic Control and Robotics* 22.2 (2024), pp. 075–085.

[8] Yutao Cui et al. "SportsMOT: A large multi-object tracking dataset in multiple sports scenes". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 9921–9931.

[9] Patrick Dendorfer et al. "Mot20: A benchmark for multi object tracking in crowded scenes". In: *arXiv preprint arXiv:2003.09003* (2020).

[10] Bauyrzhan Doskarayev et al. "Development of Computer Vision-enabled Augmented Reality Games to Increase Motivation for Sports". In: *International Journal of Advanced Computer Science and Applications* 14.4 (2023).

[11] Xu-Bo Fu, Shao-Long Yue, and De-Yun Pan. "Camera-based basketball scoring detection using convolutional neural network". In: *International Journal of Automation and Computing* 18.2 (2021), pp. 266–276.

[12] Xubo Fu et al. "Multiple player tracking in basketball court videos". In: *Journal of Real-Time Image Processing* 17 (2020), pp. 1811–1828.

[13] HomeCourt. *HomeCourt: The Basketball App*. Available at: https://www.homecourt.ai/. Accessed: 4 June 2024. 2022.

[14] Swish Hoop. *About SwishHoop*. Available at: https://www.swishhoop.com/about. Accessed: 4 June 2024. n.d.

[15] Shihao Hou et al. "A Basketball Training Posture Monitoring Algorithm Based on Machine Learning and Artificial Intelligence". In: *Mobile Information Systems* 2022 (2022).

[16] Chuyi Li et al. "YOLOv6: A single-stage object detection framework for industrial applications". In: *arXiv preprint arXiv:2209.02976* (2022).

[17] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[18] Anton Milan et al. "MOT16: A benchmark for multi-object tracking". In: *arXiv preprint arXiv:1603.00831* (2016).

[19] Banoth Thulasya Naik, Mohammad Farukh Hashmi, and Neeraj Dhanraj Bokde. "A comprehensive review of computer vision in sports: Open issues, future trends and research directions". In: *Applied Sciences* 12.9 (2022), p. 4429.

[20] Sen Qiao, Yilin Wang, and Jian Li. "Real-time human gesture grading based on OpenPose". In: *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE. 2017, pp. 1–6.

[21] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[22] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[23] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[24] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015).

[25] Shuji Tanikawa and Norio Tagawa. "Player Tracking using Multi-viewpoint Images in Basketball Analysis." In: *VISIGRAPP (5: VISAPP)*. 2020, pp. 813–820.

[26] Graham Thomas et al. "Computer vision for sports: Current applications and research topics". In: *Computer Vision and Image Understanding* 159 (2017), pp. 3–18.

[27] Ultralytics. *YOLOv5*. https://github.com/ultralytics/yolov5. Accessed: 12 June 2024. 2024.

[28] Ultralytics. *YOLOv8*. https://github.com/ultralytics/ultralytics. Accessed: 12 June 2024. 2024.

[29] Athanasios Voulodimos et al. "Deep learning for computer vision: A brief review". In: *Computational intelligence and neuroscience* 2018 (2018).

[30] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 7464–7475.

[31] Jianbo Wang et al. "Ai coach: Deep human pose estimation and analysis for personalized athletic training assistance". In: *Proceedings of the 27th ACM international conference on multimedia*. 2019, pp. 374–382.

[32] Ben G Weinstein. "A computer vision for animal ecology". In: *Journal of Animal Ecology* 87.3 (2018), pp. 533–545.

[33] Jiongen Xiao, Wenchun Tian, and Liping Ding. "Basketball action recognition method of deep neural network based on dynamic residual attention mechanism". In: *Information* 14.1 (2022), p. 13.

[34] Young Yoon et al. "Analyzing basketball movements and pass relationships using realtime object tracking techniques based on deep learning". In: *IEEE Access* 7 (2019), pp. 56564–56576.

[35] Yifu Zhang et al. "Bytetrack: Multi-object tracking by associating every detection box". In: *European conference on computer vision*. Springer. 2022, pp. 1–21.

11 Appendix

11.1 Web Application Project Structure

All application functionality is located in the /src folder. It is divided into 3 services:

1. Users Authentication
2. Video Processing
3. Python Dashboard

```
web app/
|-- migrations/ # db alembic migrations
|-- public_html/ # html files
|   |-- about.html
|   |-- blog.html
|   |-- developers.html
|   |-- free_throws.html
|   |-- homepage.html
|   |-- login.html
|   |-- register.html
|   |-- three_point_throws.html
|-- result_videos/ # for storing resulting videos after uploading and processing
|-- src/
|   |-- auth/ # authentication service
|   |   |-- base_config.py # auth configuration file
|   |   |-- manager.py # user manager
|   |   |-- models.py # db models
|   |   |-- schemas.py # pydantic models
|   |   |-- utils.py # utilities
|   |-- dashboard/ # building python dashboard
|   |   |-- dashboard.py # Dash App
|   |-- process_video/ # service for video processing
|   |   |-- classes.py # classes for video processing
|   |   |-- connection_manager.py # websocket connection manager
|   |   |-- functions.py # functions for video processing
|   |   |-- models.py # db models
|   |   |-- router_videos.py # router stores endpoints for video processing
|   |   |-- video_func.py # functions for extracting info from video
|   |-- __init__.py
|   |-- config.py # global configs
|   |-- database.py # db connection related stuff
|-- static/
|   |-- css/ # css files
|   |   |-- dashboard.css
|   |   |-- header.css
|   |   |-- homepage.css
|   |   |-- login.css
|   |   |-- main.css
|   |   |-- register.css
|   |   |-- throws_page.css
|   |-- images/
|   |-- js/ # js files
|   |   |-- header.js
|   |   |-- homepage.js
|   |   |-- login.js
|   |   |-- register.js
|-- videos/ # videos for testing
|-- .env
|-- alembic.ini
|-- app.py # main file launching the project
|-- best.pt # YOLO model
|-- requirements.txt # requirements
```

11.2 API documentation

11.2.1 Interactive API Docs

This application provides automatically generated interactive API documentation, accessible through the following paths:

- Swagger UI: Available at /docs, this interface allows users to interact with the API endpoints directly from the browser. It provides a user-friendly way to explore the API, send requests, and view responses.
- ReDoc: Available at /redoc, this interface offers a more detailed and structured view of the API documentation.

11.2.2 Endpoints Overview

The API is divided into different groups based on functionality:

1. **General Routes**
   - **/dashboard-redirect** (GET): Redirect to Dash App with transferring Cookies. Authentication is required (JWT).
     Common errors:
     - *Code 401* (Unauthorized if JWT token is missing or invalid)
   - **/free-throws** (GET): Get Free Throws Page (free_throws.html)
   - **/three-point-throws** (GET): Get Three Point Throws Page (three_point_throws.html)
   - **/register** (GET): Get Registration Page (register.html)
   - **/login** (GET): Get Login Page (login.html)
   - **/homepage** (GET): Get Home Page (homepage.html)
   - **/about** (GET): Get About Page (about.html)
   - **/developers** (GET): Get Developers Page (developers.html)
   - **/blog** (GET): Get Blog Page (blog.html)
   - **/protected-route** (GET): Return username of the current user. Authentication is required (JWT).
     Common errors:
     - *Code 401* (Unauthorized if JWT token is missing or invalid)
2. **Authentication Routes**
   - **/auth/jwt/login** (POST): Authenticates a user, giving a JWT token.
     Request Body:

```
{
    "username": "user@example.com",
    "password": "string"
}

```

     Successful Response:
     - *Code 204* (No Content)

     Common errors:
     - *Code 400* (Error: Bad Request, if user with such username and password is not found in the database):

```
{
    "detail": "LOGIN_BAD_CREDENTIALS"
}

```

- **/auth/jwt/logout** (POST): User logout using JWT. Authentication is required (JWT).
  Successful Response:
  - *Code 204* (No Content)

  Common errors:
  - *Code 401* (Error: Unauthorized, if user is not authorized):

```
{
    "detail": "Unauthorized"
}
```

- **/auth/forgot-password** (POST): Request password reset (not used in my application).
- **/auth/reset-password** (POST): Reset password using a token (not used in my application).
- **/auth/register** (POST): Registers a new user.
  Request Body:

```
{
    "username": "string",
    "email": "user@example.com",
    "password": "string"
}
```

  Successful Response:
  - *Code 201*:

```
{
    "id": 1,
    "email": " user@example.com",
    "is_active": true,
    "is_superuser": false,
    "is_verified": false,
    "username": "string",
    "role_id": 1
}
```

  Common errors:
  - *Code 400* (Error: Bad Request, if user already exists):

```
{
    "detail": "REGISTER_USER_ALREADY_EXISTS"
}
```

  - *Code 422* (Error: Unprocessable Entity, if the "email" value doesn't contain single sign):

```
{
  "detail": [
    {
      "type": "value_error",
      "loc": [
        "body",
        "email"
      ],
      "msg": "value is not a valid email address: The email address is not valid. It must have exactly one @-sign.",
      "input": "test123",
      "ctx": {
        "reason": "The email address is not valid. It must have exactly one @-sign."
      }
    }
  ]
}
```

3. **Video Management Routes**

- **/videos/statistics/{throw_type}** (GET): Authenticates a user, giving a JWT token.
  Path Parameters:
  - *throw_type* (string):
    "free-throw": for videos containing free throws.
    "three-point-throws": for videos containing 3-point throws.
    "all": for videos containing both types of throws.

  Successful Response (Example: throw_type = free-throws):
  - *Code 200*:

```
[
    {
        "id": 11,
        "name_video": "yarik_tripid_throws_1.mp4",
        "video_size": 1125993,
        "number_throws": 2,
        "number_goals": 0,
        "number_misses": 2,
        "accuracy": 0,
        "throw_type": "free-throws",
        "date_upload": "2024-07-29T23:17:29.468100",
        "user_id": 1,
        "video_path": "result_videos\\result_yarik_tripid_throws_1.mp4"
    },
    {
        "id": 12,
        "name_video": "yarik_tripid_throws_2.mp4",
        "video_size": 1663929,
        "number_throws": 1,
        "number_goals": 1,
        "number_misses": 0,
        "accuracy": 100,
        "throw_type": "free-throws",
        "date_upload": "2024-07-29T23:38:40.071928",
        "user_id": 1,
        "video_path": "result_videos\\result_yarik_tripid_throws_2.mp4"
    },
]

```

  Common Errors:
  - *Code 401* (Error: Unauthorized, if user is not authorized):

```
{
    "detail": "Unauthorized"
}

```

- **/videos/upload_video_endpoint/{throw_type}** (POST): Upload video on the platform and start video processing. Authentication is required (JWT).
  Path Parameters:
  - *throw_type* (string):
    "free-throw": for videos containing free throws.
    "three-point-throws": for videos containing 3-point throws.
    "all": for videos containing both types of throws.

  Successful Response (Example: throw_type = free-throws):
  - *Code 200*:

```
{
    "success": true,
    "message": "Video processing started",
    "task_id": "256b09dc-93e5-4ecf-9b56-42fa95503d20"
}

```

Common Errors:

- *Code 400* (Error: Bad Request, if user chooses file not in .mp4 format):

```
{
    "detail": "Invalid file format. Acceptable format .mp4"
}
```

- *Code 401* (Error: Unauthorized, if user is not authorized):

```
{
    "detail": "Unauthorized"
}
```

- **/videos/get_video_processing_results/{task_id}** (GET): Get results of a certain task of video processing, adds a new entry to the "video" table with retrieved shooting performance statistics. After getting the results, this task_id this value is permanently deleted. Authentication is required (JWT).

  Path Parameters:

  - *task_id* (string): a previous endpoint starts the video processing task and returns the id of this task as a string.

  Successful Response (Example: task_id = 256b09dc-93e5-4ecf-9b56-42fa95503d20):

  - *Code 200*:

```
{
    "success": true,
    "message": "Video processing completed",
    "score": 1,
    "misses": 0,
    "throws": 1
}
```

  Common Errors:

  - *Code 404* (Error: Not Found, if the task_id is not found):

```
{
    "detail": "Task ID not found"
}
```

  - *Code 401* (Error: Unauthorized, if user is not authorized):

```
{
    "detail": "Unauthorized"
}
```

- **/videos/delete_video/{video_path}** (DELETE): Delete a video from table "video" in the database and from the directory where all processed videos are stored.

  Path Parameters:

  - *video_path* (string): the path to a directory within the project root folder where all the uploaded videos are stored.

  Successful Response (Example: video_path = result_videos\result_yarik1.mp4):

  - *Code 200*:

```
{
    "success": true
}
```

  Common Errors:

  - *Code 404* (Error: Not Found, if the video_path is not found):

```
{
    "detail": "Video not found"
}
```

– *Code 401* (Error: Unauthorized, if user is not authorized):

```
{
    "detail": "Unauthorized"
}
```

- **/videos/stop_video_processing/{task_id}** (POST): Stop video processing task. At my application this endpoint is triggered when the user refreshes the page to reset the video processing task (For example, when the user finds out that he is uploading the wrong file).
  Path Parameters:
  – *task_id* (string): id of the video processing task.
  Successful Response (Example: task_id = e02bad05-ed31-472d-a0f8-aad83d3a336a):
  – *Code 200*:

```
{
    "success": true,
    "message": "Video processing stopped"
}
```

  Common Errors:
  – *Code 404* (Error: Not Found, if the task_id is not found):

```
{
    "detail": "Invalid task_id"
}
```

  – *Code 401* (Error: Unauthorized, if user is not authorized):

```
{
    "detail": "Unauthorized"
}
```

11.3 Authentication In Web Application

11.3.1 Overview

In this web application, user authentication and management are handled using the fastapi-users python library. This library simplifies the integration of user authentication, including handling user registration, login, and password management, with built-in support for JWT (JSON Web Token) authentication.

11.3.2 JWT Authentication Strategy

The application utilizes JWT (JSON Web Tokens) for authenticating users. JWT is a compact and self-contained way for securely transmitting information between parties as a JSON object.

When a user logs in through the /auth/jwt/login endpoint, their credentials (email/username and password) are verified. Upon successful authentication, the server generates a JWT token, which contains encoded information about the user, such as their ID, roles, and token expiration time. This JWT token stored in the cookie is automatically transmitted with every subsequent HTTP request from the client to the server such as API calls to fetch data, submit forms, or navigate to protected routes. This allows the server to verify the user's identity and permissions for each request. The token is included in the header of HTTP requests, formatted as follows (example):

```
    cookie: basketball=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiIxIiwiYXVkIjpbImZhc3RhcGktdXNlcnM6YXV0aCJdLCJleHAiOjE3MjQyNTM0MD19.7
    W_5BlnIb1FVRHk6bqz8flCNKRlH8DaIEN8nHN3Vum0
```

When the server receives a request, it checks the JWT token in the cookie to ensure that it is valid (e.g. not expired, correctly signed, etc.). If the token is valid, the server identifies the user and processes the request according to the user's permissions and roles. If the token is invalid or expired, the server will return an error response, typically a 401 Unauthorized status, prompting the user to log in again.

11.4 Code Source

11.4.1 HTML files (web_app/public_html/)

about.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About</title>
    <link rel="stylesheet" href="static/css/main.css">
    <link rel="stylesheet" href="static/css/header.css">
    <style>
    /* Additional CSS for the text pages */
    body {
        font-family: Arial, sans-serif;
        font-size: 15px;
    }

    h3 {
        text-align: center;
    }

    .content-container {
        width: 1000px; /* Fixed width */
        height: 520px; /* Fixed height */
        margin: 20px auto;
        padding: 20px;
        padding-bottom: 20px; /* Adjust padding to avoid content overlap with buttons
    */
        overflow: auto; /* Add scrollbar if content exceeds container height */
        background: rgba(255, 255, 255, 0.9);
        border-radius: 10px;
        margin-bottom: 20px;
        position: relative; /* Establish positioning context for child elements */
    }

    .text-page {
        display: none;
    }

    .text-page.active {
        display: block;
    }

    .nav-buttons {
        display: flex;
        justify-content: space-between;
        gap: 20px;
        position: sticky; /* Change from absolute to sticky */
        bottom: 0; /* Stick to the bottom */
        left: 0;
        right: 0;
        background-color: rgba(255, 255, 255, 0); /* Optional: Add a background to
    make the buttons visible over content */
        padding: 10px;
    }

    .nav-buttons button {
        padding: 10px;
        background-color: rgb(250, 199, 70);
        color: black;
```

```
        border: 2px solid black;
        border-radius: 5px;
        cursor: pointer;
        font-size: 14px;
        transition: background-color 0.3s, color 0.3s;
    }

    .nav-buttons button:hover {
        background-color: black;
        color: white;
    }

    </style>
</head>
<body>
    <div class="header">
        <div class="left-buttons">
            <button id="home-button">Home Page</button>
            <button id="about-button">About</button>
            <button id="developers-button">Developers</button>
            <button id="blog-button">Blog</button>
        </div>
        <div class="right-buttons">
            <div class="username-frame" id="username-frame">
                <span id="username-display"></span>
            </div>
            <button id="login-button">Log In</button>
            <button id="register-button">Register</button>
            <button id="logout-button" style="display: none;">Log Out</button>
        </div>
    </div>

    <div class="content-container">
        <div id="page1" class="text-page active">
            <h3>This web application was created by Gleb Bikushev, a student of MSc in Data Analytics in DkIT, as a part of Dissertation project  Computer  Vision for Identification and Measurement of Basketball Scoring .</h3>

            <p><strong>Authentication:</strong></p>
            <ul>
                <li>Users can register on the web site using their username, email and password.</li>
                <li>Users can log in to their account on the website using email and password.</li>
            </ul>

            <p><strong>Header:</strong></p>
            <p>There are 4 buttons on the left side of the header that help users to navigate through the web app and describe in detail about its capabilities:</p>
            <ul>
                <li><strong>Home Page:</strong> this is the main page of the web app where all features are presented.</li>
                <li><strong>About:</strong> this page describes in detail the functionality of the web app, what are the different buttons responsible for and video requirements for users.</li>
                <li><strong>Developers:</strong> this page is intended mainly for developers where the details and features of the development of this application will be described (e.g., what framework and what type of database management system was used, etc.)</li>
                <li><strong>Blog:</strong> this page contains the links to GitHub repository of this project and students LinkedIn.</li>
            </ul>
            <p>There are 3 buttons on the left side of the header that are responsible for users registration, logging in and out:</p>
            <ul>
                <li><strong>Register:</strong> redirects to the page with form for users registration on the platform.</li>
                <li><strong>Log In:</strong> redirects to the page with form for users to log in on the platform.</li>
                <li><strong>Log Out:</strong> performs user logout from the system and redirects to the page with form for users to log in on the platform.</li>
```

```
            </ul>
            <br> <br>

        </div>

        <div id="page2" class="text-page">
            <p><strong>Functionality:</strong></p>
            <p>There are 3 tabs on the homepage users can access after logging in to their account:</p>
            <ol>
                <li><strong>Free Throws:</strong> on this page users can upload their videos of taking free throw shots to the server, where they are processed by a custom YOLOv8 deep learning model in combination with a unique algorithm for identifying successful shots. While the video is being uploaded and processed, users can track its progress (in %) on an informative progress bar. Users can also open the table with statistics of all uploaded free throws videos and observe the following information for each video:</li>


                <table style="width:87%; border-collapse: collapse; margin-left: auto; margin-right: auto; font-size: 13px;">
                    <thead>
                        <tr style="background-color: #f2f2f2; text-align: left;">
                            <th style="padding: 10px; border: 1px solid #ddd;">Video Name</th>
                            <th style="padding: 10px; border: 1px solid #ddd;">Number of Throws</th>
                            <th style="padding: 10px; border: 1px solid #ddd;">Number of Goals</th>
                            <th style="padding: 10px; border: 1px solid #ddd;">Number of Misses</th>
                            <th style="padding: 10px; border: 1px solid #ddd;">Shooting Accuracy</th>
                            <th style="padding: 10px; border: 1px solid #ddd;">Upload Date</th>
                        </tr>
                    </thead>
                </table>

                <p>The ability is enabled for users to open the uploaded video after processing by algorithms and see exactly how the calculation of the number of goals and shots is going. Also, users can delete uploaded videos from the list, and they will also be deleted on the server.</p>
            </ol>

            <ol start="2">
                <li><strong>3-Point Throws:</strong> this page is similar to the Free Throws page but is designed for videos with three-point shots. These two types of shots are separated since scoring a 3-point shot is much more difficult than a free throw, making it more convenient to track shooting accuracy separately.</li>
                <br>
                <li><strong>Analyze:</strong> this page is an information panel created to help users better track their basketball shooting performance over a specified period through interactive visualization. The user can filter and view data based on different types of shots (Free Throws, 3-Point Throws, or All) and can choose the method of filtering data either by date range or number of recent videos. The visualization consists of:
                    <ul>
                        <li><strong>Bar Chart (Total Goals and Misses):</strong> this bar chart displays the distribution of successful goals versus missed shots over the selected time period.</li>
                        <li><strong>Line Chart (Shooting accuracy over time):</strong> this line chart tracks the accuracy of shots over the selected date range.</li>
                    </ul>
                </li>
            </ol>
            <br> <br>
        </div>

        <div id="page3" class="text-page">
            <p><strong>Video Requirements:</strong></p>
```

```
            <ol>
                <li>Each shot must end with a person touching the ball on camera, so this is considered as the end of the throw.</li>
                <li>During the throw, a person should not stand in front of the camera and obscure the visibility of the basketball hoop. This may lead to inaccurate results.</li>
                <li>If there are several basketball hoops in the video, shots and goals will be counted in the largest hoop in the frame.</li>
                <li>There must be a net on the basketball hoops throughout the video.</li>
                <li>There must be only one ball throughout the video.</li>
                <li>The color of the ball must be orange or at least close to it.</li>
                <li>The environment in the video can be both indoors and outdoors.</li>
                <li>The camera must be stationary throughout the entire video recording.</li>
                <li>The camera can be positioned on the ground or mounted at some height using a tripod.</li>
                <li>The camera must be in positions on the court marked with red dots (Picture 1).</li>
            </ol>

            <div style="text-align: center; margin-top: 20px;">
                <img src="/static/images/court_schema.png" alt="Court Diagram" style="max-width: 70%; height: auto;">
            </div>
        </div>

        <div class="nav-buttons">
            <button id="back-button">Back</button>
            <button id="next-button">Next</button>
        </div>
    </div>

    <script>
        // JavaScript for pagination
        let currentPage = 1;
        const totalPages = 3;

        function showPage(page) {
            document.querySelectorAll('.text-page').forEach((pageElement, index) => {
                pageElement.classList.remove('active');
                if (index + 1 === page) {
                    pageElement.classList.add('active');
                }
            });
        }

        document.getElementById('next-button').addEventListener('click', () => {
            if (currentPage < totalPages) {
                currentPage++;
                showPage(currentPage);
            }
        });

        document.getElementById('back-button').addEventListener('click', () => {
            if (currentPage > 1) {
                currentPage--;
                showPage(currentPage);
            }
        });

        // Initial page load
        showPage(currentPage);
    </script>
    <script src="static/js/header.js"></script>
</body>
</html>
```

Listing 1: about.html

blog.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About</title>
    <link rel="stylesheet" href="static/css/main.css">
    <link rel="stylesheet" href="static/css/header.css">
    <style>
    /* Additional CSS for the text pages */
    body {
        font-family: Arial, sans-serif;
        font-size: 15px;
    }

    h3 {
        text-align: center;
    }

    .content-container {
        width: 1000px; /* Fixed width */
        height: 520px; /* Fixed height */
        margin: 20px auto;
        padding: 20px;
        padding-bottom: 20px; /* Adjust padding to avoid content overlap with buttons */
        overflow: auto; /* Add scrollbar if content exceeds container height */
        background: rgba(255, 255, 255, 0.9);
        border-radius: 10px;
        margin-bottom: 20px;
        position: relative; /* Establish positioning context for child elements */
    }


    </style>
</head>
<body>
    <div class="header">
        <div class="left-buttons">
            <button id="home-button">Home Page</button>
            <button id="about-button">About</button>
            <button id="developers-button">Developers</button>
            <button id="blog-button">Blog</button>
        </div>
        <div class="right-buttons">
            <div class="username-frame" id="username-frame">
                <span id="username-display"></span>
            </div>
            <button id="login-button">Log In</button>
            <button id="register-button">Register</button>
            <button id="logout-button" style="display: none;">Log Out</button>
        </div>
    </div>

    <div class="content-container">
        <h2>Maintainer: Gleb Bikushev</h2>
        <h2>
            GitHub: <a href="https://github.com/gbikushev/Basketball-Score-Detection-ComputerVision" target="_blank">
                gbikushev/Basketball-Score-Detection-ComputerVision
            </a>
        </h2>
        <h2>
            LinkedIn: <a href="https://linkedin.com/in/gleb-bikushev-ba4b10293" target="_blank">
                linkedin.com/in/gleb-bikushev-ba4b10293
            </a>
        </h2>
    </div>
    <script src="static/js/header.js"></script>
</body>
```

```
</html>
```

Listing 2: blog.html

developers.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About</title>
    <link rel="stylesheet" href="static/css/main.css">
    <link rel="stylesheet" href="static/css/header.css">
    <style>
    /* Additional CSS for the text pages */
    body {
        font-family: Arial, sans-serif;
        font-size: 15px;
    }

    h3 {
        text-align: center;
    }

    .content-container {
        width: 1000px; /* Fixed width */
        height: 520px; /* Fixed height */
        margin: 20px auto;
        padding: 20px;
        padding-bottom: 20px; /* Adjust padding to avoid content overlap with buttons
    */
        overflow: auto; /* Add scrollbar if content exceeds container height */
        background: rgba(255, 255, 255, 0.9);
        border-radius: 10px;
        margin-bottom: 20px;
        position: relative; /* Establish positioning context for child elements */
    }

    .text-page {
        display: none;
    }

    .text-page.active {
        display: block;
    }

    .nav-buttons {
        display: flex;
        justify-content: space-between;
        gap: 20px;
        position: sticky; /* Change from absolute to sticky */
        bottom: 0; /* Stick to the bottom */
        left: 0;
        right: 0;
        background-color: rgba(255, 255, 255, 0); /* Optional: Add a background to
    make the buttons visible over content */
        padding: 10px;
    }

    .nav-buttons button {
        padding: 10px;
        background-color: rgb(250, 199, 70);
        color: black;
        border: 2px solid black;
        border-radius: 5px;
        cursor: pointer;
        font-size: 14px;
        transition: background-color 0.3s, color 0.3s;
    }

    .nav-buttons button:hover {
```

```
        background-color: black;
        color: white;
    }

    code {
        font-family: Consolas,"courier new";
        color: rgb(74, 71, 72);
        background-color: #f1f1f1;
        padding: 2px;
        font-size: 95%;
    }

    u {
        text-decoration: underline;
    }

    </style>
</head>
<body>
    <div class="header">
        <div class="left-buttons">
            <button id="home-button">Home Page</button>
            <button id="about-button">About</button>
            <button id="developers-button">Developers</button>
            <button id="blog-button">Blog</button>
        </div>
        <div class="right-buttons">
            <div class="username-frame" id="username-frame">
                <span id="username-display"></span>
            </div>
            <button id="login-button">Log In</button>
            <button id="register-button">Register</button>
            <button id="logout-button" style="display: none;">Log Out</button>
        </div>
    </div>

    <div class="content-container">


        <div id="page1" class="text-page active">
            <h3>This page is web app documentation for developers that describes application structure, database architecture, API documentation, authentication strategy and the main technologies that were used in the creation of the application.</h3>

            <p><strong>Technologies Used:</strong></p>
            <p>FastAPI, SQLAlchemy, Pydantic, PostgreSQL</p>

            <p><strong>Project Structure:</strong></p>
            <p>All application functionality is located in the /src folder. It is divided into 3 services:</p>
            <ol>
                <li>Users Authentication</li>
                <li>Video Processing</li>
                <li>Python Dashboard</li>
            </ol>
            <pre>
    <span style="font-weight: bold;">web app</span>
    |-- <span style="font-weight: bold;">migrations/</span>  <span style="color: green;"># db alembic migrations</span>
    |-- <span style="font-weight: bold;">public_html/</span>  <span style="color: green;"># html files</span>
    |   |-- about.html
    |   |-- blog.html
    |   |-- developers.html
    |   |-- free_throws.html
    |   |-- homepage.html
    |   |-- login.html
    |   |-- register.html
    |   |-- three_point_throws.html
    |-- <span style="font-weight: bold;">result_videos/</span>  <span style="color:
```

```
    green;"># for storing resulting videos after uploading and processing</span>
|-- <span style="font-weight: bold;">src/</span>
|   |-- <span style="font-weight: bold;">auth/</span>  <span style="color: green;"># authentication service</span>
|   |   |-- base_config.py  <span style="color: green;"># auth configuration file</span>
|   |   |-- manager.py  <span style="color: green;"># user manager</span>
|   |   |-- models.py  <span style="color: green;"># db models</span>
|   |   |-- schemas.py   <span style="color: green;"># pydantic models</span>
|   |   |-- utils.py  <span style="color: green;"># utilities</span>
|   |-- <span style="font-weight: bold;">dashboard/</span>  <span style="color: green;"># building python dashboard</span>
|   |   |-- dashboard.py  <span style="color: green;"># Dash App</span>
|   |-- <span style="font-weight: bold;">process_video/</span>  <span style="color: green;"># service for video processing</span>
|   |   |-- classes.py  <span style="color: green;"># classes for video processing</span>
|   |   |-- connection_manager.py  <span style="color: green;"># websocket connection manager</span>
|   |   |-- functions.py  <span style="color: green;"># functions for video processing</span>
|   |   |-- models.py  <span style="color: green;"># db models</span>
|   |   |-- router_videos.py  <span style="color: green;"># router stores endpoints for video processing</span>
|   |   |-- video_func.py  <span style="color: green;"># functions for extracting info from video</span>
|   |-- __init__.py
|   |-- config.py  <span style="color: green;"># global configs</span>
|   |-- database.py  <span style="color: green;"># db connection related stuff</span>
|-- <span style="font-weight: bold;">static/</span>
|   |-- <span style="font-weight: bold;">css/</span>  <span style="color: green;"># css files</span>
|   |   |-- dashboard.css
|   |   |-- header.css
|   |   |-- homepage.css
|   |   |-- login.css
|   |   |-- main.css
|   |   |-- register.css
|   |   |-- throws_page.css
|   |-- <span style="font-weight: bold;">images/</span>
|   |-- <span style="font-weight: bold;">js/</span>  <span style="color: green;"># js files</span>
|   |   |-- header.js
|   |   |-- homepage.js
|   |   |-- login.js
|   |   |-- register.js
|-- <span style="font-weight: bold;">videos/</span>  <span style="color: green;"># videos for testing</span>
|-- .env  <span style="color: green;"># environment variables</span>
|-- alembic.ini
|-- app.py  <span style="color: green;"># main file</span>
|-- best.pt  <span style="color: green;"># YOLO model</span>
|-- requirements.txt  <span style="color: green;"># requirements</span>
        </pre>
    </div>
    <div id="page2" class="text-page">

        <h3>Database</h3>
        <p>I use PostgreSQL database management system in combination with Alembic (the database migration tool for Python) and SQLAlchemy (the Python SQL toolkit and Object-Relational Mapping (ORM) library).</p>
        <p>Alembic helps manage changes to the database schema over time, allowing developers to apply, track, and undo schema changes in a systematic and version-controlled manner.</p>
        <p>SQLAlchemy provides tools for interacting with databases in a more Pythonic way, abstracting much of the template code that would otherwise be required for executing SQL queries and managing database schemas.</p>

        <h4>Tables</h4>
        <pre>
```

```
1) <span style="font-weight: bold;">    alembic_version    </span>: this table is used to store the current version of the database schema.
Columns:
- <span style="font-style: italic;">version_num</span> (Primary Key, type: character varying (32)): column stores a unique identifier that represents the specific migration version.
<br>
2) <span style="font-weight: bold;">    role    </span>: this table defines the different types of roles available in the role-based access control system of users for the web application.
Columns:
- <span style="font-style: italic;">id</span> (Primary Key, type: integer): column uniquely identifies each role, serving as the primary key.
- <span style="font-style: italic;">name</span> (type: varchar): stores the name of the role, such as 'user' or 'admin'.
- <span style="font-style: italic;">permission</span> (type: json): stores a JSON object that defines specific permissions associated with the role.
<br>
3) <span style="font-weight: bold;">    user    </span>: this table manages user information for the web application.
Columns:
- <span style="font-style: italic;">id</span> (Primary Key, type: integer): a unique identifier for each user in the table.
- <span style="font-style: italic;">email</span> (type: character varying): stores the email address of the user.
- <span style="font-style: italic;">username</span> (type: character varying): stores the username chosen by the user.
- <span style="font-style: italic;">registered_at</span> (type: timestamp without time zone): records the date and time when the user registered.
- <span style="font-style: italic;">role_id</span> (Foreign Key, type: integer): references the `id` column in the "role" table, linking each user to a specific role.
- <span style="font-style: italic;">hashed_password</span> (type: character varying): stores the hashed version of the user's password.
- <span style="font-style: italic;">is_active</span> (type: boolean): indicates whether the user's account is active.
- <span style="font-style: italic;">is_superuser</span> (type: boolean): determines whether the user has superuser privileges.
- <span style="font-style: italic;">is_verified</span> (type: boolean): indicates whether the user's email has been verified.
<br>
4) <span style="font-weight: bold;">    video    </span>: stores information about uploaded videos, including performance statistics.
Columns:
- <span style="font-style: italic;">id</span> (Primary Key, type: integer): a unique identifier for each video.
- <span style="font-style: italic;">name_video</span> (type: character varying): the name of the uploaded video.
- <span style="font-style: italic;">video_size</span> (type: integer): the size of the video in bytes.
- <span style="font-style: italic;">number_throws</span> (type: integer): the number of basketball shots made in the video.
- <span style="font-style: italic;">number_goals</span> (type: integer): the number of successful basketball shots.
- <span style="font-style: italic;">number_misses</span> (type: integer): the number of missed basketball shots.
- <span style="font-style: italic;">accuracy</span> (type: integer): the total basketball shooting accuracy.
- <span style="font-style: italic;">throw_type</span> (type: integer): the type of basketball shots in the video.
- <span style="font-style: italic;">date_upload</span> (type: integer): records the date and time when the video was uploaded.
- <span style="font-style: italic;">user_id</span> (Foreign Key, type: integer): references the `id` column in the "user" table.
- <span style="font-style: italic;">video_path</span> (type: character varying): the path where all uploaded videos are stored.
            </pre>

        </div>
        <div id="page3", class="text-page">
            <h3>API documentation</h3>
```

```
224             <h4>Interactive API Docs</h4>
225             <p>This application provides automatically generated interactive API
    documentation, accessible through the following paths:</p>
226             <ul>
227                 <li><span style="font-weight: bold;">Swagger UI</span>: Available at /
    docs, this interface allows users to interact with the API endpoints directly from
    the browser. It provides a user-friendly way to explore the API, send requests, and
     view responses.</li>
228                 <li><span style="font-weight: bold;">ReDoc</span>: Available at /redoc
    , this interface offers a more detailed and structured view of the API
    documentation. </li>
229             </ul>
230             <h4>Endpoint Overview</h4>
231             <p>The API is divided into different groups based on functionality:</p>
232             <p><span style="font-weight: bold;">1. General Routes</span></p>
233             <ul style="list-style-type:disc;">
234                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/
    dashboard-redirect</span> (GET): Redirect to Dash App with transferring Cookies.
    Authentication is required (JWT).
235                     <br><u>Common errors:</u>
236                     <ul style="list-style-type:circle;">
237                         <li><span style="font-style: italic;">Code 401</span> (
    Unauthorized if JWT token is missing or invalid)</li>
238                     </ul>
239                 <br>
240                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/free-
    throws</span> (GET): Get Free Throws Page (free_throws.html)</li>
241                 <br>
242                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/three-
    point-throws</span> (GET): Get Three Point Throws Page (three_point_throws.html)</
    li>
243                 <br>
244                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/
    register</span> (GET): Get Registration Page (register.html)</li>
245                 <br>
246                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/login</
    span> (GET): Get Login Page (login.html)</li>
247                 <br>
248                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/
    homepage</span> (GET): Get Home Page (homepage.html)</li>
249                 <br>
250                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/about</
    span> (GET): Get About Page (about.html)</li>
251                 <br>
252                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/
    developers</span> (GET): Get Developers Page (developers.html)</li>
253                 <br>
254                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/blog</
    span> (GET): Get Blog Page (blog.html)</li>
255                 <br>
256                 <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/
    protected-route</span> (GET): Return username of the current user. Authentication
    is required (JWT).
257                     <br><u>Common errors:</u>
258                     <ul style="list-style-type:circle;">
259                         <li><span style="font-style: italic;">Code 401</span> (
    Unauthorized if JWT token is missing or invalid)</li>
260                     </ul>
261             </ul>
262 
263             <p><span style="font-weight: bold;">2. Authentication Routes</span></p>
264             <ul style="list-style-type:disc;">
265                 <li><span style="color: rgb(25, 126, 43); font-weight: bold;">/auth/
    jwt/login</span> (POST): Authenticates a user, giving a JWT token.
266                     <br><u>Request Body:</u><br>
267                     <code style="display: block; padding: 10px; background-color: #
    f0f0f0; border-radius: 5px;">
268                         {<br>
269                         <span style="padding-left: 20px;">"username": "user@example.
    com",</span><br>
270                         <span style="padding-left: 20px;">"password": "string"</span><
```

```
    br>
                        }<br>
                    </code>
                    <br><u>Successful Response:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 204</span> (No
    Content)</li>
                    </ul>
                    <br><u>Common errors:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 400</span> (Error:
    Bad Request, if user with such username and password is not found in the database):
    </li>
                        <code style="display: block; padding: 10px; background-color:
    #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"detail": "
    LOGIN_BAD_CREDENTIALS"</span><br>
                            }<br>
                        </code>
                    </ul>


                </li>
                <br>
                <li><span style="color: rgb(25, 126, 43); font-weight: bold;">/auth/
    jwt/logout</span> (POST): User logout using JWT. Authentication is required (JWT).
                    <br><u>Successful Response:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 204</span> (No
    Content)</li>
                    </ul>
                    <br><u>Common errors:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 401</span> (Error:
    Unauthorized, if user is not authorized):</li>
                        <code style="display: block; padding: 10px; background-color:
    #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"detail": "Unauthorized"
    </span><br>
                            }<br>
                        </code>
                    </ul>
                </li>
                <br>
                <li><span style="color: rgb(25, 126, 43); font-weight: bold;">/auth/
    forgot-password</span> (POST): Request password reset (not used in my application).
    </li>
                <br>
                <li><span style="color: rgb(25, 126, 43); font-weight: bold;">/auth/
    reset-password</span> (POST): Reset password using a token (not used in my
    application).</li>
                <br>
                <li><span style="color: rgb(25, 126, 43); font-weight: bold;">/auth/
    register</span> (POST): Registers a new user.
                    <br><u>Request Body:</u><br>
                    <code style="display: block; padding: 10px; background-color: #
    f0f0f0; border-radius: 5px;">
                        {<br>
                        <span style="padding-left: 20px;">"username": "string",</span>
    <br>
                        <span style="padding-left: 20px;">"email": "user@example.com",
    </span><br>
                        <span style="padding-left: 20px;">"password": "string"</span><
    br>
                        }<br>
                    </code>
                    <br><u>Successful Response:</u>
                    <ul style="list-style-type:circle;">
                    <li><span style="font-style: italic;">Code 201:</span></li>
```

```
                        <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"id": 1,</span><br>
                            <span style="padding-left: 20px;">"email": " user@example.com",</span><br>
                            <span style="padding-left: 20px;">"is_active": true,</span><br>
                            <span style="padding-left: 20px;">"is_superuser": false,</span><br>
                            <span style="padding-left: 20px;">"is_verified": false,</span><br>
                            <span style="padding-left: 20px;">"username": "string",</span><br>
                            <span style="padding-left: 20px;">"role_id": 1</span><br>
                            }<br>
                        </code>
                    </ul>
                    <br><u>Common errors:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 400</span> (Error: Bad Request, if user already exists):</li>
                        <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"detail": "REGISTER_USER_ALREADY_EXISTS"</span><br>
                            }<br>
                        </code>
                    </ul>
                </li>
            </ul>
            <p><span style="font-weight: bold;">3. Video Management Routes:</span></p>
            <ul style="list-style-type:disc;">
                <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/videos/statistics/{throw_type}</span> (GET): Authenticates a user, giving a JWT token.
                    <br><u>Path Parameters:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">throw_type (string):</li>
                        <ul style="list-style-type:none;">
                            <li>‘free-throw‘: for videos containing free throws.</li>
                            <li>‘three-point-throws‘: for videos containing 3-point throws</li>
                            <li>‘all‘: for videos containing both types of throws</li>
                        </ul>
                    </ul>
                    <br><u>Successful Response</u> (Example: throw_type = free-throws):
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 200:</span></li>
                        <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            [<br>
                            <span style="padding-left: 20px;">{</span><br>
                                <span style="padding-left: 30px;">"id": 11,</span><br>
                                <span style="padding-left: 30px;">"name_video": "yarik_tripid_throws_1.mp4",</span><br>
                                <span style="padding-left: 30px;">"video_size": 1125993,</span><br>
                                <span style="padding-left: 30px;">"number_throws": 2,</span><br>
                                <span style="padding-left: 30px;">"number_goals": 0,</span><br>
                                <span style="padding-left: 30px;">"number_misses": 2,</span><br>
                                <span style="padding-left: 30px;">"accuracy": 0,</span><br>
                                <span style="padding-left: 30px;">"throw_type": "free-throws",</span><br>
                                <span style="padding-left: 30px;">"date_upload": "
```

```
    2024-07-29T23:17:29.468100",</span><br>
                                <span style="padding-left: 30px;">"user_id": 1,</span>
    <br>
                                <span style="padding-left: 30px;">"video_path": "
    result_videos\\result_yarik_tripid_throws_1.mp4"</span><br>
                            <span style="padding-left: 20px;">},</span><br>
                            <span style="padding-left: 20px;">{</span><br>
                                <span style="padding-left: 30px;">"id": 12,</span><br>
                                <span style="padding-left: 30px;">"name_video": "
    yarik_tripid_throws_2.mp4",</span><br>
                                <span style="padding-left: 30px;">"video_size":
    1663929,</span><br>
                                <span style="padding-left: 30px;">"number_throws": 1,<
    /span><br>
                                <span style="padding-left: 30px;">"number_goals": 1,</
    span><br>
                                <span style="padding-left: 30px;">"number_misses": 0,<
    /span><br>
                                <span style="padding-left: 30px;">"accuracy": 100,</
    span><br>
                                <span style="padding-left: 30px;">"throw_type": "free-
    throws",</span><br>
                                <span style="padding-left: 30px;">"date_upload": "
    2024-07-29T23:38:40.071928",</span><br>
                                <span style="padding-left: 30px;">"user_id": 1,</span>
    <br>
                                <span style="padding-left: 30px;">"video_path": "
    result_videos\\result_yarik_tripid_throws_2.mp4"</span><br>
                            <span style="padding-left: 20px;">},</span><br>
                          ]<br>
                        </code>
                    </ul>
                    <br><u>Common Errors:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 401</span> (Error:
    Unauthorized, if user is not authorized):</li>
                        <code style="display: block; padding: 10px; background-color:
    #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"detail": "Unauthorized"
    </span><br>
                            }<br>
                        </code>
                    </ul>
                </li>
                <br>
                <li><span style="color: rgb(25, 126, 43); font-weight: bold;">/videos/
    upload_video_endpoint/{throw_type}</span> (POST): Upload video on the platform and
    start video processing. Authentication is required (JWT).
                    <br><u>Path Parameters:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">throw_type (string):</li
    >
                        <ul style="list-style-type:none;">
                            <li>‘free-throw‘: for videos containing free throws.</li>
                            <li>‘three-point-throws‘: for videos containing 3-point
    throws</li>
                            <li>‘all‘: for videos containing both types of throws</li>
                        </ul>
                    </ul>
                    <br><u>Successful Response</u> (Example: throw_type = free-throws)
    :
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 200:</span></li>
                            <code style="display: block; padding: 10px; background-
    color: #f0f0f0; border-radius: 5px;">
                                {<br>
                                <span style="padding-left: 20px;">"success": true,</
    span><br>
                                <span style="padding-left: 20px;">"message": "Video
    processing started",</span><br>
```

```
                                <span style="padding-left: 20px;">"task_id": "256b09dc-93e5-4ecf-9b56-42fa95503d20"</span><br>
                                }<br>
                            </code>
                    </ul>
                    <br><u>Common Errors:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 400</span> (Error: Bad Request, if user chooses file not in .mp4 format):</li>
                        <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            {<br>
                                <span style="padding-left: 20px;">"detail": "Invalid file format. Acceptable format .mp4"</span><br>
                            }<br>
                        </code>
                        <li><span style="font-style: italic;">Code 401</span> (Error: Unauthorized, if user is not authorized):</li>
                        <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"detail": "Unauthorized"</span><br>
                            }<br>
                        </code>
                    </ul>
                </li>
                <br>
                <li><span style="color: rgb(70, 70, 174); font-weight: bold;">/videos/get_video_processing_results/{task_id}</span> (GET): Get results of a certain task of video processing, adds a new entry to the "video" table with retrieved shooting performance statistics. After getting the results, this task_id this value is permanently deleted. Authentication is required (JWT).
                    <br><u>Path Parameters:</u>
                        <ul style="list-style-type:circle;">
                            <li><span style="font-style: italic;">task_id</span> (string): a previous endpoint starts the video processing task and returns the id of this task as a string.</li>
                        </ul>
                    <br><u>Successful Response</u> (Example: task_id = 256b09dc-93e5-4ecf-9b56-42fa95503d20):
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 200:</span></li>
                            <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                                {<br>
                                <span style="padding-left: 20px;">"success": true,</span><br>
                                <span style="padding-left: 20px;">"message": "Video processing completed",</span><br>
                                <span style="padding-left: 20px;">"score": 1,</span><br>
                                <span style="padding-left: 20px;">"misses": 0,</span><br>
                                <span style="padding-left: 20px;">"throws": 1</span><br>
                                }<br>
                            </code>
                    </ul>
                    <br><u>Common Errors:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 404</span> (Error: Not Found, if the task_id is not found):</li>
                            <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                                {<br>
                                <span style="padding-left: 20px;">"detail": "Task ID not found"</span><br>
                                }<br>
                            </code>
                        <li><span style="font-style: italic;">Code 401</span> (Error:
```

```
    Unauthorized, if user is not authorized):</li>
                        <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"detail": "Unauthorized"</span><br>
                            }<br>
                        </code>
                    </ul>
                </li>
                <br>
                <li><span style="color: rgb(171, 22, 22); font-weight: bold;">/videos/delete_video/{video_path}</span> (DELETE): Delete a video from table video in the database and from the directory where all processed videos are stored.
                    <br><u>Path Parameters:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">video_path</span> (string): the path to a directory within the project root folder where all the uploaded videos are stored.</li>
                    </ul>
                    <br><u>Successful Response</u> (Example: video_path = result_videos\result_yarik1.mp4):
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 200:</span></li>
                            <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"success": true</span><br>
                            }<br>
                        </code>
                    </ul>
                    <br><u>Common Errors:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 404</span> (Error: Not Found, if the video_path is not found):</li>
                            <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"detail": "Video not found"</span><br>
                            }<br>
                        </code>
                        <li><span style="font-style: italic;">Code 401</span> (Error: Unauthorized, if user is not authorized):</li>
                            <code style="display: block; padding: 10px; background-color: #f0f0f0; border-radius: 5px;">
                            {<br>
                            <span style="padding-left: 20px;">"detail": "Unauthorized"</span><br>
                            }<br>
                        </code>
                    </ul>
                </li>
                <br>
                <li><span style="color: rgb(25, 126, 43); font-weight: bold;">/videos/stop_video_processing/{task_id}</span> (POST): Stop video processing task. At my application this endpoint is triggered when the user refreshes the page to reset the video processing task (For example, when the user finds out that he is uploading the wrong file).
                    <br><u>Path Parameters:</u>
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">task_id</span> (string): id of the video processing task.</li>
                    </ul>
                    <br><u>Successful Response</u> (Example: task_id = e02bad05-ed31-472d-a0f8-aad83d3a336a):
                    <ul style="list-style-type:circle;">
                        <li><span style="font-style: italic;">Code 200:</span></li>
                            <code style="display: block; padding: 10px; background
```

```
    -color: #f0f0f0; border-radius: 5px;">
                                    {<br>
                                    <span style="padding-left: 20px;">"success": true,
    </span><br>
                                    <span style="padding-left: 20px;">"message": "
    Video processing stopped"</span><br>
                                    }<br>
                                </code>
                            </ul>
                        <br><u>Common Errors:</u>
                            <ul style="list-style-type:circle;">
                                <li><span style="font-style: italic;">Code 404</span> (
    Error: Not Found, if the task_id is not found):</li>
                                <code style="display: block; padding: 10px; background
    -color: #f0f0f0; border-radius: 5px;">
                                    {<br>
                                    <span style="padding-left: 20px;">"detail": "
    Invalid task_id"</span><br>
                                    }<br>
                                </code>
                                <li><span style="font-style: italic;">Code 401</span> (
    Error: Unauthorized, if user is not authorized):</li>
                                <code style="display: block; padding: 10px; background
    -color: #f0f0f0; border-radius: 5px;">
                                    {<br>
                                    <span style="padding-left: 20px;">"detail": "
    Unauthorized"</span><br>
                                    }<br>
                                </code>
                            </ul>
                    </li>
                </ul>

            </div>
            <div id="page4", class="text-page">
                <h3>Authentication</h3>
                <h4>Overview</h4>
                <p>In this web application, user authentication and management are handled
     using the fastapi-users python library. This library simplifies the integration of
     user authentication, including handling user registration, login, and password
    management, with built-in support for JWT (JSON Web Token) authentication.</p>
                <h4>JWT Authentication Strategy</h4>
                <p>The application utilizes JWT (JSON Web Tokens) for authenticating users
    . JWT is a compact and self-contained way for securely transmitting information
    between parties as a JSON object.</p>
                <h4>How JWT Authentication Works</h4>
                <p>When a user logs in through the /auth/jwt/login endpoint, their
    credentials (email/username and password) are verified. Upon successful
    authentication, the server generates a JWT token, which contains encoded
    information about the user, such as their ID, roles, and token expiration time.
    This JWT token stored in the cookie is automatically transmitted with every
    subsequent HTTP request from the client to the server such as API calls to fetch
    data, submit forms, or navigate to protected routes. This allows the server to
    verify the user's identity and permissions for each request.</p>
                <p>The token is included in the header of HTTP requests, formatted as
    follows (example):</p>
                <code>
                    cookie: basketball=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiIxIiwiYXVkIjpbImZhc3RhcGktdXNlcnM6YXV0aCJdLCJleHAiOjE3MjQyNTM0MDl9.<br>7
    W_5BlnIb1FVRHk6bqz8flCNKRlH8DaIEN8nHN3Vum0
                </code>
                <p>When the server receives a request, it checks the JWT token in the
    cookie to ensure that it is valid (e.g., not expired, correctly signed, etc.). If
    the token is valid, the server identifies the user and processes the request
    according to the user's permissions and roles. If the token is invalid or expired,
    the server will return an error response, typically a 401 Unauthorized status,
    prompting the user to log in again.</p>
            </div>
            <!-- Inserted content ends here -->

            <div class="nav-buttons">
```

```
            <button id="back-button">Back</button>
            <button id="next-button">Next</button>
        </div>
    </div>
    <script>
        // JavaScript for pagination
        let currentPage = 1;
        const totalPages = 4;

        function showPage(page) {
            document.querySelectorAll('.text-page').forEach((pageElement, index) => {
                pageElement.classList.remove('active');
                if (index + 1 === page) {
                    pageElement.classList.add('active');
                }
            });
        }

        document.getElementById('next-button').addEventListener('click', () => {
            if (currentPage < totalPages) {
                currentPage++;
                showPage(currentPage);
            }
        });

        document.getElementById('back-button').addEventListener('click', () => {
            if (currentPage > 1) {
                currentPage--;
                showPage(currentPage);
            }
        });

        // Initial page load
        showPage(currentPage);
    </script>
    <script src="static/js/header.js"></script>
</body>
</html>
```

Listing 3: developers.html

free_throws.html

```
<!DOCTYPE html>
<html lang="eng">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Upload video</title>
    <link rel="stylesheet" href="static/css/main.css">
    <link rel="stylesheet" href="static/css/header.css">
    <link rel="stylesheet" href="static/css/throws_page.css">
</head>
<body>
    <div class="header">
        <div class="left-buttons">
            <button id="home-button">Home Page</button>
            <button id="about-button">About</button>
            <button id="developers-button">Developers</button>
            <button id="blog-button">Blog</button>
        </div>
        <div class="right-buttons">
            <div class="username-frame" id="username-frame">
                <span id="username-display"></span>
            </div>
            <button id="login-button">Log In</button>
            <button id="register-button">Register</button>
            <button id="logout-button" style="display: none;">Log Out</button>
        </div>
    </div>
    <input type="file" id="file-input" accept="video/mp4" style="display: none;">
    <div class="throws-text">
```

```
        Free Throws
    </div>
    <div class="upload-container">
        <div class="buttons-container">
            <button type="button" onclick="triggerFileInput()">Upload Video</button>
            <button onclick="getStatistics()">Get statistics</button>
        </div>

        <div id="progressBarContainer">
            <div id="progressBar">
                <div id="progressTextContainer">
                    <span id="progressText">0%</span>
                </div>
            </div>
        </div>

        <div id="stats-container" class="stats-container">
            <table id="stats-table" style="display: none;">
                <thead>
                    <tr>
                        <th>Video name</th>
                        <th>Number of throws</th>
                        <th>Number of goals</th>
                        <th>Number of misses</th>
                        <th>Accuracy</th>
                        <th>Upload date</th>
                    </tr>
                </thead>
                <tbody id="stats-body">
                </tbody>
            </table>
        </div>
    </div>

    <script>
        const progressBarContainer = document.getElementById('progressBarContainer');
        const progressBar = document.getElementById('progressBar');
        const progressText = document.getElementById('progressText');
        const fileInput = document.getElementById('file-input');

        function checkFileFormat(filename) {
            const extension = filename.split('.').pop().toLowerCase();
            return extension === 'mp4';
        }

        function triggerFileInput() {
            fileInput.click();

            fileInput.onchange = () => {
                const file = fileInput.files[0];
                if (file) {
                    if (checkFileFormat(file.name)) {
                        uploadFile(file);
                    } else {
                        alert('Please select an MP4 file.');
                    }
                } else {
                    alert('Please select a file to upload.');
                }
            };
        }

        let currentTaskId = null;

        async function uploadFile(file) {
            resetProgress();
            progressBarContainer.style.display = 'block'; // Show the progress bar
    container

            const formData = new FormData();
            formData.append('file', file);
```

```
100
101             try {
102                 const response = await fetch('videos/upload_video_endpoint/free-throws
    ', {
103                     method: 'POST',
104                     body: formData
105                 });
106                 const data = await response.json();
107                 if (data.success) {
108                     console.log('Test function started successfully');
109                     currentTaskId = data.task_id; // Store the current task ID
110                     startWebSocket(data.task_id);
111                 } else {
112                     console.error('Test function start failed:', data.error);
113                 }
114             } catch (error) {
115                 console.error('Error starting test function:', error);
116             }
117         }
118
119         function resetProgress() {
120             progressBar.style.width = '0%';
121             progressText.innerText = '0%';
122         }
123
124         function startWebSocket(taskId) {
125             const socket = new WebSocket(`ws://localhost:8000/videos/ws`);
126
127             socket.onmessage = async function(event) {
128                 const data = JSON.parse(event.data);
129                 if (data.progress !== undefined) {
130                     progressBar.style.width = data.progress + '%';
131                     progressText.innerText = `${data.progress.toFixed(2)}%`;
132                 }
133
134                 if (data.message !== undefined) {
135                     socket.close(); // Close the WebSocket when the message is
    received
136                 }
137
138             };
139
140             socket.onclose = function(event) {
141                 console.log('WebSocket is closed now.');
142                 progressBarContainer.style.display = 'none'; // Hide the progress bar
    container
143                 fetchResults(taskId);
144             };
145
146             socket.onerror = function(error) {
147                 console.log('WebSocket Error: ' + error);
148                 progressBarContainer.style.display = 'none'; // Hide the progress bar
    container on error
149             };
150         }
151
152         async function fetchResults(taskId) {
153             try {
154                 const response = await fetch(`videos/get_video_processing_results/${
    taskId}`);
155                 const data = await response.json();
156
157                 let alertMessage;
158
159                 if (data.success) {
160                     const score = data.score;
161                     const misses = data.misses;
162                     const throws = data.throws;
163                     alertMessage = `The video was uploaded successfully with results:\
    nScore: ${score}\nMisses: ${misses}\nThrows: ${throws}\nRefresh the page to upload
    new one)`;
```

```
            } else {
                alertMessage = `An error occurred during video uploading`;
            }

            alert(alertMessage);
        } catch (error) {
            console.error('Error fetching results:', error);
            alert('An unexpected error occurred while fetching the results.');
        }
    }

    async function getStatistics() {
        try {
            const response = await fetch('/videos/statistics/free-throws');
            if (!response.ok) {
                throw new Error('Network response was not ok');
            }
            const data = await response.json();

            const table = document.getElementById('stats-table');
            const tbody = document.getElementById('stats-body');
            tbody.innerHTML = '';

            data.forEach(video => {
                const row = document.createElement('tr');
                row.innerHTML = `
                    <td class="video-link" data-video-path="${video.video_path}">
                        <a href="javascript:void(0);" onclick="openVideoInNewTab('${video.video_path}')">${video.name_video}</a>
                    </td>
                    <td>${video.number_throws}</td>
                    <td>${video.number_goals}</td>
                    <td>${video.number_misses}</td>
                    <td>${video.accuracy}%</td>
                    <td>${new Date(video.date_upload).toLocaleString()}</td>
                    <td>
                        <button onclick="deleteVideo('${video.video_path}', this)" class="delete-button">Delete</button>
                    </td>
                `;
                tbody.appendChild(row);
            });

            table.style.display = 'table';

        } catch (error) {
            console.error('Error:', error);
        }
    }

    async function deleteVideo(videoPath, button) {
        if (confirm('Are you sure you want to delete this video?')) {
            try {

                // Make a request to the server to delete the video
                const response = await fetch(`/videos/delete_video/${videoPath}`, {
                    method: 'DELETE'
                });

                const data = await response.json();

                if (data.success) {
                    // Remove the row from the table
                    const row = button.closest('tr');
                    row.remove();
                    alert('Video deleted successfully.');
                } else {
                    alert('Failed to delete video: ' + data.error);
                }
            } catch (error) {
```

```
                    console.error('Error:', error);
                    alert('An error occurred while deleting the video.');
                }
            }
        }

        function openVideoInNewTab(videoPath) {
            // Replace backslashes with forward slashes to ensure URL correctness
            // const fixedPath = videoPath.replace(/\\/g, '/');
            const fixedPath = videoPath.replace('result_videos', 'result_videos\\r');
            // const videoURL = `/${fixedPath}`;
            const videoURL = `/${fixedPath}`;

            // console.log(videoURL); // For debugging
            window.open(videoURL, '_blank');
        }

        window.addEventListener('beforeunload', async (event) => {
            if (currentTaskId) {
                try {
                    const response = await fetch(`videos/stop_video_processing/${currentTaskId}`, {
                        method: 'POST'
                    });
                    const data = await response.json();
                    if (data.success) {
                        console.log('Video processing stopped successfully');
                    } else {
                        console.error('Failed to stop video processing:', data.message);
                    }
                } catch (error) {
                    console.error('Error stopping video processing:', error);
                }
            }
        });
    </script>
    <script src="static/js/header.js"></script>
</body>
</html>
```

Listing 4: free_throws.html

homepage.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Basketball app</title>
    <link rel="stylesheet" href="static/css/main.css">
    <link rel="stylesheet" href="static/css/header.css">
    <link rel="stylesheet" href="static/css/homepage.css">
</head>

<body>
    <div class="header">
        <div class="left-buttons">
            <button id="home-button">Home Page</button>
            <button id="about-button">About</button>
            <button id="developers-button">Developers</button>
            <button id="blog-button">Blog</button>
        </div>
        <div class="right-buttons">
            <div class="username-frame" id="username-frame">
                <span id="username-display"></span>
            </div>
            <button id="login-button">Log In</button>
            <button id="register-button">Register</button>
            <button id="logout-button" style="display: none;">Log Out</button>
        </div>
```

```
    </div>
    <div class="images-container">
        <div class="image-container">
            <img src="/static/images/free_throws.png" class="basketball-image">
            <button id="free-throws-button">Free Throws</button>
        </div>
        <div class="image-container">
            <img src="/static/images/three_point_throws.png" class="basketball-image">
            <button id="three-point-throws-button">3-Point Throws</button>
        </div>
        <div class="image-container">
            <img src="/static/images/analyze.png" class="basketball-image">
            <button id="analyze-button">Analyze</button>
        </div>
    </div>
    <script src="static/js/header.js"></script>
    <script src="static/js/homepage.js"></script>
</body>
</html>
```

Listing 5: homepage.html

login.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Enter</title>
    <link rel="stylesheet" href="static/css/login.css">
</head>
<body>
    <form id="loginForm">
        <input type="text" id="username" placeholder="Email" required>
        <div class="password-container">
            <input type="password" id="password" placeholder="Password" required>
            <span class="toggle-password">         </span>
        </div>
        <button type="submit">Enter</button>
    </form>
    <script src="static/js/login.js"></script>
</body>
</html>
```

Listing 6: login.html

register.html

```
<!DOCTYPE html>
<html lang="eng">
<head>
    <meta charset="UTF-8">
    <title>Registration</title>
    <link rel="stylesheet" href="static/css/register.css">
</head>
<body>
    <form id="registerForm">
        <input type="text" id="username" placeholder="Username">
        <input type="email" id="email" placeholder="Email">
        <div class="password-container">
            <input type="password" id="password" placeholder="Password">
            <span class="toggle-password">         </span>
        </div>
        <button type="submit">Register</button>
    </form>
    <script src="static/js/register.js"></script>
</body>
</html>
```

Listing 7: register.html

three_point_throws.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Upload video</title>
    <link rel="stylesheet" href="static/css/main.css">
    <link rel="stylesheet" href="static/css/header.css">
    <link rel="stylesheet" href="static/css/throws_page.css">
</head>
<body>
    <div class="header">
        <div class="left-buttons">
            <button id="home-button">Home Page</button>
            <button id="about-button">About</button>
            <button id="developers-button">Developers</button>
            <button id="blog-button">Blog</button>
        </div>
        <div class="right-buttons">
            <div class="username-frame" id="username-frame">
                <span id="username-display"></span>
            </div>
            <button id="login-button">Log In</button>
            <button id="register-button">Register</button>
            <button id="logout-button" style="display: none;">Log Out</button>
        </div>
    </div>
    <input type="file" id="file-input" accept="video/mp4" style="display: none;">
    <div class="throws-text">
        3-Point Throws
    </div>
    <div class="upload-container">
        <div class="buttons-container">
            <button type="button" onclick="triggerFileInput()">Upload Video</button>
            <button onclick="getStatistics()">Get statistics</button>
        </div>

        <div id="progressBarContainer">
            <div id="progressBar">
                <div id="progressTextContainer">
                    <span id="progressText">0%</span>
                </div>
            </div>
        </div>

        <div id="stats-container" class="stats-container">
            <table id="stats-table" style="display: none;">
                <thead>
                    <tr>
                        <th>Video name</th>
                        <th>Number of throws</th>
                        <th>Number of goals</th>
                        <th>Number of misses</th>
                        <th>Accuracy</th>
                        <th>Upload date</th>
                    </tr>
                </thead>
                <tbody id="stats-body">
                </tbody>
            </table>
        </div>
    </div>

    <script>
        const progressBarContainer = document.getElementById('progressBarContainer');
        const progressBar = document.getElementById('progressBar');
        const progressText = document.getElementById('progressText');
        const fileInput = document.getElementById('file-input');

        function checkFileFormat(filename) {
            const extension = filename.split('.').pop().toLowerCase();
```

```
            return extension === 'mp4';
        }

        function triggerFileInput() {
            fileInput.click();

            fileInput.onchange = () => {
                const file = fileInput.files[0];
                if (file) {
                    if (checkFileFormat(file.name)) {
                        uploadFile(file);
                    } else {
                        alert('Please select an MP4 file.');
                    }
                } else {
                    alert('Please select a file to upload.');
                }
            };
        }

        let currentTaskId = null;

        async function uploadFile(file) {
            resetProgress();
            progressBarContainer.style.display = 'block'; // Show the progress bar container

            const formData = new FormData();
            formData.append('file', file);

            try {
                const response = await fetch('videos/upload_video_endpoint/3-point-throws', {
                    method: 'POST',
                    body: formData
                });
                const data = await response.json();
                if (data.success) {
                    console.log('Test function started successfully');
                    currentTaskId = data.task_id; // Store the current task ID
                    startWebSocket(data.task_id);
                } else {
                    console.error('Test function start failed:', data.error);
                }
            } catch (error) {
                console.error('Error starting test function:', error);
            }
        }

        function resetProgress() {
            progressBar.style.width = '0%';
            progressText.innerText = '0%';
        }

        function startWebSocket(taskId) {
            const socket = new WebSocket(`ws://localhost:8000/videos/ws`);

            socket.onmessage = async function(event) {
                const data = JSON.parse(event.data);
                if (data.progress !== undefined) {
                    progressBar.style.width = data.progress + '%';
                    progressText.innerText = `${data.progress.toFixed(2)}%`;
                }

                if (data.message !== undefined) {
                    socket.close(); // Close the WebSocket when the message is received
                }

            };

```

```
            socket.onclose = function(event) {
                console.log('WebSocket is closed now.');
                progressBarContainer.style.display = 'none'; // Hide the progress bar container
                fetchResults(taskId);
            };

            socket.onerror = function(error) {
                console.log('WebSocket Error: ' + error);
                progressBarContainer.style.display = 'none'; // Hide the progress bar container on error
            };
        }

        async function fetchResults(taskId) {
            try {
                const response = await fetch(`videos/get_video_processing_results/${taskId}`);
                const data = await response.json();

                let alertMessage;

                if (data.success) {
                    const score = data.score;
                    const misses = data.misses;
                    const throws = data.throws;
                    alertMessage = `The video was uploaded successfully with results:\nScore: ${score}\nMisses: ${misses}\nThrows: ${throws}\nRefresh the page to upload new one)`;
                } else {
                    alertMessage = `An error occurred during video uploading`;
                }

                alert(alertMessage);
            } catch (error) {
                console.error('Error fetching results:', error);
                alert('An unexpected error occurred while fetching the results.');
            }
        }

        async function getStatistics() {
            try {
                const response = await fetch('/videos/statistics/3-point-throws');
                if (!response.ok) {
                    throw new Error('Network response was not ok');
                }
                const data = await response.json();

                const table = document.getElementById('stats-table');
                const tbody = document.getElementById('stats-body');
                tbody.innerHTML = '';

                data.forEach(video => {
                    const row = document.createElement('tr');
                    row.innerHTML = `
                        <td class="video-link" data-video-path="${video.video_path}">
                            <a href="javascript:void(0);" onclick="openVideoInNewTab('${video.video_path}')">${video.name_video}</a>
                        </td>
                        <td>${video.number_throws}</td>
                        <td>${video.number_goals}</td>
                        <td>${video.number_misses}</td>
                        <td>${video.accuracy}%</td>
                        <td>${new Date(video.date_upload).toLocaleString()}</td>
                        <td>
                            <button onclick="deleteVideo('${video.video_path}', this)" class="delete-button">Delete</button>
                        </td>
                    `;
                    tbody.appendChild(row);
                });
```

```
                table.style.display = 'table';

            } catch (error) {
                console.error('Error:', error);
            }
        }

        async function deleteVideo(videoPath, button) {
            if (confirm('Are you sure you want to delete this video?')) {
                try {

                    // Make a request to the server to delete the video
                    const response = await fetch(`/videos/delete_video/${videoPath}`, {
                        method: 'DELETE'
                    });

                    const data = await response.json();

                    if (data.success) {
                        // Remove the row from the table
                        const row = button.closest('tr');
                        row.remove();
                        alert('Video deleted successfully.');
                    } else {
                        alert('Failed to delete video: ' + data.error);
                    }
                } catch (error) {
                    console.error('Error:', error);
                    alert('An error occurred while deleting the video.');
                }
            }
        }

        function openVideoInNewTab(videoPath) {
            // Replace backslashes with forward slashes to ensure URL correctness
            // const fixedPath = videoPath.replace(/\\/g, '/');
            const fixedPath = videoPath.replace('result_videos', 'result_videos\\r');
            // const videoURL = `/${fixedPath}`;
            const videoURL = `/${fixedPath}`;

            // console.log(videoURL); // For debugging
            window.open(videoURL, '_blank');
        }

        window.addEventListener('beforeunload', async (event) => {
            if (currentTaskId) {
                try {
                    const response = await fetch(`videos/stop_video_processing/${currentTaskId}`, {
                        method: 'POST'
                    });
                    const data = await response.json();
                    if (data.success) {
                        console.log('Video processing stopped successfully');
                    } else {
                        console.error('Failed to stop video processing:', data.message);
                    }
                } catch (error) {
                    console.error('Error stopping video processing:', error);
                }
            }
        });
    </script>
    <script src="static/js/header.js"></script>
</body>
</html>
```

Listing 8: three_point_throws.html

11.4.2 web_app/

app.py

```python
from fastapi import FastAPI, Depends, Request
from src.process_video.router_videos import router as router_videos
from src.auth.base_config import auth_backend
from src.auth.models import User
from src.auth.schemas import UserRead, UserCreate
from src.auth.base_config import current_user, fastapi_users
from fastapi.responses import HTMLResponse
from starlette.middleware.wsgi import WSGIMiddleware
from src.dashboard.dashboard import get_dash_app
from fastapi.staticfiles import StaticFiles
from fastapi.responses import RedirectResponse


app = FastAPI(
    title="Basketball App"
)

# Serve static files
app.mount("/static", StaticFiles(directory="static"), name="static")

# Serve the static files from the '/result_videos' directory
app.mount("/result_videos", StaticFiles(directory="result_videos"), name="
    result_videos")


@app.get("/dashboard-redirect")
async def protected_route(request: Request, user: User = Depends(current_user)):
    cookie_value = request.cookies.get("bonds")
    response = RedirectResponse(url="/dashboard")
    response.set_cookie(key="bonds", value=cookie_value, httponly=True)
    return response

@app.get("/free-throws", response_class=HTMLResponse)
async def get_free_throws_page():
    with open('public_html/free_throws.html', 'r', encoding='utf-8') as f:
        register_page = f.read()
    return register_page

@app.get("/three-point-throws", response_class=HTMLResponse)
async def get_three_point_throws_page():
    with open('public_html/three_point_throws.html', 'r', encoding='utf-8') as f:
        register_page = f.read()
    return register_page

@app.get("/register", response_class=HTMLResponse)
async def get_register_page():
    with open('public_html/register.html', 'r', encoding='utf-8') as f:
        register_page = f.read()
    return register_page

@app.get("/login", response_class=HTMLResponse)
async def get_login_page():
    with open('public_html/login.html', 'r', encoding='utf-8') as f:
        login_page = f.read()
    return login_page

@app.get("/homepage", response_class=HTMLResponse)
async def get_home_page():
    with open('public_html/homepage.html', 'r', encoding='utf-8') as f:
        video_main = f.read()
    return video_main

@app.get("/about", response_class=HTMLResponse)
async def get_home_page():
    with open('public_html/about.html', 'r', encoding='utf-8') as f:
        video_main = f.read()
    return video_main

```

```python
@app.get("/developers", response_class=HTMLResponse)
async def get_home_page():
    with open('public_html/developers.html', 'r', encoding='utf-8') as f:
        video_main = f.read()
    return video_main

@app.get("/blog", response_class=HTMLResponse)
async def get_home_page():
    with open('public_html/blog.html', 'r', encoding='utf-8') as f:
        video_main = f.read()
    return video_main


app.include_router(
    fastapi_users.get_auth_router(auth_backend),
    prefix="/auth/jwt",
    tags=["auth"],
)

app.include_router(
    fastapi_users.get_reset_password_router(),
    prefix="/auth",
    tags=["auth"],
)

app.include_router(
    fastapi_users.get_register_router(UserRead, UserCreate),
    prefix="/auth",
    tags=["auth"],
)


@app.get("/protected-route")
def protected_route(user: User = Depends(current_user)):
    return user.username

app.include_router(router_videos)

# Create the Dash app
dash_app = get_dash_app()

# Mount Dash app to the FastAPI app
app.mount("/dashboard", WSGIMiddleware(dash_app.server))
```

Listing 9: app.py

11.4.3 web_app/src/

config.py

```python
from dotenv import load_dotenv
import os

load_dotenv()

DB_HOST = os.environ.get("DB_HOST")
DB_PORT = os.environ.get("DB_PORT")
DB_NAME = os.environ.get("DB_NAME")
DB_USER = os.environ.get("DB_USER")
DB_PASS = os.environ.get("DB_PASS")

SECRET_AUTH = os.environ.get("SECRET_AUTH")
```

Listing 10: config.py

database.py

```python
from typing import AsyncGenerator

from sqlalchemy import MetaData
from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine
```

```python
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

from src.config import DB_HOST, DB_NAME, DB_PASS, DB_PORT, DB_USER

DATABASE_URL = f"postgresql+asyncpg://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
Base = declarative_base()


engine = create_async_engine(DATABASE_URL)
async_session_maker = sessionmaker(engine, class_=AsyncSession, expire_on_commit=False)


async def get_async_session() -> AsyncGenerator[AsyncSession, None]:
    async with async_session_maker() as session:
        yield session
```

Listing 11: database.py

11.4.4 web_app/src/auth

base_config.py

```python
from fastapi_users import FastAPIUsers
from fastapi_users.authentication import CookieTransport, AuthenticationBackend
from fastapi_users.authentication import JWTStrategy

from src.auth.manager import get_user_manager
from src.auth.models import User
from src.config import SECRET_AUTH

# Default cookie transport
cookie_transport = CookieTransport(cookie_name="basketball", cookie_max_age=3600)

# Extended cookie transport for processing
extended_cookie_transport = CookieTransport(cookie_name="basketball", cookie_max_age=7200)

def get_jwt_strategy(processing: bool = False) -> JWTStrategy:
    lifetime = 3600
    if processing:
        lifetime = 7200
    return JWTStrategy(secret=SECRET_AUTH, lifetime_seconds=lifetime)

# auth_backend = AuthenticationBackend(
#     name="jwt",
#     transport=cookie_transport,
#     get_strategy=get_jwt_strategy,
# )

# Authentication backend
auth_backend = AuthenticationBackend(
    name="jwt",
    transport=cookie_transport,
    get_strategy=get_jwt_strategy,
)

# Extended authentication backend for processing
extended_auth_backend = AuthenticationBackend(
    name="jwt_extended",
    transport=extended_cookie_transport,
    get_strategy=lambda: get_jwt_strategy(processing=True),
)

fastapi_users = FastAPIUsers[User, int](
    get_user_manager,
    [auth_backend],
)

```

```python
current_user = fastapi_users.current_user(active=True)
```

Listing 12: base_config.py

manager.py

```python
from typing import Optional

from fastapi import Depends, Request
from fastapi_users import BaseUserManager, IntegerIDMixin, exceptions, models, schemas

from src.auth.models import User
from src.auth.utils import get_user_db

from src.config import SECRET_AUTH


class UserManager(IntegerIDMixin, BaseUserManager[User, int]):
    reset_password_token_secret = SECRET_AUTH
    verification_token_secret = SECRET_AUTH

    async def on_after_register(self, user: User, request: Optional[Request] = None):
        print(f"User {user.id} has registered.")

    async def on_after_forgot_password(
            self, user: User, token: str, request: Optional[Request] = None
    ):
        print(f"User {user.id} has forgot their password. Reset token: {token}")


    async def create(
        self,
        user_create: schemas.UC,
        safe: bool = False,
        request: Optional[Request] = None,
    ) -> models.UP:
        await self.validate_password(user_create.password, user_create)

        existing_user = await self.user_db.get_by_email(user_create.email)

        if existing_user is not None:
            raise exceptions.UserAlreadyExists()

        user_dict = (
            user_create.create_update_dict()
            if safe
            else user_create.create_update_dict_superuser()
        )
        password = user_dict.pop("password")
        user_dict["hashed_password"] = self.password_helper.hash(password)
        user_dict["role_id"] = 1

        created_user = await self.user_db.create(user_dict)

        await self.on_after_register(created_user, request)

        return created_user


async def get_user_manager(user_db=Depends(get_user_db)):
    yield UserManager(user_db)
```

Listing 13: manager.py

models.py

```python
from datetime import datetime

from fastapi_users_db_sqlalchemy import SQLAlchemyBaseUserTable
from sqlalchemy import Table, Column, Integer, String, TIMESTAMP, ForeignKey, JSON,
    Boolean, MetaData

from src.database import Base
```

```python

metadata = MetaData()

role = Table(
    "role",
    metadata,
    Column("id", Integer, primary_key=True),
    Column("name", String, nullable=False),
    Column("permissions", JSON),
)

user = Table(
    "user",
    metadata,
    Column("id", Integer, primary_key=True),
    Column("email", String, nullable=False, unique=True),
    Column("username", String, nullable=False, unique=True),
    Column("registered_at", TIMESTAMP, default=datetime.utcnow),
    Column("role_id", Integer, ForeignKey(role.c.id)),
    Column("hashed_password", String, nullable=False),
    Column("is_active", Boolean, default=True, nullable=False),
    Column("is_superuser", Boolean, default=False, nullable=False),
    Column("is_verified", Boolean, default=False, nullable=False),
)

class User(SQLAlchemyBaseUserTable[int], Base):
    id = Column(Integer, primary_key=True)
    email = Column(String, nullable=False)
    username = Column(String, nullable=False)
    registered_at = Column(TIMESTAMP, default=datetime.utcnow)
    role_id = Column(Integer, ForeignKey(role.c.id))
    hashed_password: str = Column(String(length=1024), nullable=False)
    is_active: bool = Column(Boolean, default=True, nullable=False)
    is_superuser: bool = Column(Boolean, default=False, nullable=False)
    is_verified: bool = Column(Boolean, default=False, nullable=False)
```

Listing 14: manager.py

schemas.py

```python
from pydantic import EmailStr
from fastapi_users import schemas

class UserRead(schemas.BaseUser[int]):
    id: int
    email: str
    username: str
    role_id: int
    is_active: bool = True
    is_superuser: bool = False
    is_verified: bool = False

    class Config:
        orm_mode = True


class UserCreate(schemas.CreateUpdateDictModel):
    username: str
    email: EmailStr
    password: str
```

Listing 15: schemas.py

utils.py

```python
from fastapi import Depends
from fastapi_users_db_sqlalchemy import SQLAlchemyUserDatabase
from sqlalchemy.ext.asyncio import AsyncSession

from src.auth.models import User
from src.database import get_async_session


```

```
async def get_user_db(session: AsyncSession = Depends(get_async_session)):
    yield SQLAlchemyUserDatabase(session, User)
```

Listing 16: schemas.py

11.4.5 web_app/src/dashboard

dashboard.py

```
from dash import Dash, dcc, html, Input, Output
import dash_bootstrap_components as dbc
import requests
import plotly.express as px
import pandas as pd
from flask import request

external_stylesheets = [dbc.themes.BOOTSTRAP, '/static/css/dashboard.css']

# Create Dash application
dash_app = Dash(__name__, requests_pathname_prefix='/dashboard/', external_stylesheets
    =external_stylesheets)

# Dash layout with separate containers for header, controls, and charts
dash_app.layout = html.Div([
    html.Div([
        html.A(
            dbc.Button("Back to homepage", color="success", className="btn-homepage"),
            href="http://127.0.0.1:8000/homepage",
            style={'margin-right': '20px'}
        ),
        html.H1("Score performance visualization", className="dashboard-title")
    ], className="header-container dashboard-container"),

    html.Div([
        html.Div([
            html.Label('Choose the type of the throw:', className="label"),
            dcc.RadioItems(
                id='video-type-1',
                options=[
                    {'label': 'Free Throws', 'value': 'free-throws'},
                    {'label': '3-Point Throws', 'value': '3-point-throws'},
                    {'label': 'All', 'value': 'all'}
                ],
                value='free-throws',  # Default value
                labelStyle={'display': 'inline-block', 'margin-right': '20px'},
            ),
        ], className="input-container"),

        # html.Div(id='total-accuracy', className="total-accuracy"),

        # Toggle for choosing the filtering method
        html.Div([
            html.Label('Choose the filtering method:', className="label"),
            dcc.RadioItems(
                id='filter-method',
                options=[
                    {'label': 'Date Range', 'value': 'date-range'},
                    {'label': 'Recent Videos', 'value': 'video-slider'}
                ],
                value='date-range',  # Default value
                labelStyle={'display': 'inline-block', 'margin-right': '20px'},
            ),
        ], className="input-container"),

        html.Div([
            html.Div(id='date-label', className="label"),
            dcc.DatePickerRange(
                id='date-range',
                display_format='YYYY-MM-DD',
```

```
                style={'margin-bottom': '0'}
            ),
        ], className="input-container", id='date-range-container'),

        # Video slider
        html.Div([
            html.Label('Choose the number of recent videos:', className="label"),
            dcc.Slider(
                id='video-slider',
                min=1,
                max=1,
                value=1,
                marks=None,
                step=1
            ),
        ], className="input-container", id='video-slider-container'),
    ], className="controls-container dashboard-container"),

    dcc.Interval(
        id='interval-component',
        interval=30*1000,  # half minute in milliseconds
        n_intervals=0
    ),

    html.Hr(className="divider"),

    html.Div([
        dcc.Graph(id='bar-chart')
    ], className="graph-container dashboard-container"),

    html.Hr(className="divider"),

    html.Div([
        dcc.Graph(id='line-chart')
    ], className="graph-container dashboard-container"),

], className="main-container")

# Callback to toggle between date picker and slider
@dash_app.callback(
    Output('date-range-container', 'style'),
    Output('video-slider-container', 'style'),
    Input('filter-method', 'value')
)
def toggle_filtering_method(filter_method):
    if filter_method == 'date-range':
        return {'display': 'block'}, {'display': 'none'}
    elif filter_method == 'video-slider':
        return {'display': 'none'}, {'display': 'block'}
    return {'display': 'block'}, {'display': 'none'}


# Callback for updating date-label
@dash_app.callback(
    Output('date-label', 'children'),
    Input('interval-component', 'n_intervals'),
)
def update_date_label(n_intervals):
    return f"Choose the timeline:"


# Callback to update date range
@dash_app.callback(
    Output('date-range', 'start_date'),
    Output('date-range', 'end_date'),
    Input('video-type-1', 'value')
)
def update_date_range(video_type):

    # Get cookie value from request
    cookie_value = request.cookies.get('basketball')
```

```
133
134     headers = {
135         'Cookie': f'basketball={cookie_value}'
136     }
137
138     response = requests.get(f"http://127.0.0.1:8000/videos/statistics/{video_type}",
    headers=headers)
139     data = response.json()
140
141     # Create DataFrame
142     df = pd.DataFrame(data)
143
144     if df.empty:
145         return None, None
146
147     # Convert 'date_upload' to datetime
148     df['date_upload'] = pd.to_datetime(df['date_upload'])
149
150     # Extract the date part
151     df['date'] = df['date_upload'].dt.date
152
153     # Get min and max dates
154     start_date = df['date'].min()
155     end_date = df['date'].max()
156
157     return start_date, end_date
158
159 # Callback to update video slider
160 @dash_app.callback(
161     Output('video-slider', 'min'),
162     Output('video-slider', 'max'),
163     Output('video-slider', 'value'),
164     Output('video-slider', 'marks'),
165     Input('video-type-1', 'value')
166 )
167 def update_video_slider(video_type):
168     # Get cookie value from request
169     cookie_value = request.cookies.get('basketball')
170
171     headers = {
172         'Cookie': f'basketball={cookie_value}'
173     }
174
175     response = requests.get(f"http://127.0.0.1:8000/videos/statistics/{video_type}",
    headers=headers)
176     data = response.json()
177
178     # Create DataFrame
179     df = pd.DataFrame(data)
180
181     if df.empty:
182         # Return default empty slider if the dataframe is empty
183         return 1, 1, 1, None
184
185     # Get number of videos
186     video_num = df.shape[0]
187
188     # Define the marks for the slider dynamically
189     marks = {i: str(i) for i in range(1, video_num + 1)}
190
191     # Return the slider min, max, value, and marks
192     return 1, video_num, video_num, marks
193
194
195 # Callbacks to update graphs based on selected filter method
196 @dash_app.callback(
197     Output('bar-chart', 'figure'),
198     Output('line-chart', 'figure'),
199     Input('interval-component', 'n_intervals'),
200     Input('video-type-1', 'value'),
201     Input('filter-method', 'value'),
```

```
    Input('date-range', 'start_date'),
    Input('date-range', 'end_date'),
    Input('video-slider', 'value')
)
def update_charts(n_intervals, video_type, filter_method, start_date, end_date,
    slider_value):
    # Get cookie value from request
    cookie_value = request.cookies.get('basketball')

    headers = {
        'Cookie': f'basketball={cookie_value}'
    }

    response = requests.get(f"http://127.0.0.1:8000/videos/statistics/{video_type}",
    headers=headers)
    data = response.json()

    # Create DataFrame
    df = pd.DataFrame(data)

    if df.empty:
        return {
            'data': [],
            'layout': {'title': 'No data available'}
        }, {
            'data': [],
            'layout': {'title': 'No data available'}
        }

    df = df.sort_values(by=['id'], ascending=True)
    if filter_method == 'date-range':
        # Convert 'date_upload' to datetime
        df['date_upload'] = pd.to_datetime(df['date_upload'])
        df['date'] = df['date_upload'].dt.date

        start_date = pd.to_datetime(start_date).date()
        end_date = pd.to_datetime(end_date).date()

        mask = (df['date'] >= start_date) & (df['date'] <= end_date)
        filtered_df = df.loc[mask]

    else:
        filtered_df = df.iloc[-slider_value:]

    # If the filtered DataFrame is empty, return empty graphs
    if filtered_df.empty:
        return {
            'data': [],
            'layout': {'title': 'No data available'}
        }, {
            'data': [],
            'layout': {'title': 'No data available'}
        }

    # Bar Chart
    sum_goals = sum(filtered_df['number_goals'])
    sum_misses = sum(filtered_df['number_misses'])
    total = sum_goals + sum_misses

    percentage_goals = (sum_goals / total) * 100 if total > 0 else 0
    percentage_misses = (sum_misses / total) * 100 if total > 0 else 0

    bar_fig = {
        'data': [
            {'x': ['Goals'], 'y': [sum_goals], 'type': 'bar', 'name': 'Goals', 'text':
     [f'<b>{percentage_goals:.2f}%</b>'], 'textposition': 'auto', 'texttemplate': '%{
    text}'},
            {'x': ['Misses'], 'y': [sum_misses], 'type': 'bar', 'name': 'Misses', '
    text': [f'<b>{percentage_misses:.2f}%</b>'], 'textposition': 'auto', 'texttemplate'
    : '%{text}'}
        ],
```

```
        'layout': {'title': f'Total Goals and Misses'}
    }

    filtered_df.reset_index(inplace=True)

    # Line Chart
    if filter_method == 'date-range':
        pivot_table = filtered_df.pivot_table(index='date', values='accuracy', aggfunc='mean').reset_index()
        line_fig = px.line(pivot_table, x='date', y='accuracy', title='Accuracy Over Time', markers=True)
        line_fig.update_layout(title={'x': 0.5})
    else:
        filtered_df = filtered_df.sort_values(by=['id'], ascending=True)
        filtered_df.reset_index(inplace=True)
        filtered_df['date_upload'] = pd.to_datetime(filtered_df['date_upload'])
        filtered_df['date'] = filtered_df['date_upload'].dt.date
        # print(filtered_df['date_upload'].dt.time)
        filtered_df['time'] = filtered_df['date_upload'].apply(lambda dt: dt.strftime("%H:%M:%S") + " (UTC)")


        line_fig = px.line(
            filtered_df,
            x=filtered_df.index+1,
            y='accuracy',
            title='Accuracy Over Time',
            markers=True,
            hover_data={
                'name_video': True,
                'accuracy': True,
                'date': True,
                'time': True
            },
            labels={
                'name_video': 'Video Name',
                'accuracy': 'Accuracy',
                'date': 'Date',
                'time': 'Time'
            })

        line_fig.update_layout(title={'x': 0.5})

        # Update hovertemplate to exclude x value
        line_fig.update_traces(
            hovertemplate='<br>'.join([
                'Accuracy=%{y}',
                'Video Name=%{customdata[0]}',
                'Date=%{customdata[1]}',
                'Time=%{customdata[2]}'
            ]))

        line_fig.update_xaxes(
            tickmode='array',
            tickvals=list(range(1, len(filtered_df) + 1)),  # Only integer tick values
            title_text='Video Number')

        line_fig.update_yaxes(range=[-5, 105])

    return bar_fig, line_fig

# Function to return the Dash app
def get_dash_app():
    return dash_app
```

Listing 17: dashboard.py

11.4.6 web_app/src/process_video

classes.py

```python
import numpy as np
import numpy.typing as npt
from typing import Iterable, Optional, Tuple
import cv2
from supervision import Detections
from supervision.detection.utils import clip_boxes, polygon_to_mask
from supervision.draw.color import Color
from supervision.draw.utils import draw_polygon, draw_text
from supervision.geometry.core import Position
from supervision.geometry.utils import get_polygon_center
from supervision.utils.internal import deprecated_parameter
import supervision as sv

def point_in_rect(px: int, py: int, rx1: int, ry1: int, rx2: int, ry2: int) -> bool:
    """
    Check if a point (px, py) is inside a rectangle defined by its top-left (rx1, ry1)
    and bottom-right (rx2, ry2) coordinates.

    Parameters:
    px (float): x-coordinate of the point.
    py (float): y-coordinate of the point.
    rx1 (float): x-coordinate of the rectangle's top-left corner.
    ry1 (float): y-coordinate of the rectangle's top-left corner.
    rx2 (float): x-coordinate of the rectangle's bottom-right corner.
    ry2 (float): y-coordinate of the rectangle's bottom-right corner.

    Returns:
    bool: True if the point is inside the rectangle, False otherwise.
    """
    return rx1 <= px <= rx2 and ry1 <= py <= ry2

def rectangles_intersect(r1_p1, r1_p2,
                         r2_p1, r2_p2) -> bool:
    """
    Check if two rectangles intersect. Each rectangle is defined by two points:
    the top-left and the bottom-right corners.

    Parameters:
    r1_p1 (tuple[float, float]): Top-left corner of the first rectangle (x1, y1).
    r1_p2 (tuple[float, float]): Bottom-right corner of the first rectangle (x2, y2).
    r2_p1 (tuple[float, float]): Top-left corner of the second rectangle (x1, y1).
    r2_p2 (tuple[float, float]): Bottom-right corner of the second rectangle (x2, y2).

    Returns:
    bool: True if the rectangles intersect, False otherwise.
    """
    # Unpack rectangle points
    r1_x1, r1_y1 = r1_p1
    r1_x2, r1_y2 = r1_p2
    r2_x1, r2_y1 = r2_p1
    r2_x2, r2_y2 = r2_p2

    # Define the vertices for both rectangles
    r1_vertices = [(r1_x1, r1_y1), (r1_x2, r1_y1), (r1_x2, r1_y2), (r1_x1, r1_y2)]
    r2_vertices = [(r2_x1, r2_y1), (r2_x2, r2_y1), (r2_x2, r2_y2), (r2_x1, r2_y2)]

    # Check if any vertex of rectangle 1 is inside rectangle 2 or vice versa
    return any(point_in_rect(px, py, r2_x1, r2_y1, r2_x2, r2_y2) for px, py in
    r1_vertices) or \
           any(point_in_rect(px, py, r1_x1, r1_y1, r1_x2, r1_y2) for px, py in
    r2_vertices)

# define annotation classes
bounding_box_annotator = sv.BoundingBoxAnnotator(thickness=2)
label_annotator = sv.LabelAnnotator(text_thickness=1, text_scale=0.4)

class PolygonZone:
    """
    A class for defining a polygon-shaped zone within a frame for detecting objects.

    Attributes:
```

```
        polygon (np.ndarray): A polygon represented by a numpy array of shape
            `(N, 2)`, containing the `x`, `y` coordinates of the points.
        frame_resolution_wh (Tuple[int, int]): The frame resolution (width, height)
        triggering_anchors (Iterable[sv.Position]): A list of positions specifying
            which anchors of the detections bounding box to consider when deciding on
            whether the detection fits within the PolygonZone
            (default: (sv.Position.BOTTOM_CENTER,)).
        current_count (int): The curren|t count of detected objects within the zone
        mask (np.ndarray): The 2D bool mask for the polygon zone
    """

    @deprecated_parameter(
        old_parameter="triggering_position",
        new_parameter="triggering_anchors",
        map_function=lambda x: [x],
        warning_message="`{old_parameter}` in `{function_name}` is deprecated and will "
        "be remove in `supervision-0.23.0`. Use '{new_parameter}' "
        "instead.",
    )
    def __init__(
        self,
        polygon: npt.NDArray[np.int64],
        frame_resolution_wh: Tuple[int, int],
        triggering_anchors: Iterable[Position] = (Position.BOTTOM_CENTER,),

    ):
        self.polygon = polygon.astype(int)
        self.frame_resolution_wh = frame_resolution_wh
        self.triggering_anchors = triggering_anchors

        self.current_count = 0

        width, height = frame_resolution_wh
        self.mask = polygon_to_mask(
            polygon=polygon, resolution_wh=(width + 1, height + 1)
        )

    def trigger(self, detections: Detections, class_id):
        """
        Determines if the spicific objects are within the polygon zone.

        Return:
        False, if none of the ojects of the specified class are within the polygon zone.
        True, if at least one of the ojects of the specified class are within the polygon zone.
        """

        triggers = []

        for obj in iter(detections):
            if obj[3] == class_id: # check for ball
                obj_x1, obj_y1, obj_x2, obj_y2 = obj[0]
                obj_p1 = (obj_x1, obj_y1)
                obj_p2 = (obj_x2, obj_y2)

                zone_p1 = self.polygon[0]
                zone_p2 = self.polygon[2]

                triggers.append(rectangles_intersect(obj_p1, obj_p2, zone_p1, zone_p2))

        return any(triggers)


class PolygonZoneAnnotator:
    """
    A class for annotating a polygon-shaped zone within a
        frame with a count of detected objects.

```

```
    Attributes:
        zone (PolygonZone): The polygon zone to be annotated
        color (Color): The color to draw the polygon lines
        thickness (int): The thickness of the polygon lines, default is 2
        text_color (Color): The color of the text on the polygon, default is black
        text_scale (float): The scale of the text on the polygon, default is 0.5
        text_thickness (int): The thickness of the text on the polygon, default is 1
        text_padding (int): The padding around the text on the polygon, default is 10
        font (int): The font type for the text on the polygon,
            default is cv2.FONT_HERSHEY_SIMPLEX
        center (Tuple[int, int]): The center of the polygon for text placement
        display_in_zone_count (bool): Show the label of the zone or not. Default is
    True
    """

    def __init__(
        self,
        zone: PolygonZone,
        color: Color,
        thickness: int = 2,
        text_color: Color = Color.BLACK,
        text_scale: float = 0.5,
        text_thickness: int = 1,
        text_padding: int = 10,
        display_in_zone_count: bool = True,
    ):
        self.zone = zone
        self.color = color
        self.thickness = thickness
        self.text_color = text_color
        self.text_scale = text_scale
        self.text_thickness = text_thickness
        self.text_padding = text_padding
        self.font = cv2.FONT_HERSHEY_SIMPLEX
        self.center = get_polygon_center(polygon=zone.polygon)
        self.display_in_zone_count = display_in_zone_count

    def annotate(self, scene: np.ndarray, label: Optional[str] = None) -> np.ndarray:
        """
        Annotates the polygon zone within a frame with a count of detected objects.

        Parameters:
            scene (np.ndarray): The image on which the polygon zone will be annotated
            label (Optional[str]): An optional label for the count of detected objects
                within the polygon zone (default: None)

        Returns:
            np.ndarray: The image with the polygon zone and count of detected objects
        """
        annotated_frame = draw_polygon(
            scene=scene,
            polygon=self.zone.polygon,
            color=self.color,
            thickness=self.thickness,
        )

        if self.display_in_zone_count:
            annotated_frame = draw_text(
                scene=annotated_frame,
                text=str(self.zone.current_count) if label is None else label,
                text_anchor=self.center,
                background_color=self.color,
                text_color=self.text_color,
                text_scale=self.text_scale,
                text_thickness=self.text_thickness,
                text_padding=self.text_padding,
                text_font=self.font,
            )

```

```
        return annotated_frame
```

Listing 18: classes.py

connection_manager.py

```
from fastapi import WebSocket
from typing import List

class ConnectionManager:
    def __init__(self):
        self.active_connections: List[WebSocket] = []

    async def connect(self, websocket: WebSocket):
        await websocket.accept()
        self.active_connections.append(websocket)

    def disconnect(self, websocket: WebSocket):
        self.active_connections.remove(websocket)

    async def send_message(self, message: str):
        for connection in self.active_connections:
            await connection.send_text(message)

    async def send_progress(self, progress: float):
        for connection in self.active_connections:
            await connection.send_json({"progress": progress})

    async def send_completion_message(self):
        for connection in self.active_connections:
            await connection.send_json({"message": "Video uploaded successfully"})
        for connection in self.active_connections:
            await connection.close()
```

Listing 19: connection_manager.py

functions.py

```
import numpy as np
import supervision as sv
import os
from ultralytics import YOLO
from supervision.draw.color import Color
from supervision.draw.utils import draw_text
from src.process_video.classes import PolygonZone, PolygonZoneAnnotator,
    bounding_box_annotator, label_annotator, rectangles_intersect
# from classes import PolygonZone, PolygonZoneAnnotator, bounding_box_annotator,
    label_annotator, rectangles_intersect
import tqdm


def detect(frame: np.ndarray, model) -> sv.Detections:
    result = model.predict(frame, imgsz=1280, conf=0.5, verbose=False)[0]
    return sv.Detections.from_ultralytics(result)


def setup_zone_above(detections, frame_wh):
    zones = []
    count_zone = 0
    for i in range(len(detections)):
        obj = detections[i]

        # choosing only baskets with confidence score >= 0.5
        if (obj.class_id[0] == 1) & (obj.confidence[0] >= 0.5):
            count_zone += 1

            obj_xyxy = obj.xyxy[0]

            # get coordinates of 4 points of bounding box
            obj_p1 = np.array([obj_xyxy[0], obj_xyxy[1]])
            obj_p2 = np.array([obj_xyxy[0], obj_xyxy[3]])
            obj_p3 = np.array([obj_xyxy[2], obj_xyxy[3]])
            obj_p4 = np.array([obj_xyxy[2], obj_xyxy[1]])
```

```
34
35              # get height and width of bounding box
36              obj_height = obj_xyxy[3] - obj_xyxy[1]
37              obj_width = obj_xyxy[2] - obj_xyxy[0]
38
39              # polygon construction strategy
40              # 1) zone width: the width of the    basket    bounding box increased on
    10% (5% from both
41              # left and right sides).
42              # 2) zone height: 60% of the height of the "basket" bounding box.
43              # 3) zone location: higher on 10% of the "basket" bounding box height from
     the upper bound
44              # of "basket" bounding box.
45
46              zone_p1 = [obj_p1[0] - 0.05 * obj_width, obj_p1[1] - 0.7 * obj_height]
47              zone_p2 = [obj_p1[0] - 0.05 * obj_width, obj_p1[1] - 0.1 * obj_height]
48              zone_p3 = [obj_p4[0] + 0.05 * obj_width, obj_p4[1] - 0.1 * obj_height]
49              zone_p4 = [obj_p4[0] + 0.05 * obj_width, obj_p4[1] - 0.7 * obj_height]
50
51              # round coordinates of each point of zone
52              zone_p1 = list(map(round, zone_p1))
53              zone_p2 = list(map(round, zone_p2))
54              zone_p3 = list(map(round, zone_p3))
55              zone_p4 = list(map(round, zone_p4))
56
57              zone_coordinates = np.array([zone_p1, zone_p2, zone_p3, zone_p4])
58              zone_name = f"Zone {count_zone}"
59
60              zone = {
61                  'name': zone_name,
62                  'polygon': zone_coordinates,
63                  'count': 0
64              }
65
66              zone['PolygonZone'] = PolygonZone(
67                  polygon=zone['polygon'],
68                  frame_resolution_wh=frame_wh
69              )
70
71              zone['PolygonZoneAnnotator'] = PolygonZoneAnnotator(
72                  zone=zone['PolygonZone'],
73                  color=sv.Color.WHITE,
74                  thickness=2,
75                  text_thickness=1,
76                  text_scale=0.3,
77                  text_padding=3
78              )
79
80              zones.append(zone)
81
82      return zones
83
84
85  def setup_zone_below(detections, frame_wh):
86      zones = []
87      count_zone = 0
88      for i in range(len(detections)):
89          obj = detections[i]
90
91
92          # choosing only baskets with confidence score >= 0.5
93          if (obj.class_id[0] == 1) & (obj.confidence[0] >= 0.5):
94              count_zone += 1
95
96              obj_xyxy = obj.xyxy[0]
97
98              # get coordinates of 4 points of bounding box
99              obj_p1 = np.array([obj_xyxy[0], obj_xyxy[1]])
100             obj_p2 = np.array([obj_xyxy[0], obj_xyxy[3]])
101             obj_p3 = np.array([obj_xyxy[2], obj_xyxy[3]])
102             obj_p4 = np.array([obj_xyxy[2], obj_xyxy[1]])
```

```
103
104             # get height and width of bounding box
105             obj_height = obj_xyxy[3] - obj_xyxy[1]
106             obj_width = obj_xyxy[2] - obj_xyxy[0]
107
108             # polygon construction strategy
109             # 1) zone width: the width of the     basket     bounding box decreased on
    95% (47.5% from both
110             # left and right sides).
111             # 2) zone height: 80% of the height of the "basket" bounding box.
112             # 3) zone location: lower on 70% of the "basket" bounding box height from
    the lower bound
113             # of "basket" bounding box.
114
115             zone_p1 = [obj_p1[0] + 0.475 * obj_width, obj_p1[1] + 0.7 * obj_height]
116             zone_p2 = [obj_p1[0] + 0.475 * obj_width, obj_p1[1] + 1.5 * obj_height]
117             zone_p3 = [obj_p4[0] - 0.475 * obj_width, obj_p4[1] + 1.5 * obj_height]
118             zone_p4 = [obj_p4[0] - 0.475 * obj_width, obj_p4[1] + 0.7 * obj_height]
119
120             # round coordinates of each point of zone
121             zone_p1 = list(map(round, zone_p1))
122             zone_p2 = list(map(round, zone_p2))
123             zone_p3 = list(map(round, zone_p3))
124             zone_p4 = list(map(round, zone_p4))
125
126             zone_coordinates = np.array([zone_p1, zone_p2, zone_p3, zone_p4])
127             zone_name = f"Zone {count_zone}"
128
129             zone = {
130                 'name': zone_name,
131                 'polygon': zone_coordinates,
132                 'count': 0
133             }
134
135             zone['PolygonZone'] = PolygonZone(
136                 polygon=zone['polygon'],
137                 frame_resolution_wh=frame_wh
138             )
139
140             zone['PolygonZoneAnnotator'] = PolygonZoneAnnotator(
141                 zone=zone['PolygonZone'],
142                 color=sv.Color.WHITE,
143                 thickness=2,
144                 text_thickness=1,
145                 text_scale=0.3,
146                 text_padding=3
147             )
148
149             zones.append(zone)
150
151     return zones
152
153
154 def setup_zone_general(detections, frame_wh, zones_above, zones_below):
155     general_zones = []
156
157     for count_zone in range(len(zones_above)):
158
159         # polygon construction strategy
160         # 1) zone width: the width of the zone_above increased on 100% (50% from both
161         # left and right sides)
162         # 2) zone height: 260% of the height of the zone_above.
163         # 3) zone location: lower on 10% of the height of zone_above from the lower
    bound of zone_above
164
165         height_zone_above = zones_above[count_zone]['polygon'][1][1] - zones_above[
    count_zone]['polygon'][0][1]
166         y1 = round(zones_above[count_zone]['polygon'][0][1] - 2.5 * height_zone_above)
     # the upper y coordinate
167         y2 = round(zones_above[count_zone]['polygon'][1][1] + 0.1 * height_zone_above)
     # the lower y coordinate
```

```
        width = zones_above[count_zone]['polygon'][3][0] - zones_above[count_zone]['polygon'][0][0]
        x1 = round(zones_above[count_zone]['polygon'][0][0] - 0.5 * width)
        x2 = round(zones_above[count_zone]['polygon'][3][0] + 0.5 * width)

        general_zone_p1 = [x1, y1]
        general_zone_p2 = [x1, y2]
        general_zone_p3 = [x2, y2]
        general_zone_p4 = [x2, y1]

        general_zone_coord = np.array([general_zone_p1, general_zone_p2, general_zone_p3, general_zone_p4])

        general_zone_name = f"Zone {count_zone + 1}"

        general_zone = {
            'name': general_zone_name,
            'polygon': general_zone_coord,
            'count': 0
        }

        general_zone['PolygonZone'] = PolygonZone(
            polygon=general_zone['polygon'],
            frame_resolution_wh=frame_wh
        )

        general_zone['PolygonZoneAnnotator'] = PolygonZoneAnnotator(
            zone=general_zone['PolygonZone'],
            color=sv.Color.WHITE,
            thickness=2,
            text_thickness=1,
            text_scale=0.3,
            text_padding=3
        )

        general_zones.append(general_zone)

    return general_zones


def choosing_largest_zone(zones):
    max_square = 0
    for zone in zones:
        zone_coord = zone.get('polygon')
        x1 = zone_coord[0][0]
        x2 = zone_coord[3][0]
        y1 = zone_coord[0][1]
        y2 = zone_coord[0][1]

        square = (x2 - x1) * (y2 - y1)

        if square > max_square:
            max_square = square

    for zone in zones:
        zone_coord = zone.get('polygon')
        x1 = zone_coord[0][0]
        x2 = zone_coord[3][0]
        y1 = zone_coord[0][1]
        y2 = zone_coord[0][1]

        if (x2 - x1) * (y2 - y1) == max_square:
            return zone


def filter_balls(detections: sv.Detections):
    """
    Remove all instances of objects with class_id = 0 (ball instances), except for the one with the highest confidence score

    Return: filtered detections
```

```
    """

    # Get the indices of detections with class_id = 0
    indices_class_0 = np.where(detections.class_id == 0)[0]

    # Get the indices of detections with class_id != 0
    indices_other_classes = np.where(detections.class_id != 0)[0]

    # If there are any detections with class_id = 0
    if len(indices_class_0) > 0:
        # Find the index of the detection with the highest confidence score among class_id = 0
        best_class_0_index = indices_class_0[np.argmax(detections.confidence[indices_class_0])]

        # Create a mask to include only the best detection with class_id = 0 and others
        mask = np.zeros(detections.class_id.shape, dtype=bool)
        mask[best_class_0_index] = True
        mask[indices_other_classes] = True

        # Apply the mask to filter the attributes
        filtered_detections = sv.Detections(
            xyxy=detections.xyxy[mask],
            confidence=detections.confidence[mask],
            class_id=detections.class_id[mask],
            mask=detections.mask,
            tracker_id=detections.tracker_id
        )
    else:
        # If no detections with class_id = 0, just keep the detections with other class_ids
        filtered_detections = sv.Detections(
            xyxy=detections.xyxy[indices_other_classes],
            confidence=detections.confidence[indices_other_classes],
            class_id=detections.class_id[indices_other_classes],
            mask=detections.mask,
            tracker_id=detections.tracker_id
        )

    return filtered_detections


def remove_ball_detections(detections: sv.Detections, ball_coord, ball_prev_coord):
    """
    Protection against accidential detections:
    if the distance between two same points of the ball on the current and previous frames differs by 3 widths of the ball,
    we consider such detection of the ball to be random and false, so we remove this ball instance from detections

    Return: filtered detections if the condition is true, and unchanged detections if the condition is false
    """

    ball_p1 = ball_coord[0]
    ball_p2 = ball_coord[1]
    ball_width = abs(ball_p1[0] - ball_p2[0])

    ball_prev_p1 = ball_prev_coord[0]

    if abs(ball_p1[0] - ball_prev_p1[0]) > 3 * ball_width and ball_width != 0:
        # Filter out the detections with class_id != 0
        mask = detections.class_id != 0

        # Apply the mask to filter the attributes
        filtered_detections = sv.Detections(
            xyxy=detections.xyxy[mask],
            confidence=detections.confidence[mask],
            class_id=detections.class_id[mask],
            mask=detections.mask,
```

```
            tracker_id=detections.tracker_id
        )
        ball_coord = ((0, 0), (0, 0))

        return filtered_detections, ball_coord

    return detections, ball_coord

def video_callback(model, frame: np.ndarray, frame_wh, frame_num: int, zone_above, zone_below, zone_general, zones_identified, zone_above_triggered, zone_general_triggered, score, throws, ball_prev_coord) -> np.ndarray:

    detections = detect(frame, model)
    detections = filter_balls(detections)

    # get ball bounding boxes coordinates of points of main diagonal
    ball_p1 = ball_p2 = (0, 0)
    for obj in iter(detections):
        if obj[3] == 0:
            ball_x1, ball_y1, ball_x2, ball_y2 = obj[0]
            ball_p1 = (ball_x1, ball_y1)
            ball_p2 = (ball_x2, ball_y2)
    ball_coord = (ball_p1, ball_p2)

    # Protection against accidential ball detections
    detections, (ball_p1, ball_p2) = remove_ball_detections(detections, ball_coord, ball_prev_coord)

    intersect_person_ball = False
    for obj in iter(detections):
        if obj[3] == 2:
            person_x1, person_y1, person_x2, person_y2 = obj[0]
            person_p1 = (person_x1, person_y1)
            person_p2 = (person_x2, person_y2)

            # True, if person and ball bounding boxes intersect
            intersect_person_ball = rectangles_intersect(person_p1, person_p2, ball_p1, ball_p2)

            if intersect_person_ball:
                break

    if not zones_identified:
        zones_above = setup_zone_above(detections, frame_wh)
        zones_below = setup_zone_below(detections, frame_wh)
        zones_general = setup_zone_general(detections, frame_wh, zones_above, zones_below)

        # choose only one zone above, below and general by greatest size
        zone_above = choosing_largest_zone(zones_above)
        zone_below = choosing_largest_zone(zones_below)
        zone_general = choosing_largest_zone(zones_general)

        if (zone_above, zone_below, zone_general) != (None, None, None):
            zones_identified = True


    # bounding boxes labels
    labels = [
        f"{model.model.names[class_id]} {confidence:0.2f}"
        for confidence, class_id
        in zip(detections.confidence, detections.class_id)
    ]

    # draw bounding boxes with labels
    annotated_frame = frame.copy()
    annotated_frame = bounding_box_annotator.annotate(
        scene=annotated_frame,
        detections=detections)
    annotated_frame = label_annotator.annotate(
        scene=annotated_frame,
```

```
366         detections=detections,
367         labels=labels)
368
369
370     # Draw text for person and ball intersection
371     text_anchor = sv.Point(x=500, y=200)
372     if intersect_person_ball:
373         annotated_frame = draw_text(scene=annotated_frame, text=f"Person and ball
    intersect: {intersect_person_ball}", text_anchor=text_anchor, text_scale=1,
    text_thickness=2, text_color=Color.GREEN)
374     else:
375         annotated_frame = draw_text(scene=annotated_frame, text=f"Person and ball
    intersect: {intersect_person_ball}", text_anchor=text_anchor, text_scale=1,
    text_thickness=2, text_color=Color.RED)
376
377     if zones_identified:
378         zone_presence_above = zone_above['PolygonZone'].trigger(detections, 0)
379         zone_presence_below = zone_below['PolygonZone'].trigger(detections, 0)
380         zone_presence_general = zone_general['PolygonZone'].trigger(detections, 0)
381
382         # write text for above zone
383         text_anchor = sv.Point(x=500, y=50)
384         if zone_presence_above and not intersect_person_ball:
385             annotated_frame = draw_text(scene=annotated_frame, text=f"Ball in above
    zone: {zone_presence_above}", text_anchor=text_anchor, text_scale=1, text_thickness
    =2, text_color=Color.GREEN)
386             zone_above['count'] += 1
387             zone_above_triggered = True
388         else:
389             annotated_frame = draw_text(scene=annotated_frame, text=f"Ball in above
    zone: {zone_presence_above}", text_anchor=text_anchor, text_scale=1, text_thickness
    =2, text_color=Color.RED)
390
391         # write text for below zone
392         text_anchor = sv.Point(x=500, y=100)
393         if zone_presence_below and not intersect_person_ball:
394             annotated_frame = draw_text(scene=annotated_frame, text=f"Ball in below
    zone: {zone_presence_below}", text_anchor=text_anchor, text_scale=1, text_thickness
    =2, text_color=Color.GREEN)
395             zone_below['count'] += 1
396         else:
397             annotated_frame = draw_text(scene=annotated_frame, text=f"Ball in below
    zone: {zone_presence_below}", text_anchor=text_anchor, text_scale=1, text_thickness
    =2, text_color=Color.RED)
398
399         # write text for general zone
400         text_anchor = sv.Point(x=500, y=150)
401         if zone_presence_general and not intersect_person_ball:
402             annotated_frame = draw_text(scene=annotated_frame, text=f"Ball in general
    zone: {zone_presence_general}", text_anchor=text_anchor, text_scale=1,
    text_thickness=2, text_color=Color.GREEN)
403             zone_general['count'] += 1
404             zone_general_triggered = True
405         else:
406             annotated_frame = draw_text(scene=annotated_frame, text=f"Ball in general
    zone: {zone_presence_general}", text_anchor=text_anchor, text_scale=1,
    text_thickness=2, text_color=Color.RED)
407
408
409
410         # if the above zone is triggered by ball, we wait for the zone below to also
    be triggered by ball.
411         # If the outcome when player touches the ball ccures earlier, than the zone
    below is triggered by ball, then we reset the zone_above_triggered to False.
412         if zone_above_triggered and zone_presence_below and not intersect_person_ball:
413             score += 1
414             zone_above_triggered = False
415         elif zone_above_triggered and intersect_person_ball:
416             zone_above_triggered = False
417
418         # if the general zone is triggered by ball, we wait till the player touches
```

```
    the ball. Then we conclude that the throw was made
        if zone_general_triggered and intersect_person_ball:
            throws += 1
            zone_general_triggered = False



        annotated_frame = zone_above['PolygonZoneAnnotator'].annotate(
            scene=annotated_frame,
            label=f"{zone_above['name']}: {zone_above['count']}"
        )

        annotated_frame = zone_below['PolygonZoneAnnotator'].annotate(
            scene=annotated_frame,
            label=f"{zone_below['name']}: {zone_below['count']}"
        )

        annotated_frame = zone_general['PolygonZoneAnnotator'].annotate(
            scene=annotated_frame,
            label=f"{zone_general['name']}: {zone_general['count']}"
        )

    # draw text for score
    text_anchor = sv.Point(x=130, y=50)
    annotated_frame = draw_text(scene=annotated_frame, text=f"Score: {score}",
    text_anchor=text_anchor, text_scale=1.5, text_thickness=2)

    # draw text for throws
    text_anchor = sv.Point(x=130, y=100)
    annotated_frame = draw_text(scene=annotated_frame, text=f"Throws: {throws}",
    text_anchor=text_anchor, text_scale=1.5, text_thickness=2)

    ball_prev_coord = ball_coord

    return annotated_frame, zone_above, zone_below, zone_general, zones_identified,
    zone_above_triggered, zone_general_triggered, score, throws, ball_prev_coord


import asyncio
from src.process_video.connection_manager import ConnectionManager

def video_process(SOURCE_VIDEO_PATH, TARGET_VIDEO_PATH, manager: ConnectionManager,
    loop, result_holder, stop_event):
    HOME = os.getcwd()

    MODEL = f"{HOME}/best.pt"
    model = YOLO(MODEL)

    frames_generator = sv.get_video_frames_generator(source_path=SOURCE_VIDEO_PATH)
    video_info = sv.VideoInfo.from_video_path(video_path=SOURCE_VIDEO_PATH)
    total_frames = video_info.total_frames

    zone_above = {}
    zone_below = {}
    zone_general = {}
    zones_identified = False
    zone_above_triggered = False
    zone_general_triggered = False
    score = 0
    throws = 0
    ball_prev_coord = ((0, 0), (0, 0))

    with sv.VideoSink(target_path=TARGET_VIDEO_PATH, video_info=video_info) as sink:
        for i, frame in enumerate(frames_generator):
            if stop_event.is_set():
                print("Stopping video processing as requested.")
                break

            # Infer
            annotated_frame, zone_above, zone_below, zone_general, zones_identified,
    zone_above_triggered, zone_general_triggered, score, throws, ball_prev_coord = \
```

```
            video_callback(model, frame, video_info.resolution_wh, i, zone_above, zone_below, zone_general, zones_identified, zone_above_triggered, zone_general_triggered, score, throws, ball_prev_coord)

            print(f"Processing frame {i+1}/{total_frames}, Score {score}, Throws {throws}")
            sink.write_frame(frame=annotated_frame)

            progress = (i+1) / total_frames * 100
            asyncio.run_coroutine_threadsafe(manager.send_progress(progress), loop)
            # asyncio.run_coroutine_threadsafe(manager.send_progress(f"{progress}"), loop)

    misses = throws - score

    # Store results in the result holder
    result_holder['score'] = score
    result_holder['misses'] = misses
    result_holder['throws'] = throws

    if not stop_event.is_set():
        asyncio.run_coroutine_threadsafe(manager.send_completion_message(), loop)
```

Listing 20: functions.py

models.py

```
from sqlalchemy import Table, Column, Integer, String, TIMESTAMP, MetaData, ForeignKey
from src.auth.models import user
from datetime import datetime

metadata = MetaData()

video = Table(
    "video",
    metadata,
    Column("id", Integer, primary_key=True),
    Column("name_video", String, nullable=False),
    Column("video_size", Integer, nullable=False),
    Column("number_throws", Integer, nullable=False),
    Column("number_goals", Integer, nullable=False),
    Column("number_misses", Integer, nullable=False),
    Column("accuracy", Integer, nullable=False),
    Column("throw_type", String, nullable=False),
    Column("date_upload", TIMESTAMP, default=datetime.utcnow),
    Column("user_id", Integer, ForeignKey(user.c.id)),
    Column("video_path", String, nullable=False),  # New column for video path
)
```

Listing 21: models.py

router_videos.py

Listing 22: router_videos.py

video_func.py

```
from fastapi import HTTPException, status
from fastapi import UploadFile
from moviepy.editor import VideoFileClip
import moviepy.editor as mp
import moviepy.config as mp_conf
import os

# check the file format
def check_file_format(filename):
    allowed_format = 'mp4'
    file_ext = filename[filename.rfind('.'):].lower()
    if file_ext == f'.{allowed_format}':
        return True
    else:
        return False
```

```python

# saves the video to disk
async def save_video(video: UploadFile, video_path: str) -> None:
    with open(video_path, "wb") as video_file:
        video_file.write(await video.read())


# saves the video in the specified directory
async def prepare_video_files(Video):
    UPLOADS_DIR = "result_videos"
    os.makedirs(UPLOADS_DIR, exist_ok=True)

    video_path = os.path.join(UPLOADS_DIR, Video.filename)
    await save_video(Video, video_path)

    processed_filename = f"processed_{Video.filename}"
    processed_path = os.path.join(UPLOADS_DIR, processed_filename)

    return video_path, processed_path

# change the result video fps
def change_fps(processed_video_path):

    result_video_path = processed_video_path.replace("processed_", "result_")
    # Desired frame rate
    desired_frame_rate = 30
    # Load the video
    clip = VideoFileClip(processed_video_path)
    # Set the new frame rate
    clip = clip.set_fps(desired_frame_rate)
    # Write the result to a file
    clip.write_videofile(result_video_path, codec='libx264')
    return result_video_path


# calculates the size of the video
async def calculate_video_size(file_path):
    try:
        size_in_bytes = os.path.getsize(file_path)
        return int(size_in_bytes)
    except FileNotFoundError:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="
                    ")
```

Listing 23: video_func.py

11.5 Ethics Approval

DUNDALK INSTITUTE OF TECHNOLOGY
School of Informatics & Creative Arts
Ethical Approval Form for Research Projects

Researcher Name____Gleb Bikushev____ Year ___1___ Course __MSc in Data Analytics__

Title of project: Computer Vision for Identification and Measurement of Basketball Scoring.

Name of supervisor/s ___ Kevin Mc Daid _____ Date ___16.02.2024___
(if applicable)

This application is to be completed by the researcher and where appropriate, in conjunction with the project supervisor. The lead researcher/supervisor is responsible for submitting the completed form to the appropriate Research Ethics Committee (details below)

Please note: If your submission is incomplete or unclear, your application will be returned to you and your project may be delayed.

Section 1

Type of Researcher

Please tick the appropriate box below to indicate the type of researcher you are:

Undergraduate
(proceed to section 2) ☐

Completed Ethical Approval forms for undergraduate research should be submitted to the relevant Departmental Research Ethics Committee (DREC).

Drec.dcamm@dkit.ie – Department of Creative Arts Media and Music
Drec.dcsm@dkit.ie – Department of Computing Science and Mathematics
Drec.dvhcc@dkit.ie – Department of Visual and Human Centred Computing

Postgraduate
(proceed to section 3) ☒ Y

Completed Ethical Approval forms for Postgraduate research should be submitted directly to the School Research Ethics Committee (SREC).

Srec.ica@dkit.ie

Staff
(proceed to section 3) ☐

Completed Ethical Approval forms for Staff research should be submitted directly to the School Research Ethics Committee (SREC).

Srec.ica@dkit.ie

Section 2

Please complete questions 1-4 listed below.

	Human and / or Animal Research	YES	NO
1	**Does your research involve human participants other than the following[1]?** • Research using exclusively secondary sources. • Research using materials legally accessible to the public that have legal protection, e.g., record of court judgements, data archives. • Research using materials that are publicly accessible and where there is no reasonable expectation for privacy, e.g., books, published third party interviews. • Observations of human behaviour in public where (i) those being observed have no reasonable expectation of privacy, (ii) there is no intervention on the part of the researcher nor any interaction between the researcher and those observed, and (iii) individuals are not identifiable in the results. If '**YES**', please complete **B and C** below.	X	
2	**Does your project involve working with animals?** – If '**YES**', please complete **B and C** below. – Please note that for ethical consideration: 'Animals' are classed as vertebrate animals including cyclostomes and cephalopods (DIRECTIVE 2010/63/EU)		**N**
3	**Does your project involve working with participants from any of the following categories?** • Minors (under 18 years of age) • People with learning or communication difficulties • Patients • People in custody • People engaged in illegal activities If '**YES**', please complete **D** below.		**N**
4	**Does your project have any possible ethical implications other than those outlined in questions 1, 2 and 3?** If '**YES**', please complete **E** below.		**N**

A. **I consider that this project has no significant ethical implications[2] to be brought through the ICA School Ethics Review Process**
Please complete Section 4

☐

[1] If there is any doubt, researchers should contact the Chair of the SREC.
[2] In determining significant implications, please consider all potential risks attached to this project. If there is any doubt, researchers should contact the Chair of the SREC.

B.

I. **Is this study part of a larger project that already has ethical clearance?**

YES / NO

If **YES** please answer question B II.
If **NO** please answer question C.

II. If this study is part of a larger project that already has ethical clearance, are you proposing any changes to the operational plan already ethically approved?

YES / NO

If **YES,** please complete *Sections 3 and 4.*
If **NO,** then please provide the project details below and complete *Section 4*.

Title of project with ethical clearance: _____

C. Could this project have ethical implications that should be brought before the appropriate ICA Departmental Ethics Review Committee as it will be carried out with human participants?

YES / NO

If **YES,** please complete *Sections 3 and 4.*
If **NO,** please complete *Section 4.*

D. I consider that this project may have ethical implications that should be brought before the School Research Ethics Committee as it will be carried out with human participants in a "vulnerable" category.
Please complete Sections 3 and 4 ☐

All research carried out with human participants in a vulnerable category must be referred by the Departmental Research Ethics Committee to the School Research Ethics Committee for approval.

E. Could this project have ethical implications, other than those previously outlined, that should be brought before the appropriate ICA Departmental Ethics Review Committee?

YES / NO

If **YES,** please complete *Sections 3 and 4.*
If **NO,** please complete *Section 4.*

Section 3

3.1 Application Form Checklist

Please complete Section 3 and provide additional information as attachments.

My application includes the following documentation:	**INCLUDED (mark as YES)**	**NOT APPLICABLE (mark as N/A)**
Recruitment advertisement		**N/A**
Participant Information Leaflet	**X**	
Participant Informed Consent form	**X**	
Questionnaire/Survey		**N/A**
Interview/Focus Group Questions		**N/A**
Debriefing material		**N/A**
Evidence of approval to gain access to off-site location		**N/A**
Ethical approval from external organizations. If ethical approval from external organizations is pending give details below		**N/A**

Note that instructions for the participant are included with the information leaflet.

3.2 Project Details

a) Lay description (Maximum 200 words)

Please outline, in terms that any non-expert would understand, what your research project is about, including what participants will be required to do. Please explain any technical terms or discipline-specific phrases.

<u>Summary</u>

The research project involves building a mobile application that will detect 3 objects: humans, ball and basketball hoop. The detection will be available in 2 options: using a pre-recorded video and real-time video shooting. This object detection feature will allow its users to evaluate the accuracy of hitting the basketball hoop by counting how many times a player successfully threw the ball into the basketball hoop and how many times he missed the shot.

The project will use a combination of publicly available basketball video and video provided by amateur and professional adult basketball players following a protocol specified by the researcher. The main body of this application refers to this element of the application.

YouTube Videos

In relation to the acquisition of video from the YouTube platform, when we state publicly available basketball video we mean video data that can be accessed through the YouTube platform without any sign in required by the accessor. The videos selected will be restricted to simple shooting videos (rather than game videos) which show single players shooting at the basket from a range of positions. It is intended that the number of videos selected will be in the order of 100. The selection of these videos will based on criteria which require a single adult basketball player to be in the frame shooting at a basket. Only this portion of the video will be used.

For these videos, we do not propose to directly gain the approval of those uploading the video to YouTube or even approval from those players in the videos. However, and importantly, the project will not seek to obtain or store any personal details relating to the players or the user account the videos were sourced from. Furthermore, the work is not commercial in nature.

The use of “publicly available” social media data in scientific research is an interesting one. A very recent review of the use of youtube information [1] in research in the much more controlled area of health research showed that of 119 articles published in academic journals in areas such as chronic desease and mental health, 69% made no mention of ethical considerations, 13% stated that they did not meet the definition of human participation and of the remainder 80% were determined by IRB not to meet the criteria for human participant research. Only 1 of the 119 studies actually sought informed consent from YouTube users and it was not clear whether this study was in a particularly sensitive area of health research or not or whether it sought to profile users in any way.

Participant Videos

The study will also gather video from individual adult basketball players gathered by participants at a publicly accessible basketball court of their choosing. Appropriate consent and information forms are included in this revised application. Note that section 3 relates to this element of the study which gathers data directly from participants.

Note on Mobile Application

Finally, although a final mobile application will be developed there is no intention to evaluate this application using human participants.

Reference

[1] Tanner, J.P., Takats, C., Lathan, H.S., Kwan, A., Wormer, R., Romero, D. and Jones, H.E., 2023. Approaches to research ethics in health research on YouTube: systematic review. Journal of Medical Internet Research, 25, p.e43060.

b) Research objectives (Maximum 150 words)

Please summarise briefly the objectives of the research.

- Building deep learning mode computer vision model using available technologies such as YOLO l that will detect 3 objects: humans, ball and basketball hoop.
- Evaluation the performance of the computer vision model in the prediction of basketball shooting success.
- Development a function for counting the number of times a player successfully threw the ball into the basketball hoop.
- Development of a simple real time mobile application to automate the prediction of shooting success for controlled activities. Note that although a final mobile application will be developed there is no intention to evaluate this application using human participants.

c) Research location and duration

Location(s)/Population*	DkIT Carroll's Building
Research start date	26.01.2024
Research end date	01.10.2024
Approximate duration	9 months

* If location/Population other than DkIT campus/population, provide details of the approval to gain access to that location/population as an appendix.

The location of the research is DkIT. However, the location of the participant activity will be at publicly accessible basketball courts. It will be the responsibility of the participant to ensure that the courts are publicly accessible.

3.3 Participants

		YES	NO	N/A
Do participants fall into any of the following special groups?	**Minors (under 18 years of age)**		**N**	
	People with learning or communication difficulties		**N**	
	Patients		**N**	
	People in custody		**N**	
	People engaged in illegal activities (e.g. drug-taking)		**N**	
Have you given due consideration to the need for satisfactory Garda clearance?			**N**	

3.4 Sample Details

Approximate number	20
Where will participants be recruited from?	Experiences adult basketball players who are friends/acquaintances of the researcher
Inclusion Criteria	Adults age 18+ who have at least 2 years basketball experience
Exclusion Criteria	

Will participants be remunerated, and if so in what form?

No

Justification for proposed sample size and for selecting a specific gender, age, or any other group if this is done in your research.

To develop reliable computer vision models for the controlled activity a significant number of participants is required. It is hoped that this will generate a variety of court settings and

shooting styles.

3.5 Risk to Participants

a) Please describe any risks to participants that may arise due to the research. Such risks could include physical stress, emotional distress, perceived coercion e.g. lecturer interviewing own students. Detail the measures and considerations you have put in place to minimize these risks

b) What will you communicate to participants about any identified risks? Will any information be withheld from them about the research purpose or procedure? If so, please justify this decision.

3.6 Informed Consent

	YES	NO	N/A
Will you obtain active consent for participation?	X		
Will you describe the main experimental procedures to participants in advance?	X		
Will you inform the participants that their participation is voluntary and may be withdrawn at any point?	X		
If the research is observational, will you ask for their consent to being observed?			**N/A**
With questionnaires, will you give participants the option of omitting questions they do not want to answer?			**N/A**
Will you tell participants that their data will be treated with full confidentiality and that, if published, it will not be identifiable as theirs?	X		
Will the data be anonymous?	X		
Will you debrief participants at the end of their participation?			**N/A**
Will your project involve deliberately misleading participants in any way, or will information be withheld? **If you answer yes, give details and justification for doing this below.**			**N/A**

a) Please outline your approach to ensuring the confidentiality of data (that is, that the data will only be accessible to agreed upon parties and the safeguarding mechanisms you will put in place to achieve this.) You should include details on how and where the data will be stored, and who will have access to it.

The data will be stored on DkIT's OneDrive system and will only be directly accessible to the researcher and the research supervisors. Should the external examiner request access than this will be provided to them. Access to this OneDrive system will be securely password protected and access by the researcher to the data will be through a securely protected windows laptop device.

All communications relating to the project will be through secure DkIT systems including Teams and Outlook.

b) Please outline how long the data will be retained for, if it will be destroyed and how it will be destroyed.

The data will be deleted at the end of the study, September 2024 at the latest. The data will be destroyed by removable from the OneDrive system and secure deletion of the folder used to store all project artefacts.

1.Storage
DkIT OneDrive System

2. Access
Researcher and Superviors and possibly external examiner for the programme.

3. Communication
Through DkIT systems including Teams and Outlook (pulling from Microsoft Exchange Server)

4. Digital Platform Usage
NA

5. Data maintenance
NA

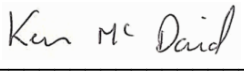
Section 4

Researcher I have read and I understand the DkIT Ethics Policy available from: https://www.dkit.ie/assets/uploads/documents/Research/Policies/DkIT%20Research%20Ethics%20Policy.pdf

Signed:  Print Name: ____Gleb Bikushev_____ Date: _15.04.2024_
(Researcher)

Supervisor: Applications for Ethical Approval of Undergraduate projects are forwarded to the Departmental Research Ethics Committee for approval or referral to the School Research Ethics Committee. Applications for Ethical Approval of Postgraduate and Staff projects are sent to the School Research Ethics Committee for Approval.

I have read and approved this form & information:

Signed:  Print Name: ___Kevin McDaid____ Date: _15.04.2024_
(Supervisor/Head of Department/ Research Centre Director/ Head of School)

There is an obligation on the researcher and/or supervisor to bring to the attention of the Departmental/School Research Ethics Committee(s): (a) Any issues with ethical implications not clearly covered by this form (b) Any ethical issues which may arise during the carrying out of the research; (c) Any ethically significant change made to the project after approval.

Section 5 (For office use only)

STATEMENT OF ETHICAL APPROVAL (FOR UNDERGRADUATE PROJECTS ONLY)

This project has been considered using agreed department procedures and is now:

Approved: [#]

Referred to the School Ethics Committee: [ ]

Signed:  _____ Print Name: _Adèle Commins_____ Date: 23.04.24____
(Chair of Departmental Research Ethics Committee/Head of Department)

STATEMENT OF ETHICAL APPROVAL

This project has been considered using agreed School procedures and is now:

Approved: [ ]

Rejected (further information sought): [ ]

Chair of School Research Ethics Committee
This project has been considered by the Ethics Committee and ethical approval is granted.

Signed: _____Print Name: _____ Date: _____
Chair of School Research Ethics Committee

Ethical Approval Application - Feedback Form	
Application No.	No. 14
Date.	16/02/2024
Applicants Name.	Gleb Bikushev
Supervisor.	Kevin McDaid
Project Title.	Computer Vision for Identification and Measurement of Sports/Other Activities
Decision. (Approved/ Not Approved/ Approved Subject to indicated Requirements)	Approved
Comments	All actions addressed and a more detailed application