Technical, Functional and Graphical Design for POC

Automatic score calculation for Shootsoft

Students: Ruben Stoop, Nick Glas, Jaimy Monsuur
Student numbers: 670240,670516,668040
Date: 14-2-2023
Class: INF3B
Version: 1

Content

Content **2**
Introduction **3**
1.1 Project Scope 3
Technical Design **4**
2.1 System context 4
2.2 Front end technical design 4
2.3 API Design 5
2.4 Entity relationship diagram 13
Functional Design **18**
1. Introduction 18
1.1 Purpose of the document 18
1.2 Scope of the document 18
2. System/ Solution Overview 19
2.1 Diagrams 19
2.2 System Actors 23
Graphical Design **24**
1. Front end design 25
Main menu 25
Taking pictures 26
Selected images 26
Selecting shooting discipline 27
Calculation screen 27
Results screen 28
Appendix **29**
Figma design 29
Division of tasks **29**
Roboflow 29
Swagger documentation 29

Introduction

This document contains the Technical, Functional and Graphical design for the Shootsoft project. All of the content containing this project can be found at the following git repository: https://github.com/neburpoots/shootsoft

1.1 Project Scope

ShootSoft is a company that specializes in software for shooting ranges. ShootSoft currently has an existing application that handles many aspects of the sport, but scorecards still need to be counted manually. However, there is a gap in the market as no commercial application focuses on Dutch shooting disciplines, which involve shooting with small caliber pistols, rifles, and carbines at a 10/12-meter distance due to limited space in the Netherlands.

ShootSoft would like to expand the existing application with the ability to automatically sum the shooting cards. This expansion will be done through a computer vision model. This model will be able to recognize the shooting cards and be able to see the holes in the paper created by a pellet gun. In turn these holes are added up based on the score of the designated area and a total will be calculated. The computer vision model will work alongside an API to receive images and send the appropriate data back.

The action plan is meant for the client, project leader and team members. The importance of this document is in that it can be referred to in the future for what will and will not fall within the scope of the project.

Technical Design

The purpose of this technical design document (TDD) is to explain in a detailed manner how a system or software application will be built, including the technical details, specifications, and requirements. It will serve as a for the development team. The technical design document (TDD) provides an overview of the system's architecture, data flow, interfaces, and functionality. The purpose of this document is to create a detailed plan for the development team to follow, ensuring that the system or application is built to meet the desired specifications and requirements. This document is written for the technical team, who will use it as a guide throughout the development process.

2.1 System context

The software development team is using Python version 3.10 This system relies on the YOLOV8 model to function, and the team plans to make it available to the public through an API. The API is built using the Flask framework version 2.2.3 and runs on the Werkzeug package version 2.2.2. For local development, the team utilizes a SQLite database and interacts with it using an ORM tool. Several Flask packages are also used, including Flask-Bcrypt version 1.0.1, Flask-Migrate version 4.0.4, Flask-RestX version 1.1.0, and Flask-SQLAlchemy version 3.0.3. To use the YOLOV8 model in the API, the team requires the ultralytics package version 8.0.58. All the packages are installed in a virtual environment. In summary, the dependencies of the text are Python 3.10, YOLOV8 model, Flask, Werkzeug, SQLite, ORM tool, Flask-Bcrypt, Flask-Migrate, Flask-RestX, Flask-SQLAlchemy, and ultralytics package.

All the packages/tools/software that the team uses are:

- Python version 3.10
- YOLOV8 model
- Flask framework version 2.2.3
- Werkzeug package version 2.2.2
- SQLite database
- ORM tool
- Flask-Bcrypt version 1.0.1
- Flask-Migrate version 4.0.4
- Flask-RestX version 1.1.0
- Flask-SQLAlchemy version 3.0.3
- Ultralytics package version 8.0.58

2.2 Front end technical design

The front-end design can be realized through a multitude of ways. The first proposal will be a web application making use of Next.js. Next.js is a full stack framework built on top of react. Currently the group has done testing and sees this as a realistic approach to get the results

which equal the design displayed in the graphical design. The great benefit of using a web application is that it can be used on any device. The second approach would be to use react native to build an application. The team currently has done some testing with using react native and does see this as a doable approach in terms of development. This would be the most clean solution for mobile phones. The downside of this is that the app will have to work on both IOS and android and be published in their respective stores.

2.3 API Design

An API design document is a document that outlines the design and functionality of an API. It serves as a blueprint for developers to follow when building the API and ensures that all parties involved have a clear understanding of the API's capabilities and limitations.

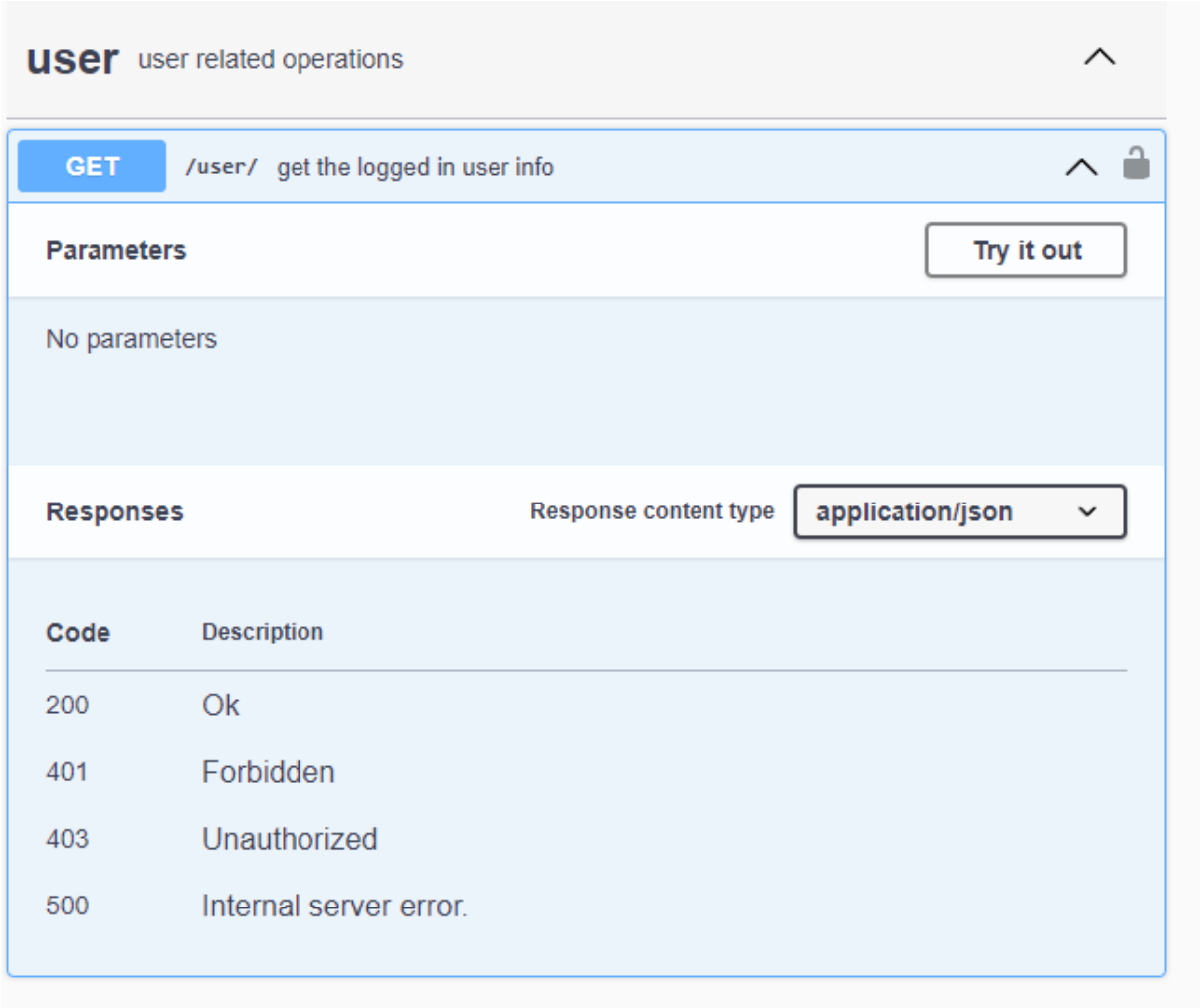
For our project, we used the API design template provided by our lecturer, Wiley. However, we had to strip it down a bit due to some packages being deprecated in our Windows environment.

In terms of technical details, our API design document includes five different routes that cover different aspects of our project. The first route is the user route, which includes options for user-related tasks. The second route is the detection route, which is responsible for handling detection tasks for us. The third route is the discipline route, which is used for getting information on the available disciplines. The fourth route is the training route, which is used to create/post, and view the history of trainings. Lastly, the authentication route, if applicable, is used for registering and logging in users.

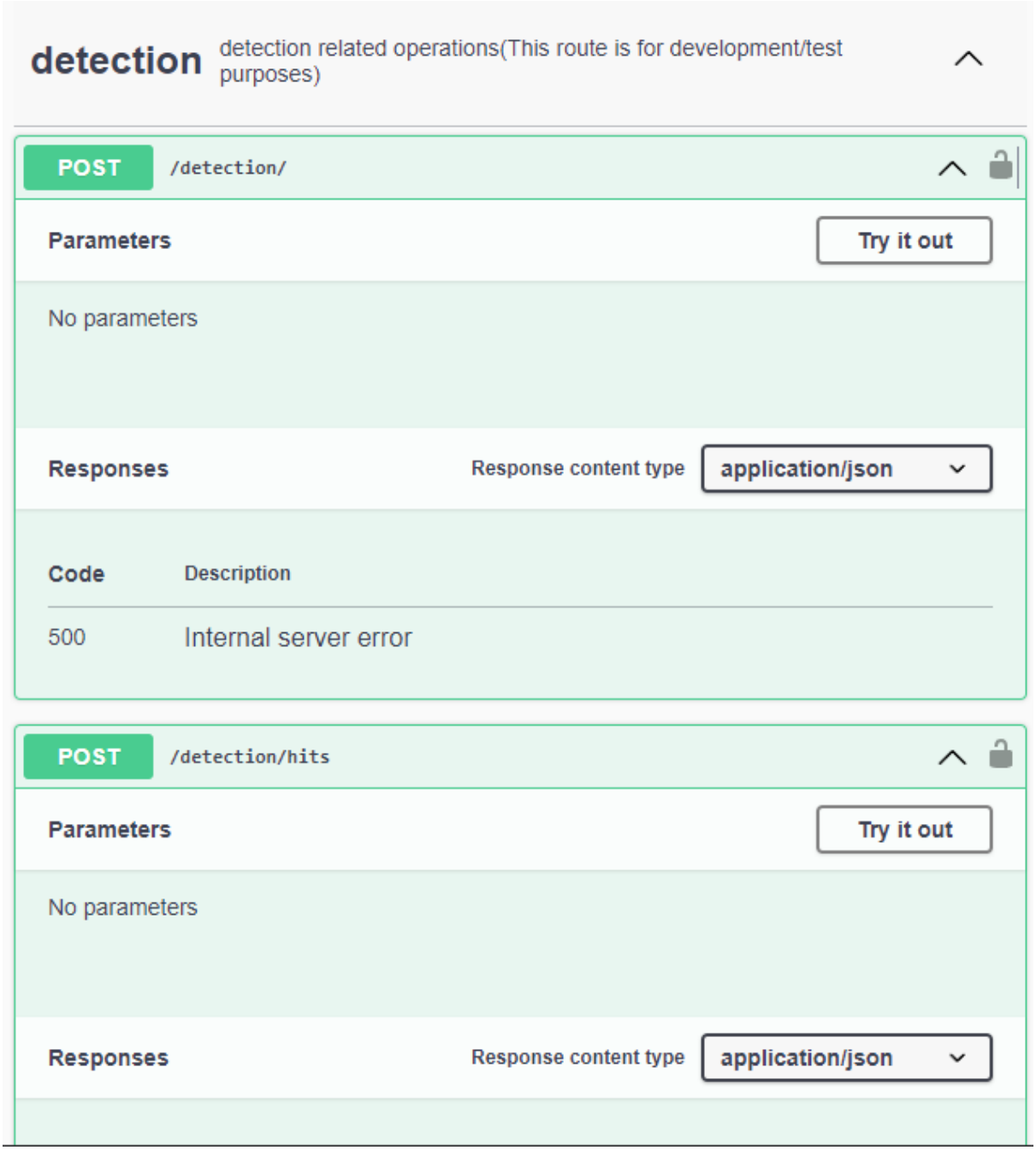
All of our routes, HTTP status codes, and other indicators have been automatically generated by Swagger, which has made the process of creating our API design document much simpler and more streamlined. Overall, our API design document serves as a comprehensive guide to the functionality and capabilities of our API and will be instrumental in ensuring that all developers involved in the project are on the same page. In this current iteration, we will not have any models. This means that the routes are build, but the required data is not defined yet.

All our code and the corresponding generated swagger file can be viewed/downloaded by going to the following GitHub repository: https://github.com/nickglas/flask_api or view the generated json document in the appendix.

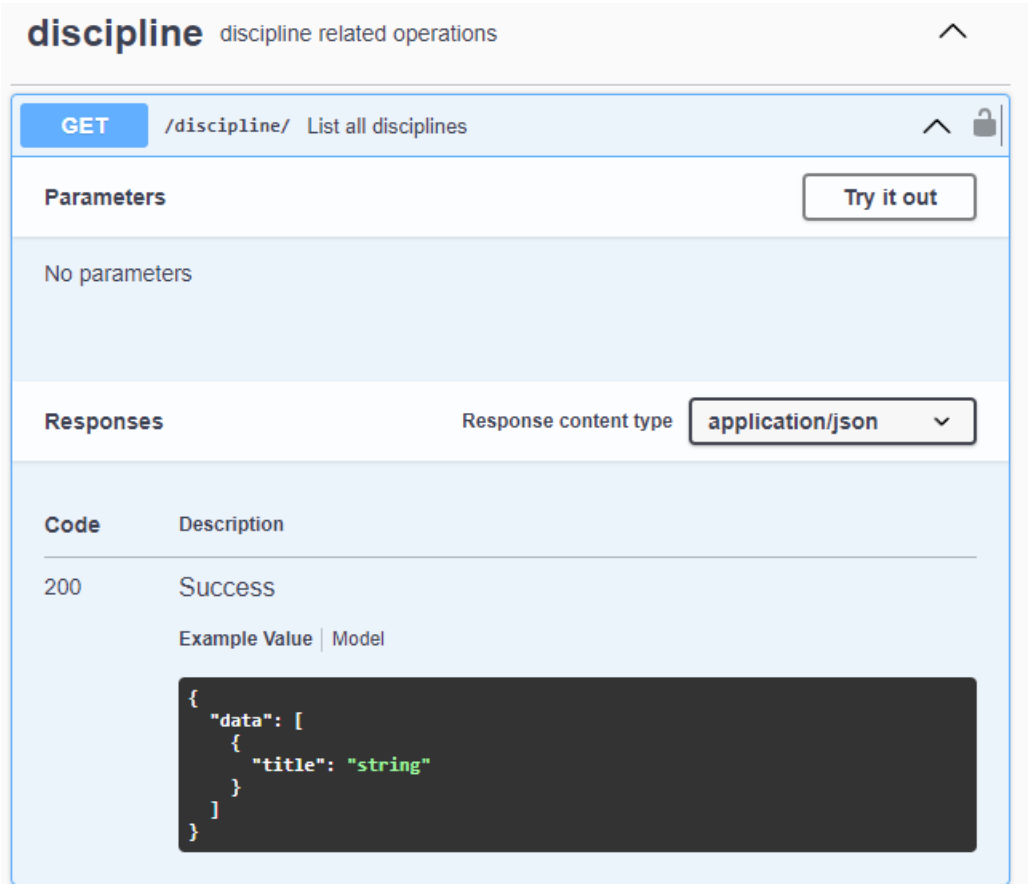
The user route holds one route. This route will get the user's personal information. This of course requires the user to be logged in.

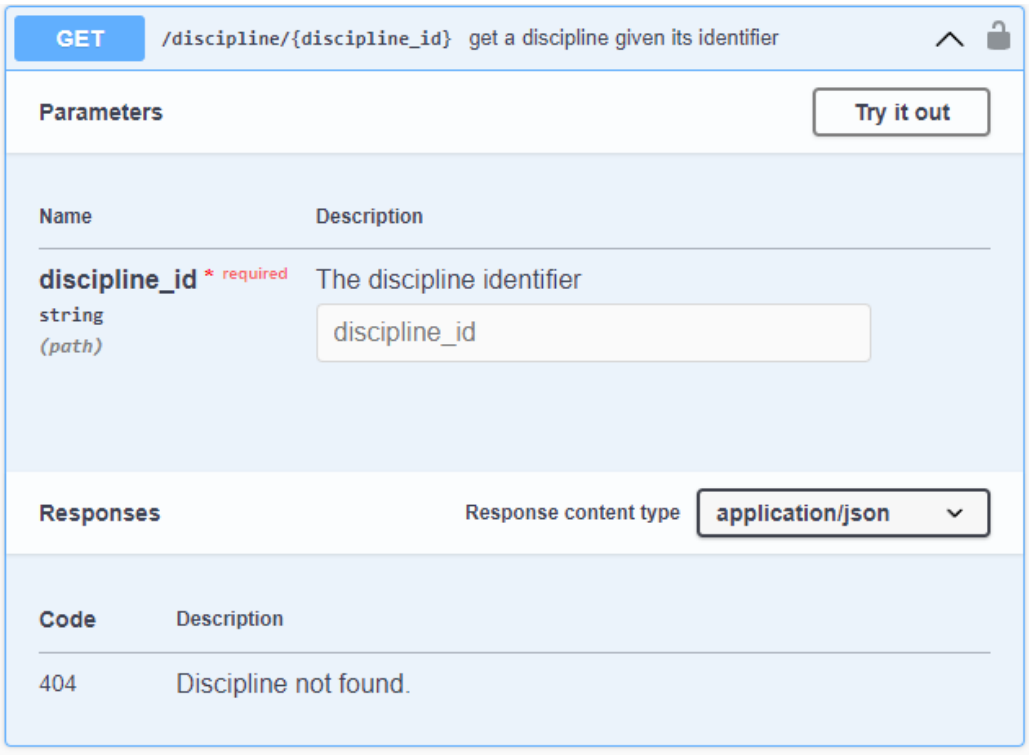


For development purposes, we included a detection route. This route is used to detect the hits inside an image. This route requires an image to be sent to the hits endpoint

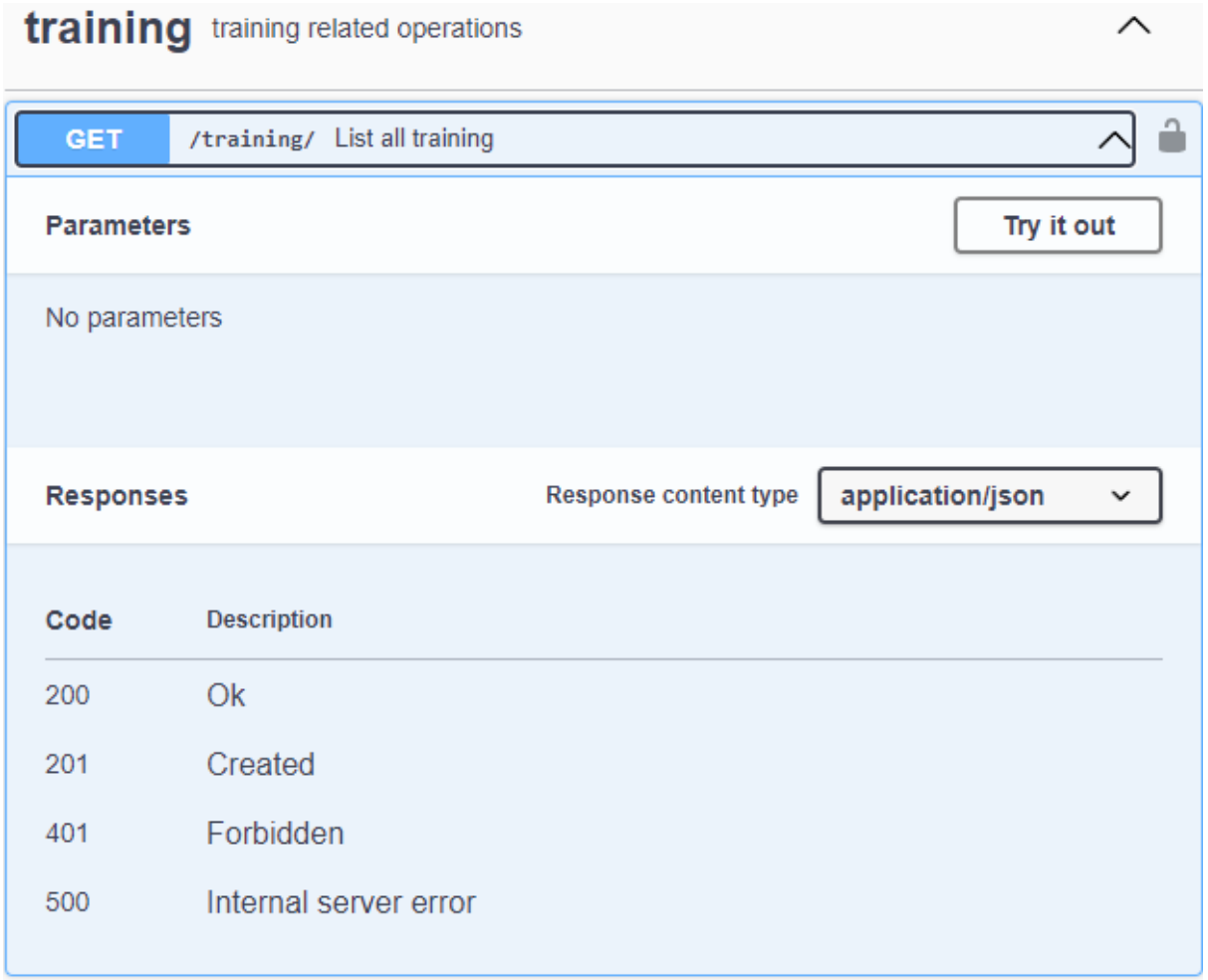


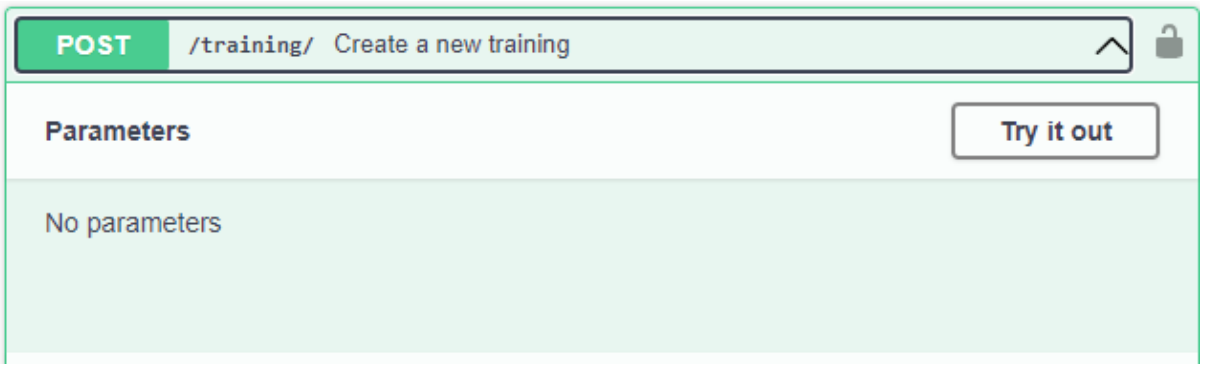
There are 2 discipline routes. The first get route will return all the available disciplines. The other get route, is a route to get the discipline by an identifier.





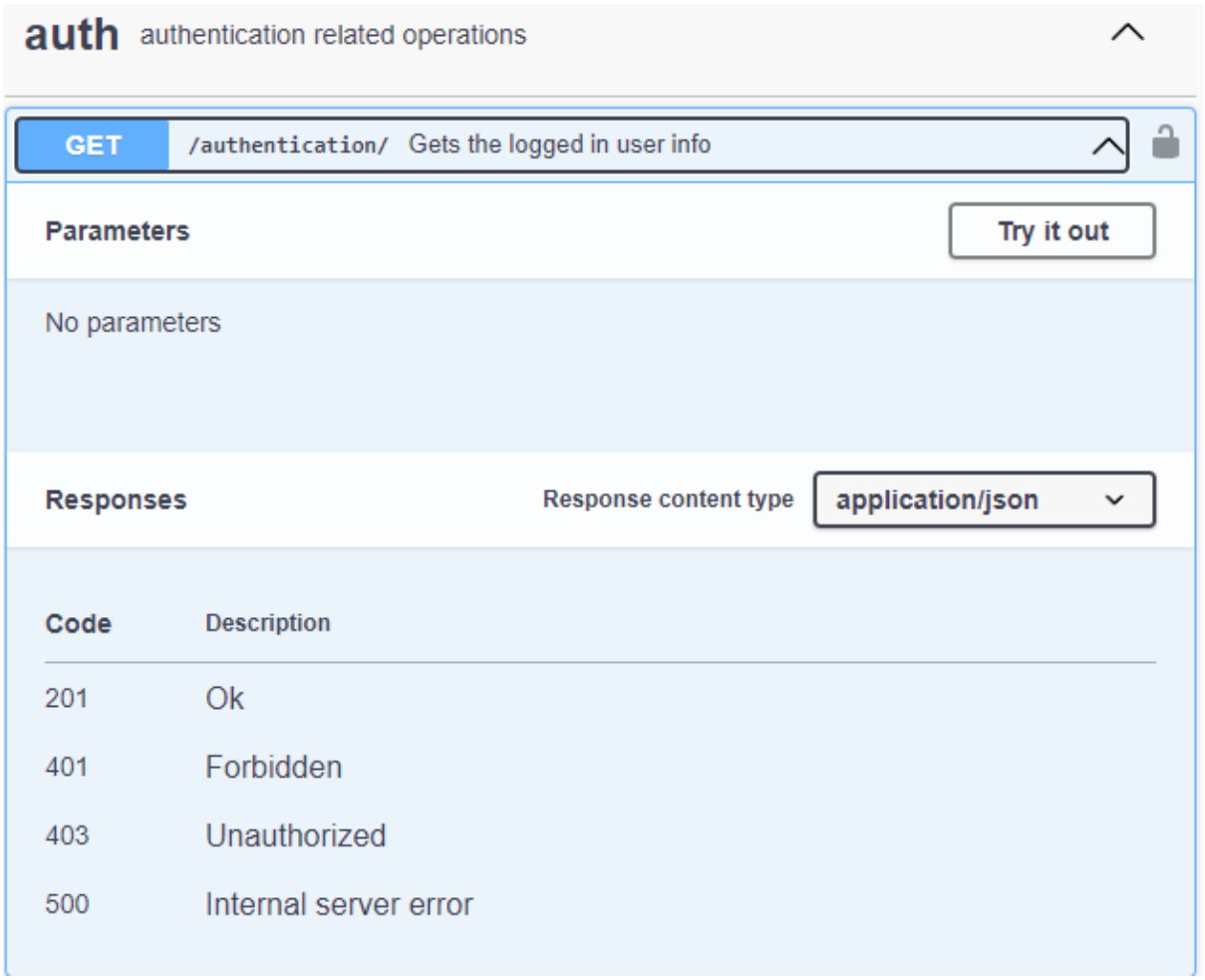
There are currently 2 training routes. The routes that we have are a get and post route. The first route is the get route, which will return all the training performed by a user. The post route is used when a user wants to create a new training. Currently, we do not have the required data for this route, but it is included.





Lastly, we have the authentication routes. These routes will support the following features.

- Login a user
- Logout a user
- Refresh a token
- Get the logged-in user his info



POST /authentication/login Login user

Parameters Try it out

No parameters

Responses Response content type application/json

Code	Description
201	Created
403	Unauthorized

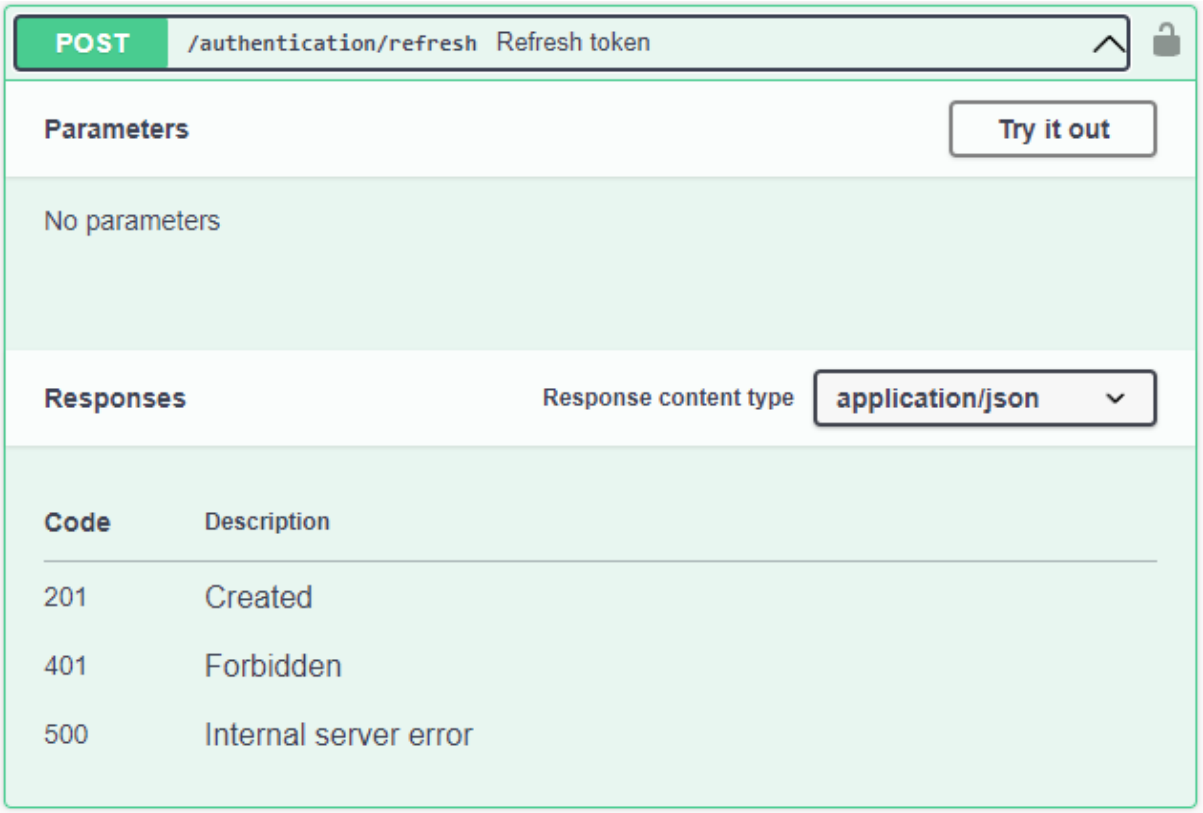
POST /authentication/logout Logout user

Parameters Try it out

No parameters

Responses Response content type application/json

Code	Description
200	Ok
401	Forbidden


POST
/authentication/refresh
Refresh token
Parameters
Try it out
No parameters
Responses
Response content type
application/json
Code
Description
201
Created
401
Forbidden
500
Internal server error

2.4 Entity relationship diagram

An entity relationship diagram (ERD) is a visual representation of entities (objects, concepts or things) and their relationships to each other. It shows how data is structured and related in a database or information system, helping to plan, design and understand complex data structures. We developed an ERD for the ShootSoft application. There are multiple ERD's for multiple use cases. The team developed 3 different types of entity relationship diagrams. Firstly, let's explain which tables/classes we will or might have, depending on the version.

- User: this table is used to store information about users in our application or system. In this case, the table contains 6 fields. This table is required if we want to save training data for our users. At this time, it is not clear whether our application or API will support user authentication and registration. This means that it is unclear whether users will need to log in or register for an account in order to access the training program, or whether access will be granted to all users without authentication. If ShootSoft already has a user base, it may be possible to include their user ID to create new training sessions and track their progress. This would allow users to easily access their training history and track their progress over time, which can be helpful for both the user and the organization. However, we will need to determine whether user authentication and registration are necessary for the success of the training program or if access can be granted without these features. Ultimately, the decision will depend on the specific goals of the client and their needs.

- Training: The training table is an important component of a database that is used to store information about the training sessions or courses taken by users. Typically, this table would hold information about when the training took place, what discipline it was related to, who the user was that took the training, and other relevant details. In this case, the training table is related to both the user table and discipline table, meaning that it contains fields that are linked to those tables.

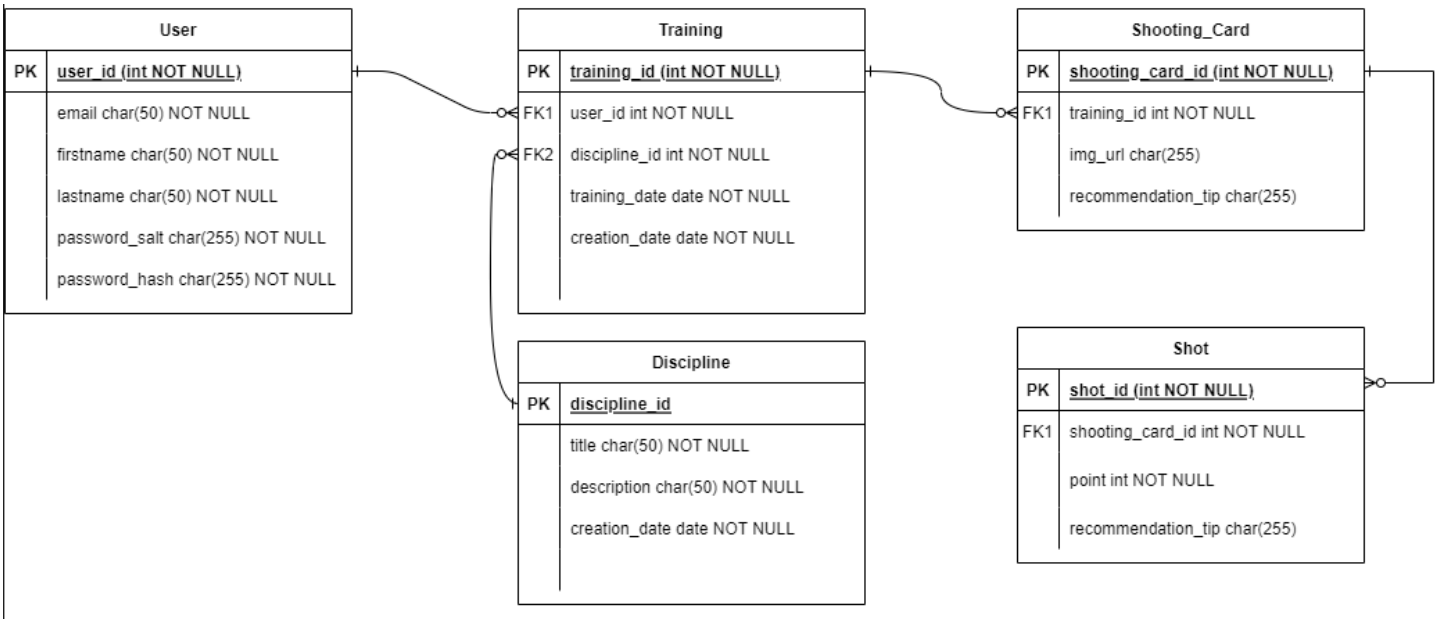
- Discipline: The discipline table is a database table that is used to store information about the disciplines. For instance, the pistol discipline and carbine discipline will be stored inside this database table. This will automatically translate to a choice set where the users can choose which type of discipline/training they want to upload.

- Card: The card table is used to store the shooting card information. In this case, the card table holds 4 fields that provide information about each card associated with a training session. Additionally, it may be possible to upload an image of the card to the table so that users can view their scores visually. However, it's important to note that including images in the table may require additional hard drive disk space, especially if there are many images associated with the training session. While adding images to the card table can be tricky due to storage concerns, it can be a valuable feature for users who want to visually track their progress in a training program. By providing

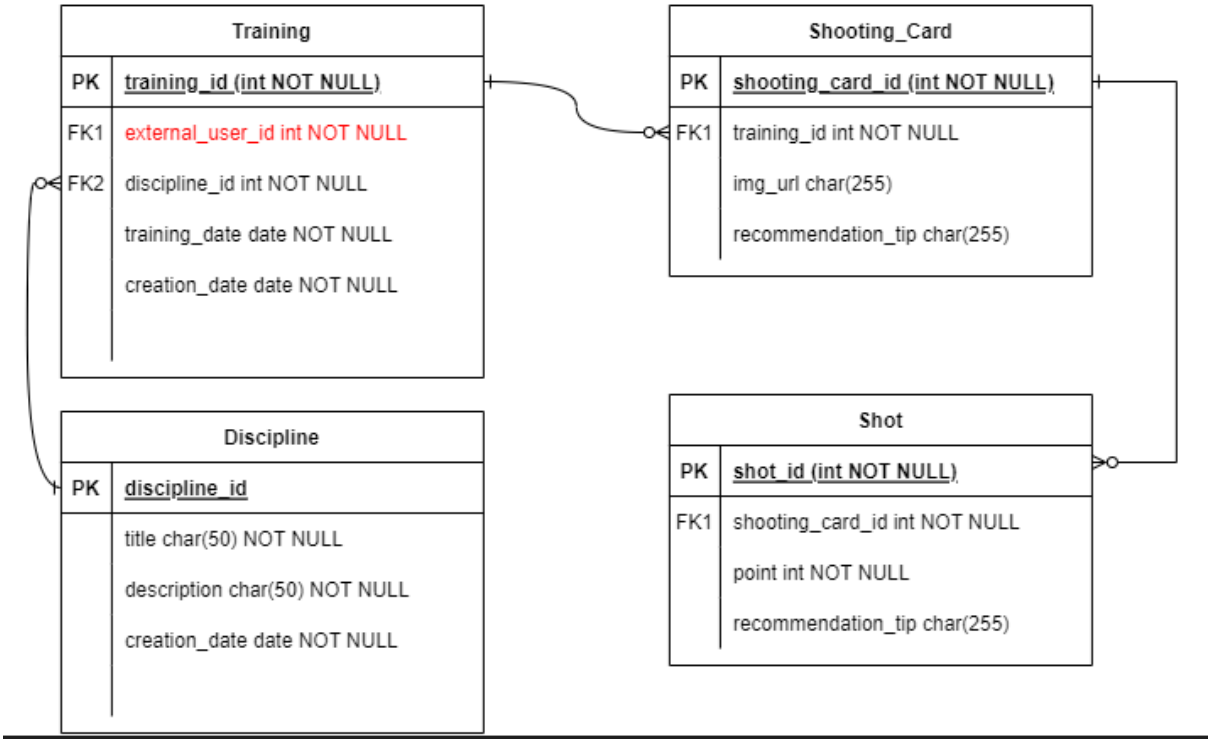
this additional context, users can better understand their strengths and weaknesses in a given discipline and work to improve their skills accordingly.

- Shot: The last table that we have is the shot table. This table is used to store information about the shots associated with a particular card in a training session. This table will hold information about the point that the user shot on the card, as well as any recommendations or tips associated with that particular shot. The shot table holds 4 fields that provide information about each shot associated with a card in a training session. By maintaining information about each shot in a separate table, users can more easily access information about their performance on a particular card and track their progress over time. Additionally, the inclusion of recommendations or tips associated with each shot can help users improve their skills and performance. However, we need to weigh the benefits of having a separate table versus the potential added complexity and maintenance of having multiple tables.

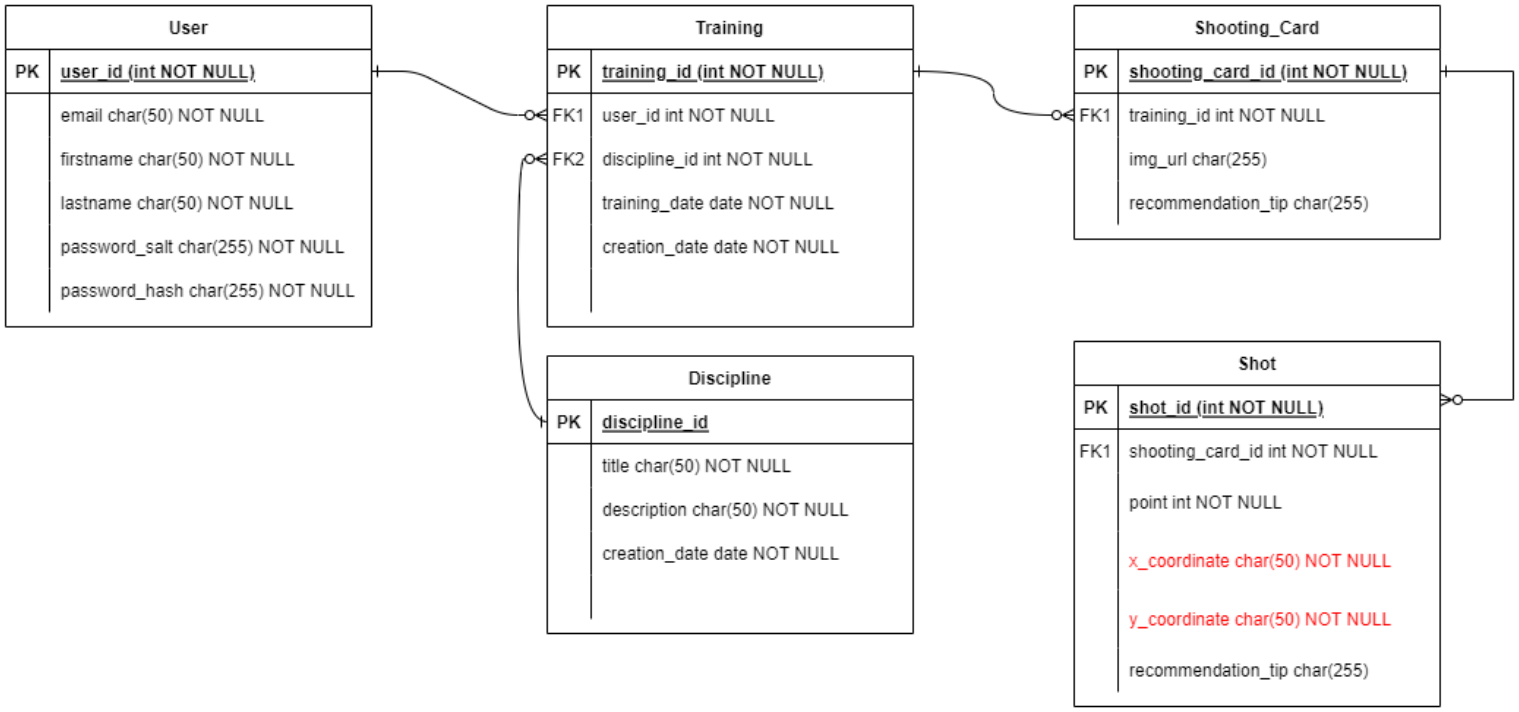
1. This first diagram is a representation of a bare-bones implementation of the ShootSoft bullet hole detection application. This version shows the minimal tables and fields required if we save user data ourselves. This means that users can register an account via our API or application.



2. This next ERD is used when ShootSoft already has its own user base and no separate user table is needed. In this case, the main difference between this ERD and the previous one is that there is no user table. Instead, the user IDs are stored in the training table under the field "external_user_id". This allows users to keep track of their scores and training history over time. By storing the user IDs in the training table, we can easily link training sessions with specific users and maintain a record of each user's training history. This is especially important when tracking progress and setting goals for improvement. With this approach, users can easily access their training history and view their scores over time.



3. This next ERD is used when we store users and have some additional features that are not present in the previous ERD. One of the additional features is the storage of the coordinates of a shot in order to save disk space. By storing the coordinates of a shot, we can recreate the target image using the coordinates of the bullet impact holes. This approach allows us to minimize the amount of disk space required to store target images, as we only need to store the coordinates of the bullet impact holes instead of the entire target image. This can be particularly helpful when storing numerous target images or when disk space is limited.



Functional Design

1. Introduction

1.1 Purpose of the document

Describe what a Functional Specification Document is and its intended purpose for the audience.

The purpose of this document is to outline the specific features, functionalities, and behavior of an application and/or systems. This functional specification document serves as a blueprint that guides the software development process. This is done by clarifying what the system should do and how it should do it. It also clarifies how the system should interact with users and other systems. This document is primarily for the developers, project managers, stakeholders, and anyone involved in the software development process. This functional specification document helps to establish a common understanding of the software requirements. This makes sure that all involved parties are on the same page, and that the development team has a clear direction for building the system

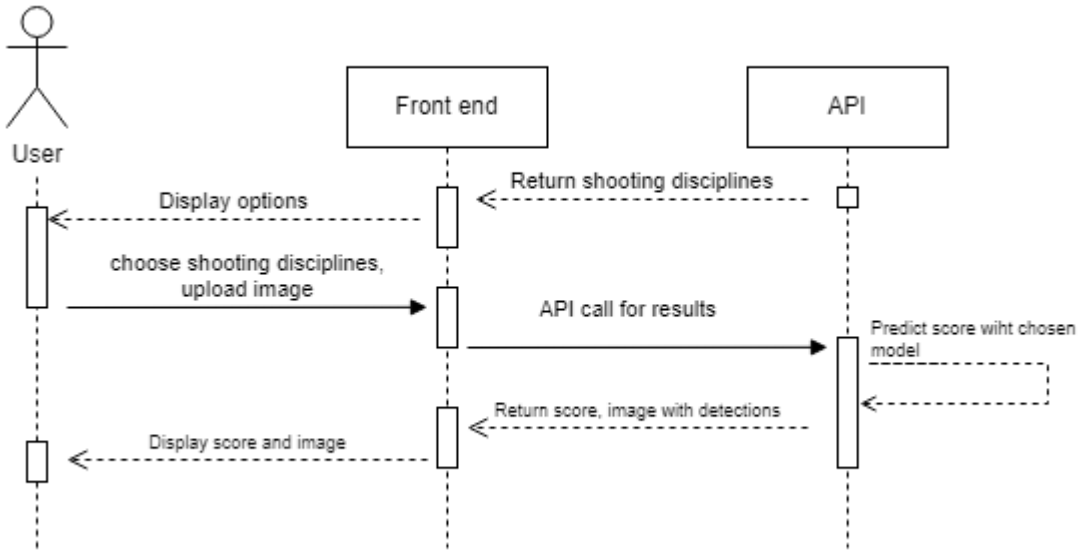
1.2 Scope of the document

The scope of this Functional Specification Document (FSD) is to define the requirements and specifications for the Shootsoft software application. It specifies the software's functions, features, user interface, as well as how the software interacts with users and other systems. The Functional Specification Document (FSD) includes information on data inputs and outputs, error handling, security, and other technical details required to build the software

2. System/ Solution Overview

2.1 Diagrams

Sequence diagram

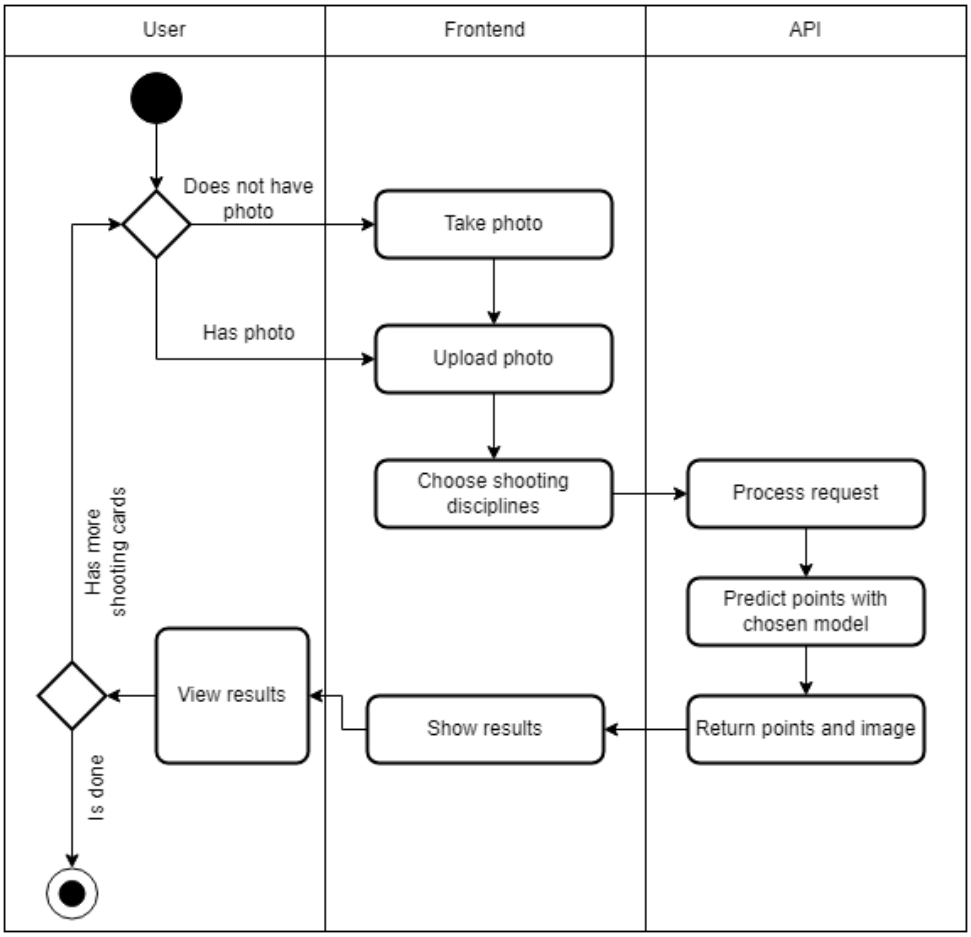


In this sequence diagram, we have one actor: the user and the components the frontend, and the API. The API starts the process by sending the shooting disciplines to the frontend. Once the frontend receives the disciplines, the user can then upload their images and select the shooting discipline where they want to get the score from.

Next, the frontend sends a request to the API, passing along the images uploaded by the user and the selected shooting discipline. The API then uses the YOLOv8 model to predict the scores of the images and returns the results back to the frontend.

Finally, the frontend receives the predicted scores from the API and displays the images to the user, along with the corresponding scores. The user can then view their scores.

Swimlane activity diagram



The swimming lane activity diagram is an important component of the functional design document. It helps to provide a clear overview of the process flow and the components involved, allowing for a better understanding of the system's functionality.

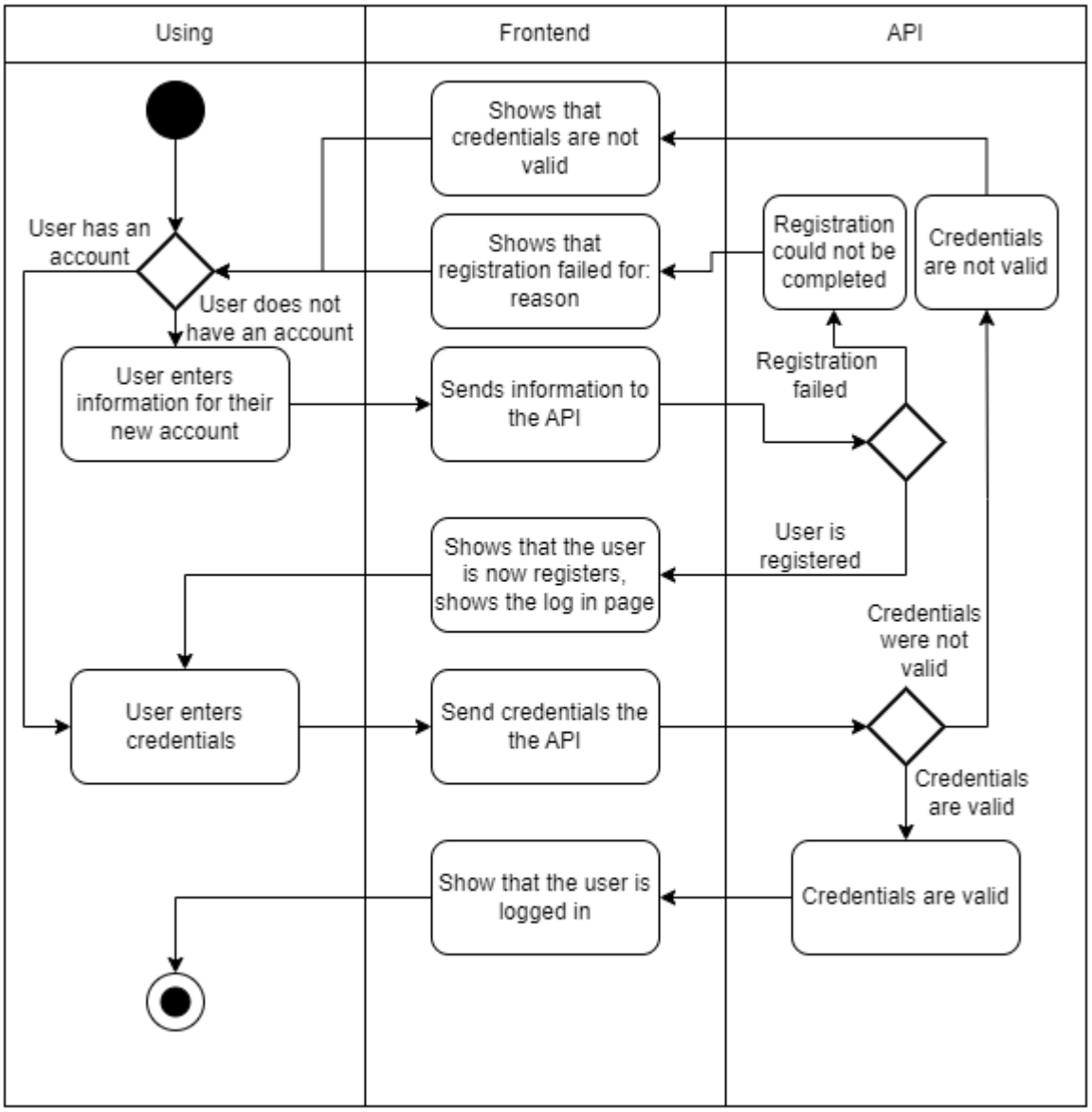
In our specific use case, the diagram is divided into three different lanes: the user lane, frontend lane, and API lane. The user lane represents the choices and action of a potential user, the frontend lane shows the responsibilities of the frontend and the API lane shows the responsibilities en functionalities of the API.

The user then has two options, which are both represented in the frontend lane. The first option involves uploading a photo, while the second option involves taking a photo first, which is then followed by uploading it.

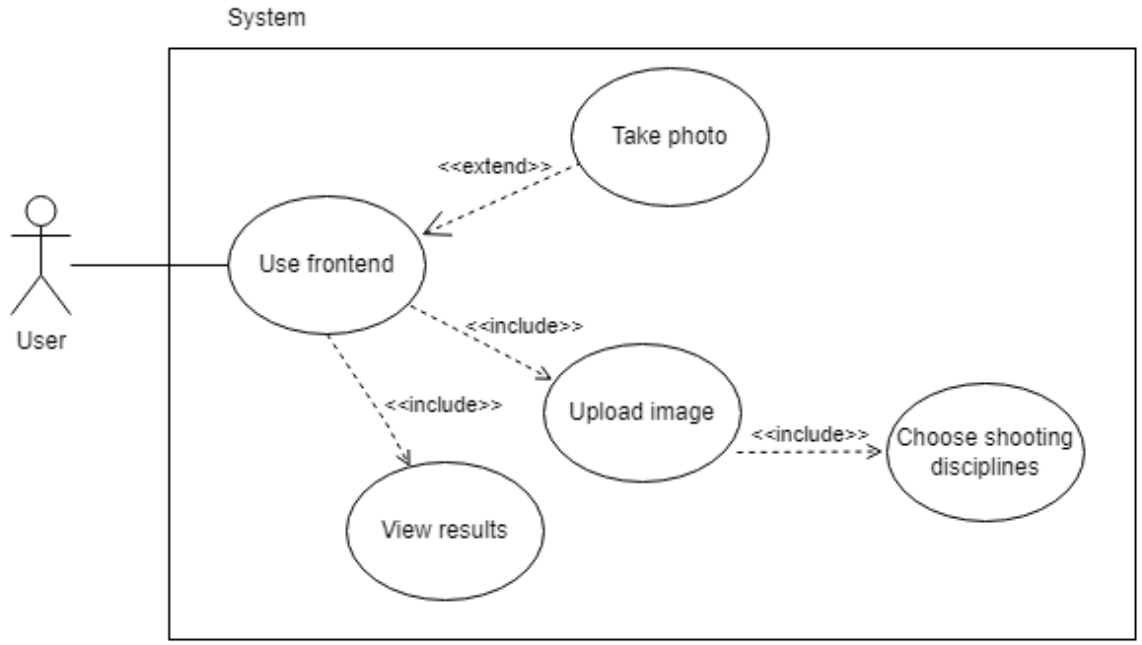
After the photo is uploaded, the user will select the shooting discipline. The frontend will then send a request to the API, where the request will be processed. In the API, the YOLOv8 model will predict the score and the points will be extracted, which will be returned to the frontend along with the image.

The frontend will then show the results to the user, who will have two options: either to upload images again or to stop.

Optional:
As an optional feature, this diagram could contain some step regarding logging in or registering before uploading images. It would then contain steps for all lanes. The log in process could be something similar to this:



Use case diagram



A Use Case is a description of the interaction between an actor and a system in order to achieve a certain goal. In this case, the actor is the user and the goal is to train their shooting skills and view the results.

Identifying a Use Case in the functional design process is important because it helps define the system requirements. By understanding which actors are involved in the system and what actions they will perform, the functionalities of the system can be established. This helps in designing the system, developing the software, and testing the system to ensure that it meets the requirements.

In this Use Case, there is one actor, the user. The user has access to the website/frontend, where they have the ability to take photos. If the user wants to upload photos, they must do so before choosing a shooting discipline for their training photos.

Once the user has selected a shooting discipline and uploaded their photos, the system will call the API with the YOLOv8 model to get the results of their training. The training results are displayed to the user, along with the corresponding points.

Optional:
In addition to the steps already described in the Use Case, there are optional functionalities that are not included in the use case. Think of logging in and registering a new user. This provides the ability to keep track of the user's score, allowing them to view their score history and track their progress over time. This can be useful in monitoring the improvement of their shooting skills and identifying any weaknesses.

2.2 System Actors

User/Role	Example	Frequency of Use	Security/Access, Features Used
User	Someone who is a member of a shooting club	Someone who is a member of a shooting club goes to the shooting club 1 or more times a week and does 1 or more training sessions there. Each training can be submitted to count the points.	- Taking a picture. - Uploading a photo. - Choosing a discipline, - View results.

Graphical Design

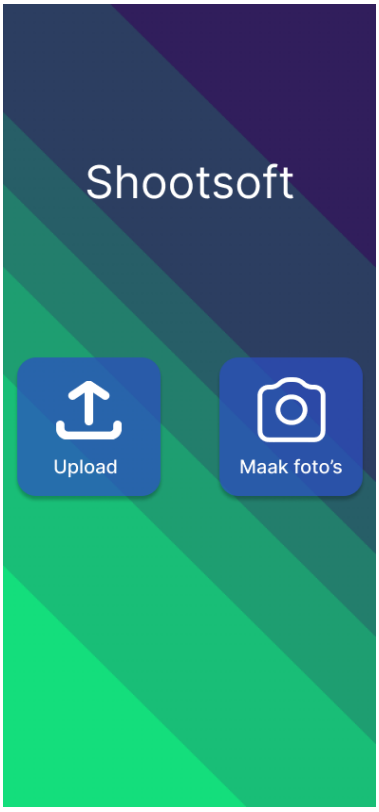
This graphical design document (GDD) is a detailed document that outlines the visual/graphical elements of the software application and other connected system. The graphical design document (GDD) will include information about the user interface (UI) design, typography, layout, color scheme, and other graphical assets. The purpose of the graphical design document (GDD) is to provide a clear plan for the design team to follow during the development process. This will make sure that the elements of the website / desktop application or mobile application are consistent, aesthetically pleasing, and user-friendly. This document is for the design team, which includes people who design how the software looks and works, as well as the people in charge of the project. They will use this document to make sure the software looks and works how they want it to. It's also a way for the design team to talk to the people who make the software, so everyone knows what the software is supposed to look like and how it should work.

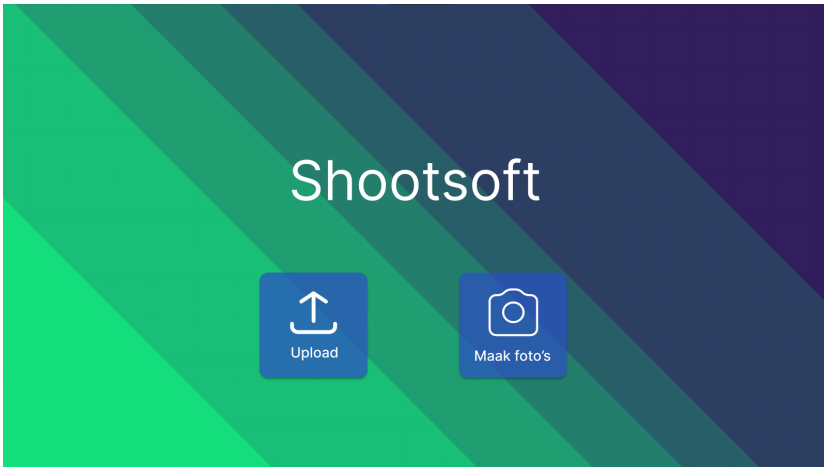
1. Front end design

For our proposal of the front-end design we wanted to make it easy to use and with as few steps as we possibly can. The front-end application should strive to be either as fast as counting the total number of points by hand or faster. This is why the design is mobile first and as minimalistic as possible. The design currently only features the must haves in this project. But there is the possibility of expanding upon any area like for instance user management or automatically increasing the dataset with scanned images. The design is made in Figma and can be seen with the following link:

https://www.figma.com/file/DuXVpsDaH2HAVYQy1OzMTR/Shootsoft-application-design?node-id=17%3A216&t=4wLaRpbpozN7eatr-1

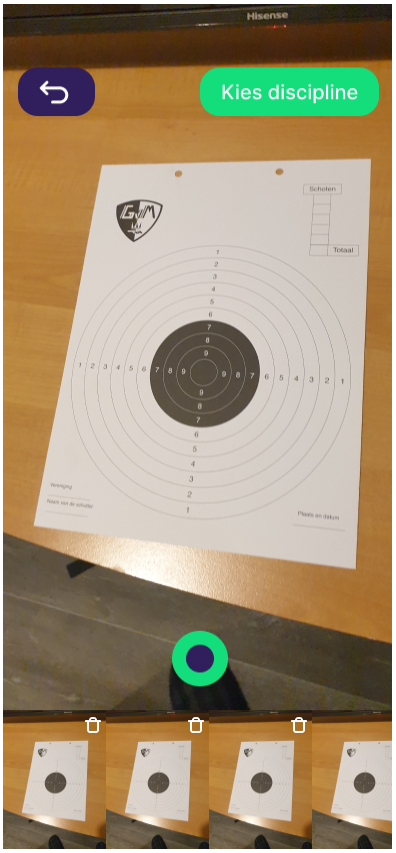
Main menu

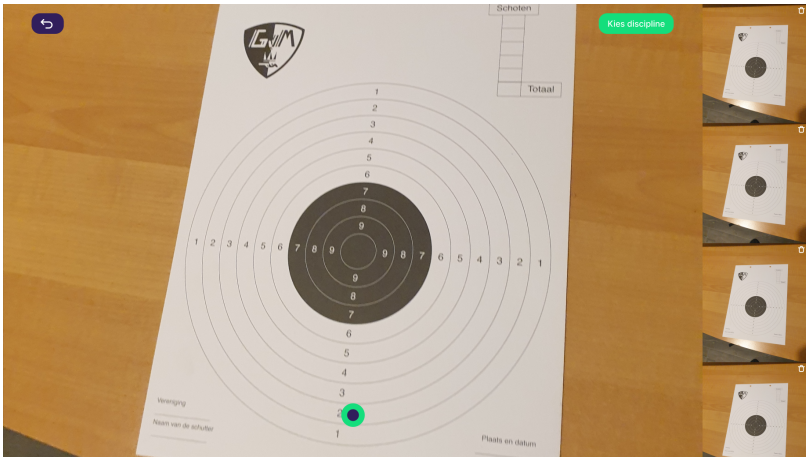




The first image is what the homepage of the shootsoft application on mobile phone will look like. You will have 2 options. You can either upload images from your storage, depending on your device. Or take images with your camera. The second image is what the application will look like on the desktop. The design is very similar but does still allow for both options. Taking images on a desktop will obviously require a webcam.

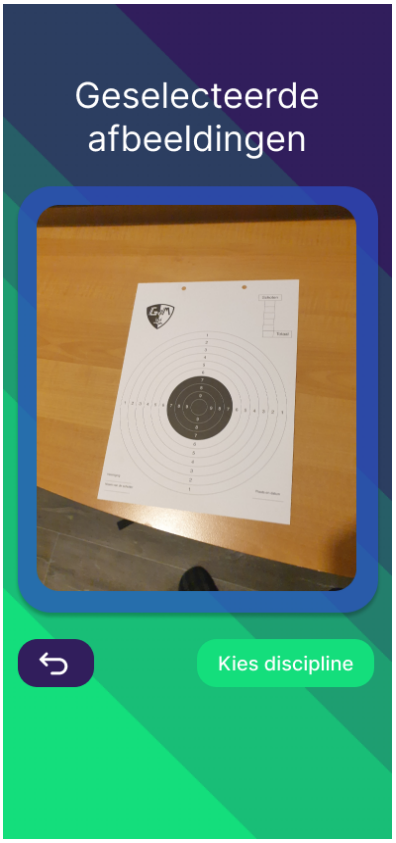
Taking pictures

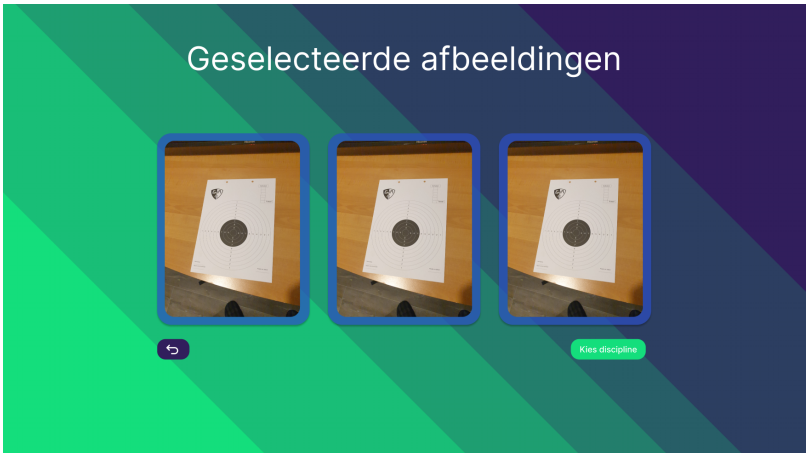




In the front-end application you will be able to take multiple images and then have the option to detect the scores. This will save time since you don't have to go through the entire process repeatedly. In both the desktop and mobile version you will be able to see the images that you took and delete these incase necessary. At the top you will have two options. The green button will allow you to choose the shooting discipline and the purple button will be the back button. The big dot in the bottom center is to take images.

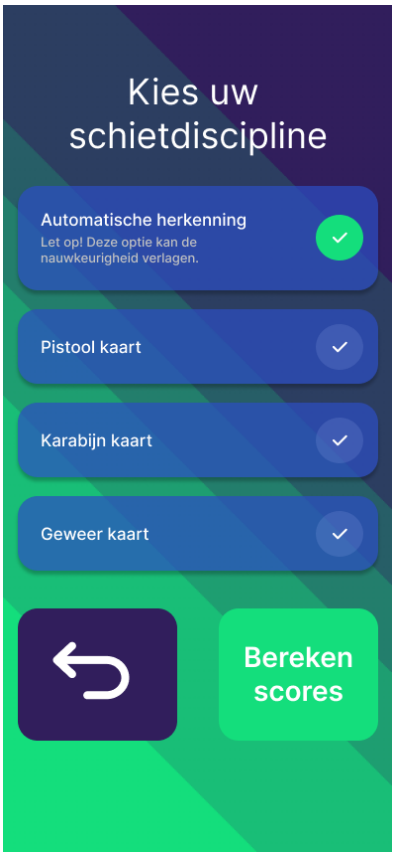
Selected images

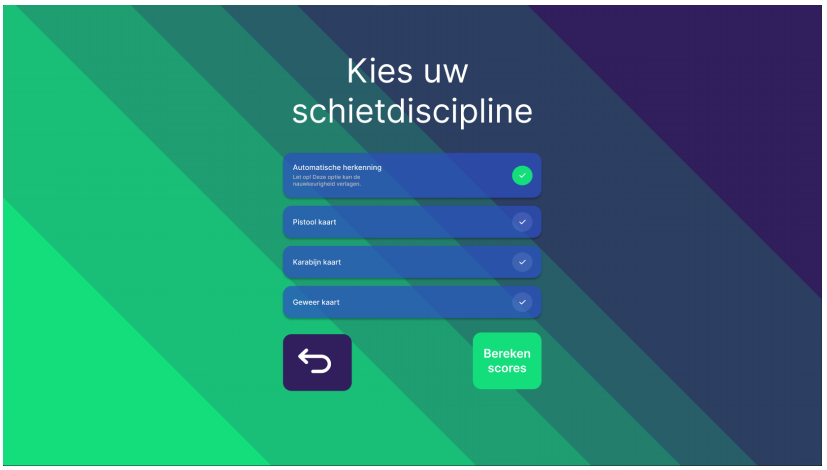




If you upload images through the file upload functionality you will be able to have an overview of the images. At the bottom there will be two options. You could either go back to the main menu or to the choose discipline screen.

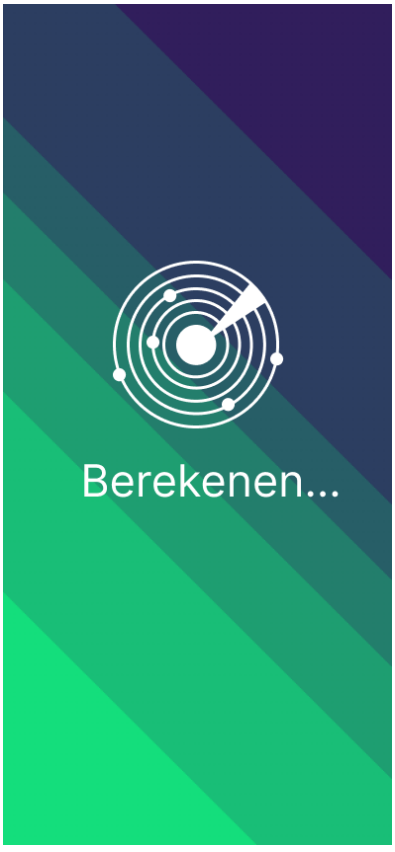
Selecting shooting discipline

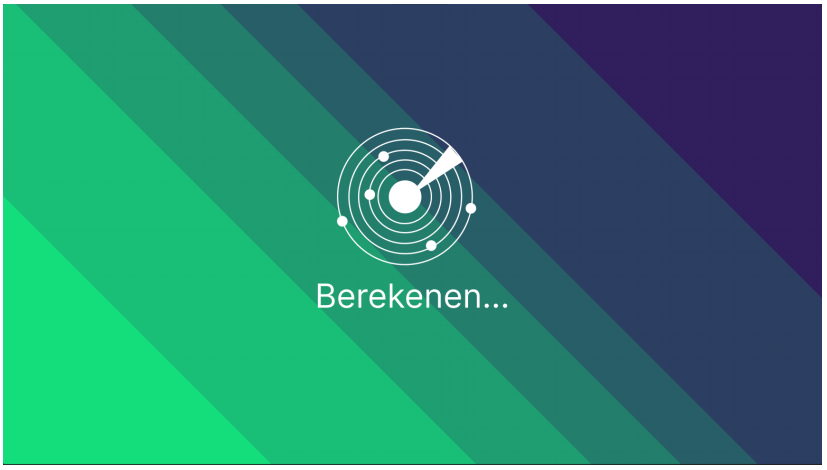




Selecting the shooting discipline will consist of a number of options. The first option that will be auto selected will be automatic recognition. This is important for ease of use and allows for minimal effort of the user. The other options will be the different types of shooting disciplines. At the bottom there will be two options. A back button which will take you to the taking pictures or selected images screen depending on your option at the start. The big green button will be to calculate the scores.

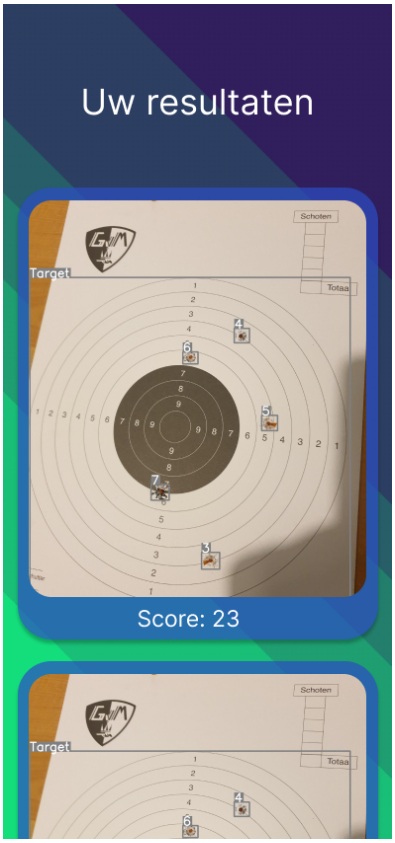
Calculation screen





The calculation screen will be shown when the request is sent to the API to calculate all of the images. Since this could take 1 or 2 seconds. This screen needs to be interesting enough to stare at for 2 seconds. The actual application will have this icon scanning like an actual radar.

Results screen



The results screen will show all of the images with the scores underneath. At the bottom there will be the total score. On mobile a single card will take up the entire width of the application. On the desktop however the cards will be stacked three wide for a better overview.

Appendix

Figma design

The figma design can be viewed with the following link:
https://www.figma.com/file/DuXVpsDaH2HAVYQy1OzMTR/Shootsoft-application-design?node-id=17%3A216&t=4wLaRpbpozN7eatr-1

Division of tasks

The division of tasks can be found in the following git repository:
https://github.com/neburpoots/shootsoft/tree/main/Contributions

Roboflow

The link to the roboflow containing the original dataset can be found here:
https://app.roboflow.com/project-bat/project-oqaxk/overview

Swagger documentation

The link to the roboflow containing the original dataset can be found here:
https://github.com/neburpoots/shootsoft/blob/main/swagger.json