SUPPORTING THE BOEKELO MILITARY GRAND JURY

An application for supporting the Grand Jury of the Boekelo Military cross-country

by Aimé Ntagengerwa

HBO-IT Software Engineering at Saxion University of Applied Sciences

SAXION UNIVERSITY OF APPLIED SCIENCES

Ntagengerwa, Aimé
Version 1.0

Table of content

1. Abstract

This document describes the graduation project I carried out at the AMI research group at Saxion Hogeschool in Enschede during the period of 18-11-2019 through 22-4-2020. The goal is to prove that I possess the ability to design and create professional-grade software in a professional setting and in the process earn the right to call myself a Software Engineer.

This report covers the full process of taking the idea of solving a business case introducing a piece of software. This was done in cooperation with end users and experts on the topic. The project is related to the working practices of the cross-country jury of the Military in Boekelo.

The cross-country is an annual horse and jockey competition where the goal for participants is to cross many obstacles over a six-kilometer-long track and reach the finish line with as little penalty points as possible. Penalty points are awarded when an obstacle is not cleared correctly, or when the time it took a participant to reach the finish line is over a certain threshold.

Currently, the jury overseeing this event consists of three top experts – the Grand Jury – and a couple of people at each obstacle forming the many field juries. The Grand Jury is the only body to award penalties, but to do so, they rely on the observations of the field juries which are communicated through hand-held radios. Even though their help is very much appreciated, the existence of the, often voluntary, field juries is out of practical necessity. It is the only way the Grand Jury has access to observations made about the performance of participants.

This is the problem which this project aims to solve; the Grand Jury must rely on the observations of others and cannot “take a closer look” at anything. Once the horse has passed an obstacle it’s gone, and if the field jury missed it (or is unsure of what they saw in a flash) there is no way to determine what has actually happened.

The software solution that enables this is referred to as “the Jury Application”. In essence, it allows for viewing the camera live streams and play back recordings of this. Having with their own eyes seen the video of a participant crossing an obstacle, the jury can come to a well-informed verdict. These verdicts are broadcasted to all users of the EIS and stored in a database for later reference.

2. Terminology

Term	Description
Grand Jury	The people responsible for taking verdicts about participants and race control decisions. The members of this jury are simply referred to as “jury members”.
Participant	A horse-jockey combination which takes part in the cross country of the Military.
To assign a jury member to a participant	When a jury member is assigned to a participant, he or she becomes responsible for monitoring and judging that participant from the time it starts the race, until the time it reaches the finish line. All decisions and rulings about this participant must be made by this jury member.
Available participant	A participant who is about to start the race *and* has at that moment not yet been assigned to a jury member, and has not yet started the race.
EIS	“Event Information System”. The EIS aims to enhance the experience of the visiting audience of the Military, as well as that of the organization members in charge of governing the event. It does so by incorporating many subsystems to provide a set of features such as a live video stream of the event, a horse monitoring device, and a jury application.
EIS node, Camera node	A device equipped with a camera and networking infrastructure, allowing for the automatic filming and live streaming of a participant at an obstacle.
Video clip	A video recording of a participant made by an EIS node overviewing an obstacle.
Trick-play	The act of operating on a video clip being displayed. Provides a jury member with fast forward, rewind, play, pause and skip-to-frame operations.
Tactical overview	A user interface component which displays a map of the cross-country track with participant positions marked on it, as well as the heart rate of the currently assigned participant. When a video is playing, this component is synchronized with that video.
CEL	“Cumulative exhausting level”; the output of an algorithm taking as an input heartrate values. Used for determining the risk to the wellbeing of a horse due to exhaustion. A high CEL is harmful to the wellbeing of a horse.
Verdict	An official decision about a participant made by a judge, based on observations. Such decisions are made every time a participant passes an obstacle.
Horse monitoring data (HM data)	The position- and heartrate measurements of a horse.
Race control decision	A decision issued in case of a calamity. The two decisions are halting the race or resuming it.

Table 1: Terminology

3. Introduction

Every year, the highlight of the Military event – the ever popular cross country – draws thousands of people to the small town of Boekelo. This event sports a 6 kilometer long, obstacle littered track for competing horse-jockey combinations to complete, and every year the visitors to the festival walk alongside it through the magnificent forests surrounding Boekelo. Around one hundred horse-jockey combinations compete in an effort to finish the race with as little penalty points as they can. These penalty points are awarded by a jury, and they base their decision on what is being observed during the race. If the time limit for reaching the finish line is exceeded, a combination will be dealt penalties. Likewise, if a combination does not successfully clear an obstacle, they are awarded penalty points.

This event is at the moment a very “analogue” one. However, efforts to change this are being made, and the accumulation of these digital systems is what is known as the Event Information System (EIS). The EIS aims to enhance the experience of the audience visiting the Military, as well as that of the organization members in charge of governing the event. It does so by creating a high-bandwidth ad-hoc network over which many services are provided. These services include cameras at each obstacle streaming live video of participants, a user application for visitors to view these live streams and the performance of their favorite participants, and an application supporting the work of the Grand Jury. This project focusses on providing services to the Grand Jury.

3.1. Problem statement

One challenging aspect of the work of the Grand Jury is the physical distances between them and the place where the participants perform their feats. To overcome this, they have people in place at each obstacle who will describe what they observe via hand-held radios. This can lead to misunderstandings and unfortunately leaves no evidence of what has actually happened. For a participant, it is therefore very difficult to challenge the jury’s verdict.

3.2. Solution

The Military in Boekelo is the proving ground of the Event Information System (EIS) which is being developed by the AMI research group. This system allows for a high-bandwidth ad-hoc network to be deployed anywhere. The EIS includes cameras which broadcast live streams of the cross-country, a user application for visitors of the event and a jury application. The latter is what this project focusses on; improving the practical operability of the tasks carried out by the jury of this large-scale event.

With the EIS is in place, jury members are able to see in real time what is happening when a participant approaches an obstacle. They see a live stream of the action every time a combination approaches an obstacle, together with a tactical overview of the combination’s position and the horse’s health. With this information made available, they can take decisions regarding the awarding of penalty points or the disqualification of a participant out of concern for the horse’s wellbeing. If needed, these jury members can look back at a recording of what has passed at the obstacle before reaching their verdict, or to justify their decision.

4. System context

The jury application exists as a sub-system within the larger Event Information System (EIS). The EIS is a distributed system. To function properly, the jury application relies on interfaces provided by other sub-systems. It will then use incoming data flows to display information to a user. When a user interacts with the system, this generates information. The jury application makes this information available to other sub-systems.

The benefit of this architecture is that the many sub-systems which the EIS comprises can be easily developed independently of each other. Working with well-defined interfaces, the development of jury application can be completed even before some other sub-systems are available. The architecture also allows for easy maintenance and extension, by providing a backbone to which systems could be added, and over which these systems can exchange information.

4.1. Event Information System

The EIS is built up of multiple systems working together. These sub-systems must be able to communicate in order to provide the services that make the EIS. They all have their own role to play, and the jury application only makes up one section of it.

One interesting thing to note is that the EIS is not connected to the internet. This means that all information in the system has either been packaged within its sub-systems, or generated on-site. Figure 1 shows the flow of information in the EIS.

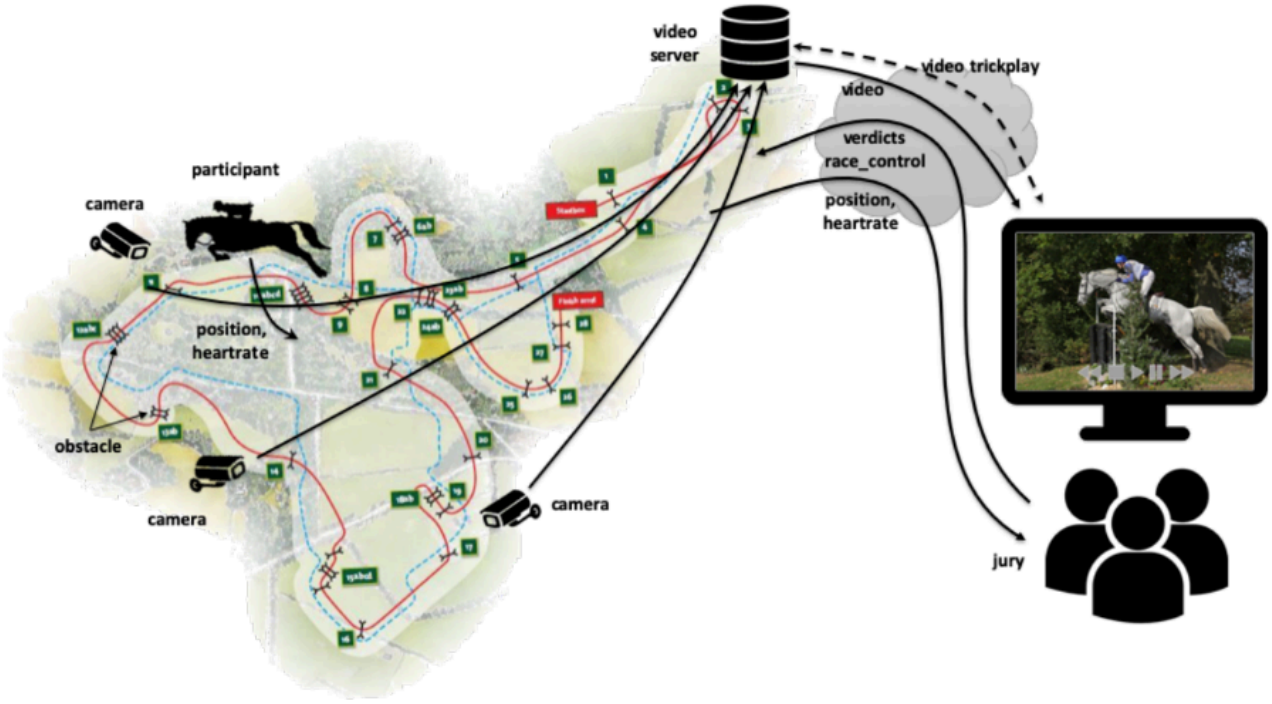


Figure 1: Sub-systems of the EIS

4.1.1. Message broker

It's not depicted in Figure 1, but all information flows between sub-systems (except the live video stream) go via an MQTT message broker. This is in itself a sub-system which is dedicated to broker these information flows.

4.1.2. Horse monitor

The horse monitoring sub-system is a device which is attached to a participant. It tracks their position and heart rate, and broadcasts this to the EIS. It does this through LoRa radio communication, but for this level of abstraction it's accurate enough to say that this information makes its way to the message broker *somehow*. This information is then available to the rest of the sub-systems.

4.1.3. Visitor application

The visitor application allows for visitors of the cross-country event to stay informed about the progress of the race, including the displaying of live streams of participants at obstacles, and the verdicts reached by the Grand Jury.

4.1.4. Camera node

Live video streams are provided by the camera nodes located at obstacles. Their core objective is to film the performance of participants at their obstacle, but they are also equipped with the networking infrastructure which facilitates all communication across the entire EIS. The live video streams follow the fire-and-forget principle; camera nodes do not record or store any data.

4.1.5. Jury application

The jury application is what this project is all about. This comprises not only the user interface for jury members, but also all other components and services which enable this. It includes;

- the video server, which records, stores and serves live video streams;
- the event archive, which stores all information generated by horse monitors, jury members, and non-video information published by camera nodes, and lastly;
- the user interface with which the Grand Jury interacts.

4.2. External interfaces

The external interfaces of the jury application define how it interacts with the EIS.

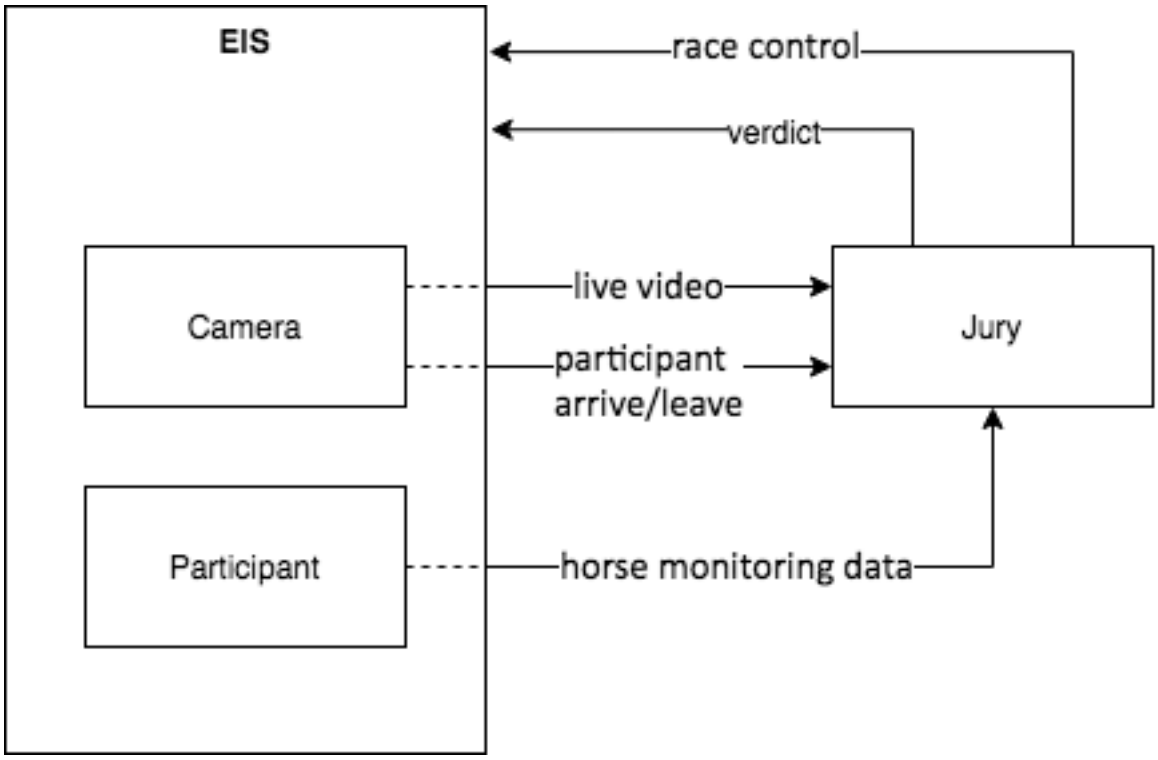


Figure 2: External interfaces

As shown in Figure 2, the jury application exposes two information flows, and relies on three others as inputs, coming from two different sources.

The interface between the jury application component and the camera component is through a multicast group. The jury application joins a multicast group on the network level and receives live video. This allows the camera component to send one live stream to many receivers and reduces the load on the network.

All other interfaces work by a component publishing information to a message broker. This broker will then make it available to any subscribers. In Figure 2, this broker has been omitted to show only the information flows. The participant component only publishes information. The jury application component both a publishes- and subscribes to information.

The jury application must operate in real time, but must also be able to display recorded video streams and HM data. For this purpose, the jury application continuously stores incoming information. During the playback of a recording, it uses this stored data.

5. The product

This chapter describes what the result of the project is from a user's perspective before diving into specifics of the system. This will help placing things into perspective later.

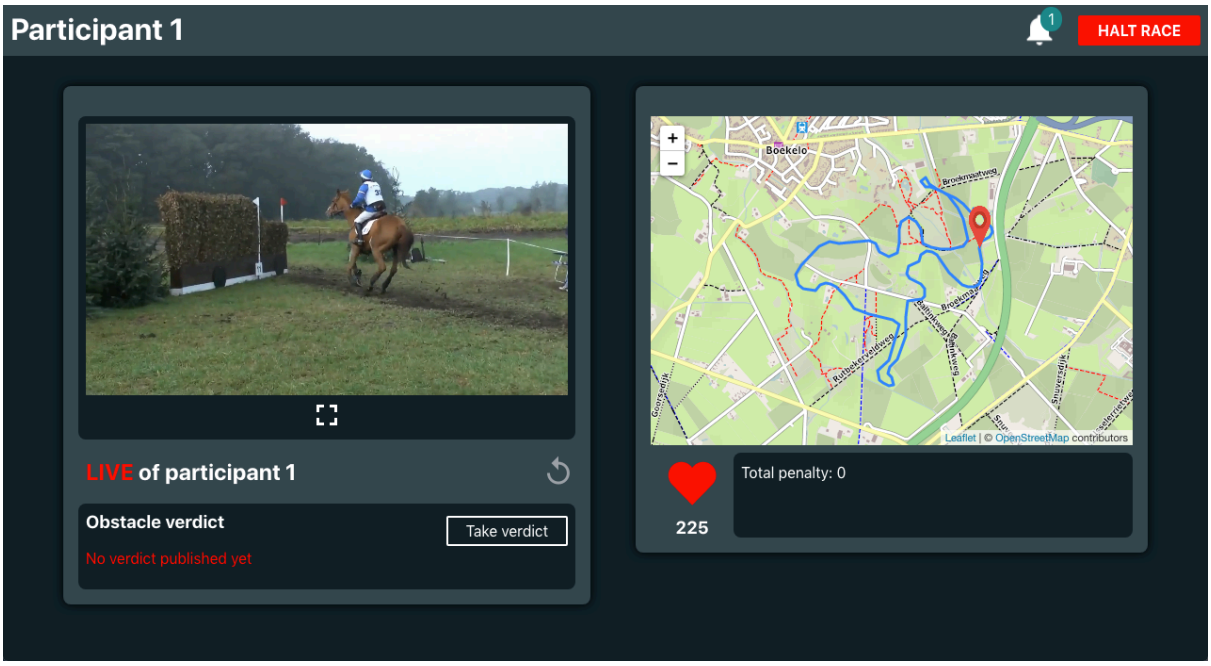


Figure 3: Screenshot of live view

Figure 3 shows what a jury member sees when following a participant live. On the left-hand side, the screenshot shows a live video stream from a camera node. When a participant approaches an obstacle, a live stream is started. When this participant leaves the obstacle, there is no video being streamed by the camera, and the application shows a placeholder. When a participant again approaches an obstacle, the application will automatically switch to it.

Below the video player, the system offers jury members the option to publish a verdict of the performance they are seeing in the video (see Figure 5).

On the right, the application shows a tactical overview of the participant's position and heart rate, and the total amount of penalty points the participant has received. These values are synchronized with the live video. This means that the tactical overview never shows information which does not match the video being played. This makes the tactical overview match the latency of the live video, but avoids confusion with the user.

The header of the application shows which participant the user is assigned to, and provides options which are not exclusive to the live view. The bell icon allows a jury member to go to the playback view of the assigned participant's performance at obstacles which have not been given a verdict yet (pending verdicts, see Figure 4). The big red button on the far right allows a jury member to publish a race control decision (Figure 6). This race control decision is always available for safety reasons.

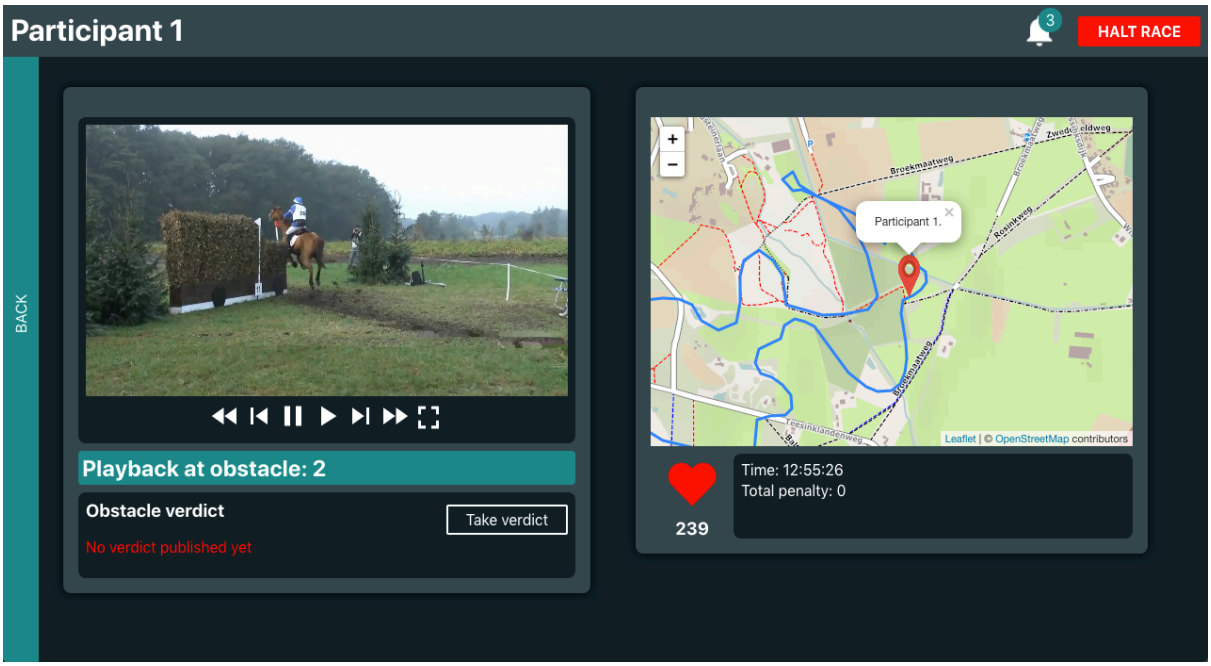


Figure 4: Screenshot of playback view

Figure 4 shows the playback view. This is the screen that a jury member goes to when they want to look back at a video recording to come to a more well-informed verdict. This is assisted by the trick-play control options under the video. This allows a jury member to play or rewind the video at half the speed, or to go back and forth through each frame of the video. Pausing the video allows a jury member to take their time with looking at crucial moments of the participant's performance. Since the horses in the cross-country are moving very fast, these trick-play options provide a lot of value to the Grand Jury.

The rest of the screen looks almost exactly the same as the live view. Here too, the tactical overview is synchronized with the video that is being played. This means that when the video is rewound at half the normal speed, the tactical overview will show the participant's position going "backward" too.

On the outer left side of the screen, there is a button which takes the jury member back to the live view when they click it.

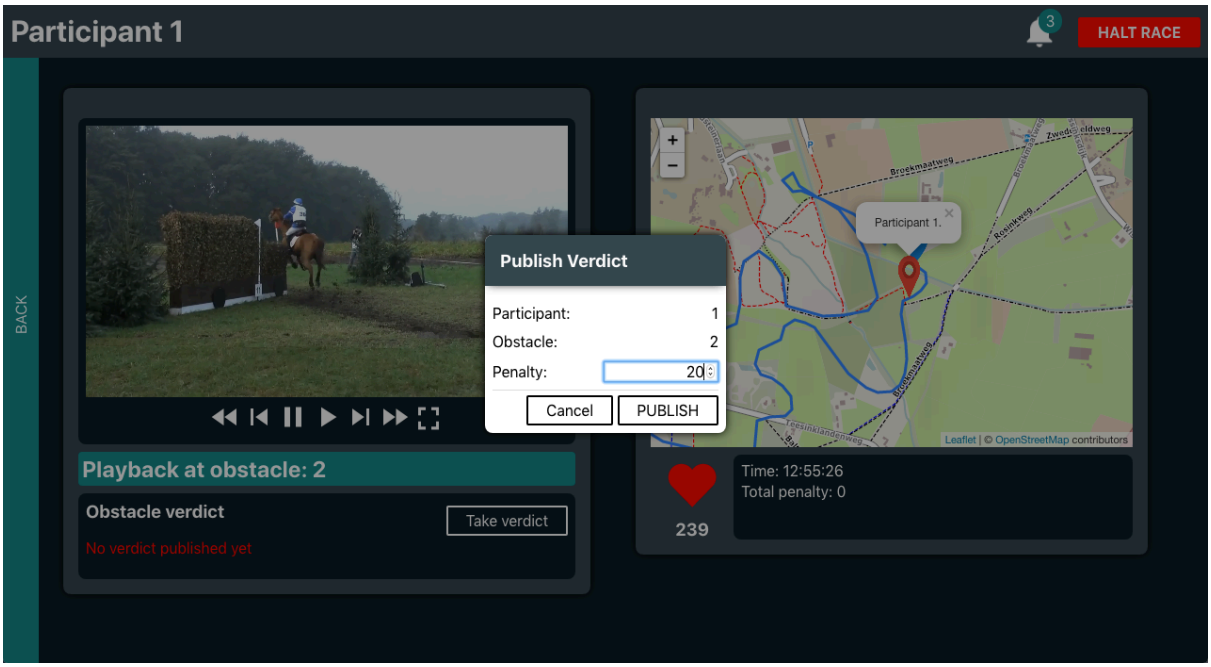


Figure 5: Screenshot of publishing a verdict

Figure 5 shows how a jury member would go about taking a verdict. The form that is depicted has three fields. Out of these, the participant and obstacle ID are automatically provided, and relate to the performance which the jury member is viewing at that moment. The jury member fills in an amount of points within the interval of zero to a hundred, inclusive, and publishes this verdict to the whole of the EIS. Having done so, the form is disabled and closes.

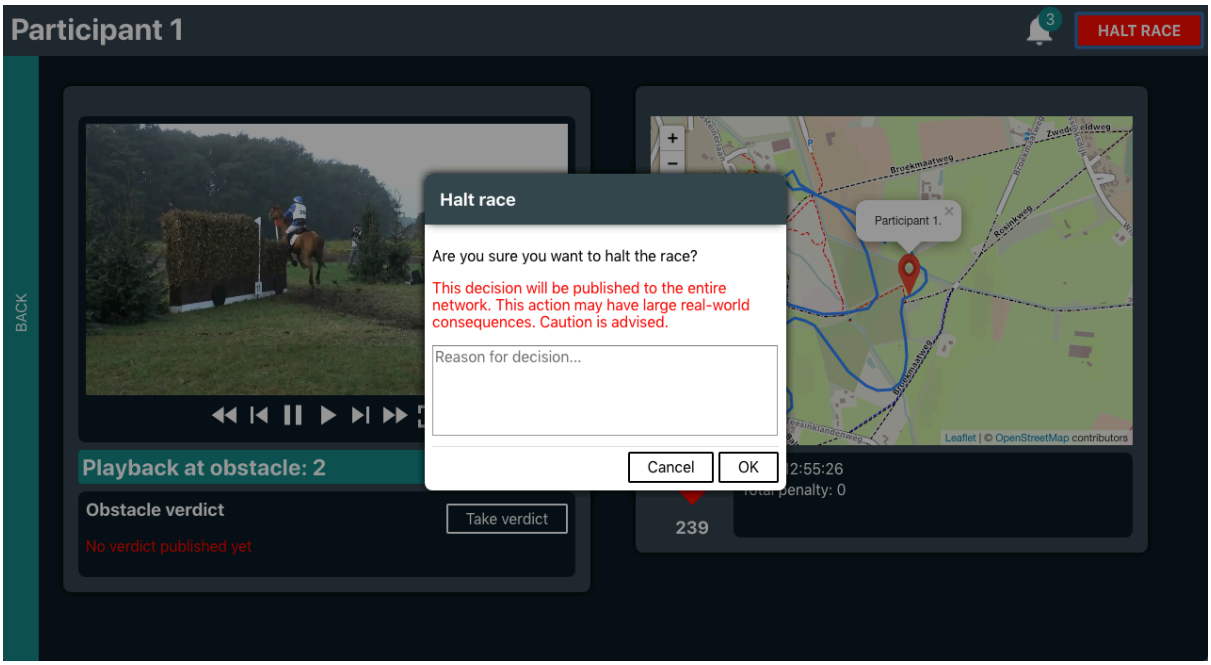


Figure 6: Screenshot of publish race control decision

Figure 6 shows how a jury member publishes a race control decision. A jury member can do two things; halt or resume the race. Which of these options is displayed to the user depends on the current state of the race. When the race is currently halted, the option to resume it is shown. When the race is in progress, the option to halt it is shown.

Race control decisions are published to the EIS. Because they affect the real-world event in a major way, a message of caution is included in the form. The description box prompts the jury member to provide a description of what has led to making this decision. This description is optional.

6. Functional design

The functional design describes what is agreed with the product owner on what the system will do, and how it will behave. This agreement is important to check that the system will be useful to the client, and that it tends to the needs they have. The other way in which the information described below is useful, is by providing an overview of the externally observable aspects of the system as a whole to which other designs can be linked.

6.1. Use cases

The functionality of the system is described in use cases. These describe the ways in which a user can interact with the system.

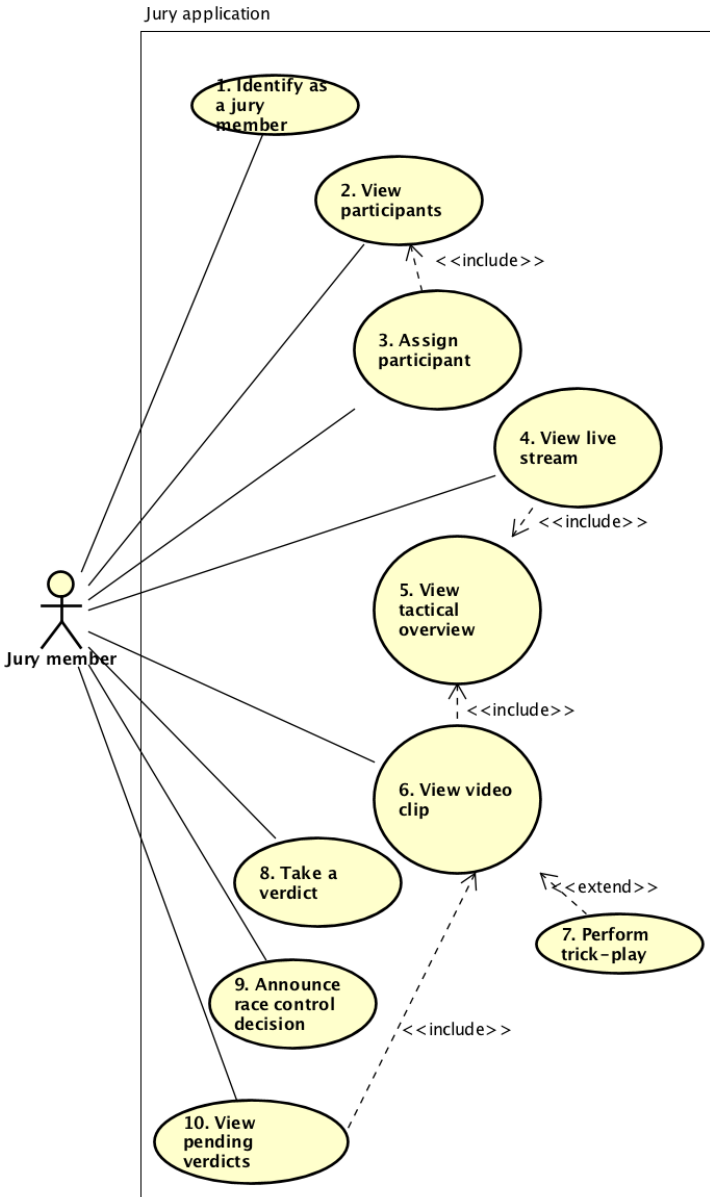


Figure 7: Use case diagram

Figure 7 shows all the interactions a user can have with the system. These interactions are further elaborated upon in use case descriptions. An example of such descriptions can be found below. For the full list of use case descriptions, please refer to Appendix A.1.

Identifier	UC.8
Name	Take a verdict
Summary	A jury member takes a verdict based on the observations of a participant.
Rationale	Having observed the participant, the jury member is well-informed and confident in taking a verdict. This decision can later be reviewed and compared to a video clip in case of a dispute.
Actors	Jury member
Preconditions	- A participant is assigned to the jury member. - The participant has crossed an obstacle. - The participant has not received judgement for crossing the obstacle.
Scenario	1. The jury member observes the performance of the participant at an obstacle. 2. The jury member enters their verdict into the system. 3. The system informs the jury member that the judgement process is completed.
Alternatives	1a. The jury member takes a verdict based on CEL observations. i. Proceed to step 2. 3a. The system informs the jury member that insufficient details have been entered. i. Return to step 2.
Postconditions	- The verdict is published to the EIS.

A summary of each use case can be found below in Table 2.

Identifier	Summary
UC.1	Fill in information which uniquely links a user to a jury member.
UC.2	View which participants are taking part in the event, and what their status is.
UC.3	Assign a participant to a jury member.
UC.4	View the live video footage of a participant crossing an obstacle.
UC.5	View a tactical overview of horse monitoring data, showing a participant's position on a map and the heartrate and CEL of a horse.
UC.6	View a recorded video clip.
UC.7	Perform trick-play on a video clip.
UC.8	A jury member takes a verdict based on the observations of a participant.
UC.9	Announce a jury member's decision regarding the continuation of the race.
UC.10	View which obstacles have been crossed by an obstacle, but have not yet been subject to a verdict.

Table 2: Use case summaries

The use cases relate to the set of requirements listed in Chapter 7: Requirements and constraints. All requirements are covered by the ten use cases defined above. The exact relationship between use cases and requirements can be found in Appendix A.2.

At the time of defining use cases and requirements, mockups were created which are based on the use cases. These mockups have played a big role in communicating the system's use cases to stakeholders. The user interface of the system is based on the mockups. One mockup is shown in Figure 8 as an example. The rest of the mockups are included in Appendix B.

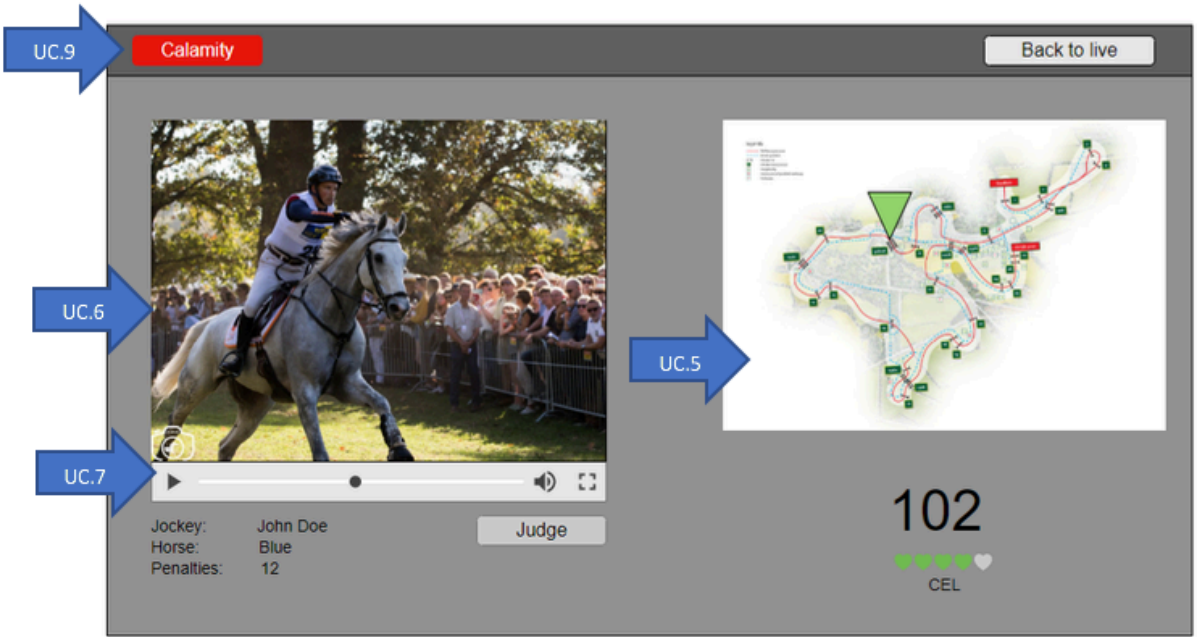


Figure 8: Example mockup screen

7. Requirements and constraints

The following requirements are determined and prioritized in agreement with the client, using the MoSCow notation.

7.1. Must

r1. Jury members must identify themselves to the system before they take any other actions.
r2. Jury members must be able to assign themselves to an available participant.
r3. Jury members must be able to see live video of a participant assigned to them.
r4. Jury members must be able to see real-time horse monitoring data of a participant assigned to them.
r5. Jury members must be able to see recorded video clips of a participant assigned to them.
r6. Jury members must be able to perform trick-play on a video clip of a participant assigned to them.
r7. Jury members must at any time be able to reach verdicts about the performance of a participant, if a verdict about that performance has not already been published.
r8. The system must publish verdicts to the EIS, directly after a jury member has taken the decision.
r9. Jury members must be able to announce race control decisions at any time during the Military event.
r10. The system must publish race control decisions to the EIS, directly after a jury member has taken the decision.
r11. The system must, when information is being published, always unambiguously describe who it was – in case the information to be published is the direct result of a user's action – that generated the information.
r12. The system must store a permanent archive of all of its publications (verdicts and race control decisions) done during the cross, including the description of who generated the information (**r11**).
r13. The system must interface with the *horse, camera, and contest* components of the EIS.

7.2. Should

r14. The system should notify the responsible jury member when the CEL of a participant assigned to them exceeds the threshold value of 100.
r15. The system should prompt jury members to take a verdict about the performance of a participant they are assigned to every time that participant passes an obstacle.

7.3. Won't

r16.Jury members won't authenticate with the system.
r17.The system won't be running on a mobile device.

7.4. Justification

The system requirements are the result of meetings with the product owner. During these meetings we refined what the system must be able to do and how it should behave. This was a time consuming process. There was a meeting with an end user of the system towards the end of the project. Unfortunately, this meeting could not be scheduled earlier, but luckily the product owner had a fairly good understanding of what the end user needed from the start. The late meeting has served as a confirmation that we were on the right track.

Regarding the justification of the priority of functional requirements; the two requirements placed in the *should* category are there because they are small requirements which in and of themselves do not truly add a new functionality, but rather extend another functionality by drawing a user's attention to that functionality.

There are two *won't* requirements; r16 and r17. User authentication is considered out of scope, because the application will in the business case be physically secured. Only a select group of people has access to the network and the computers running the system. This makes user authentication a luxury, and given the time constraints of the project, this will not be implemented.

7.5. Constraints

There are several constraints that apply to this system. These are criteria which frame the system in a way such that they ensure interoperability within the EIS, or express technical limitations of choice.

- The incoming live stream enters the system as an RTP stream over multicast. The system must allow for an RTP input, and must support multicast.
- Messages to and from the EIS are sent and received through an MQTT broker.
- External libraries and tools must be available under a license which allows it to be used free of charge, and for commercial goals.
- All EIS components – including the jury application – exist in a fully managed and controlled network. This means that design choices must not be affected by network restrictions, such as firewall issues.
- The end-to-end delay between the frames a Camera Node records and what the user sees must be as close to none as possible. The video streaming solution for the jury application must allow for a low latency live stream.

8. Research

Given the system requirements, research will focus on answering questions related to how those requirements can be met. This is an activity based in literary exploration and quick prototyping. The resulting conclusions will contribute to the technical choices made for implementation. Some preliminary research on video streaming has been done by the product owner, and this was used as a starting point.

The research questions are as follows:

1. *How can horse monitoring data be synchronized with recorded video footage shot by a camera node?*
2. *How can a video server allow for the trick-play streaming of video data in a one-to-one relationship between itself and a front-end application?*

Answering these questions has required extensive research. They related to the aspects of the system which were expected – and have proven – to be the most complex. Below, both research reports are described.

8.1. The merging of Video- and Horse Monitoring data

This research was used to mitigate uncertainties regarding best practices for the synchronizing and combining of video material and horse monitoring data (VHM merging) before designing and implementing the EIS jury application system. This merging can be approached from different angles. These approaches have been investigated and compared. For the full research report, please refer to the external document of the same name ("The merging of Video- and Horse Monitoring data").

Recall the first research question:

How can horse monitoring data be synchronized with recorded video footage shot by a camera node?

This comes down to three more sub-questions:

1. *Where and how are video- and horse monitoring data stored;*
2. *Where and how are video- and horse monitoring data evaluated, and;*
3. *Where and how are video- and horse monitoring data controlled?*

At the time of the research, the entities involved with the system and its (functional) requirements were known. This resulted in a global understanding of the system at the highest level of abstraction. A simplified decomposition of the jury application is shown in Figure 9.

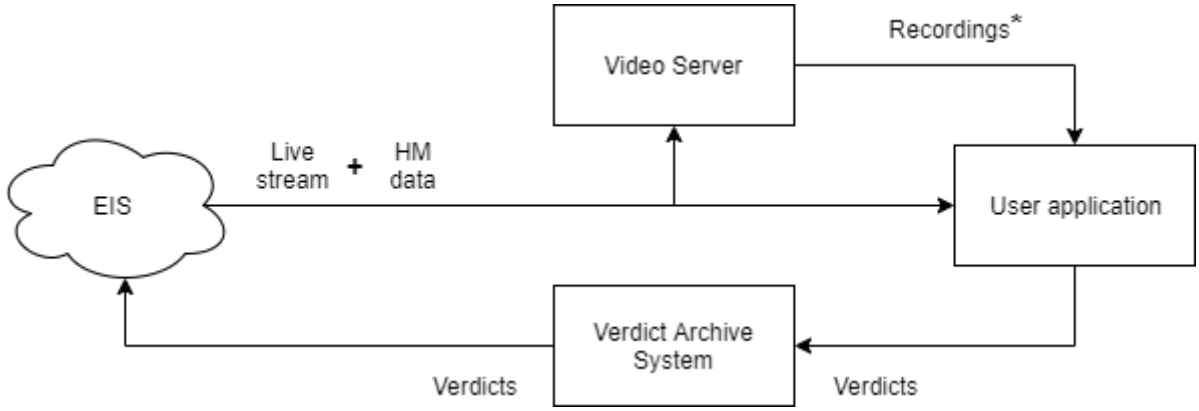


Figure 9: VHM merging system overview

The research focuses its scope around the entities *Video Server* and *User Application* of Figure 9. This includes both the entities themselves (their responsibilities, their in- and outputs) and the connection(s) between them. We discern three approaches to VHM merging:

A. Separate resources;
    - Video data and horse monitoring data (HM data) are stored and transmitted separately from each other.
B. Packed resources;
    - Video data and HM data are stored and transmitted as one package.
C. Multiplexing;
    - Video data and HM data are stored separately, but transmitted together.

8.1.1.Trick-play

An important factor in choosing an optimal approach for VHM merging is the complexity that trick-play operations would impose on the system. Not only is trick-play a central feature of the jury application, it also greatly affects VHM merging. For an elaborate comparison of how trick-play is approached in different VHM merging scenarios, please refer to the external document “The merging of Video- and Horse Monitoring data” included with this report.

8.1.2.Conclusion

It's important to remember that all of these approaches would do their job just fine. They would all provide the behavior expected from the system, and their effectiveness is not a discriminating factor. When comparing the three approaches mentioned above, there are several categories in which they differ. They describe the same functionality (**r5** and **r6**), and do so with the same base components. The following questions relate to how each scenario differs:

1. Is data stored separately or combined?
2. Who takes charge of the content being delivered?
3. Which component is responsible for VHM merging / presenting VHM data synchronously?
4. What is the complexity of the approach on the server side?
5. What is the complexity of the approach on the user application side?

	A. Separate resources	**B. Packed resources**	**C. Multiplexing**
Data storage	Separate	Combined	Separate
Content delivery	PUSH / PULL	PUSH / PULL	PUSH
VHM merging	Video player (client)	Packer (server)	Multiplexer (server)
Complexity on server	Low	High upon storage, Low upon request	Low upon storage, High upon request
Complexity on client	High	High	High
Trick-play complexity	Low	Medium	High

Table 3: A comparison of VHM approaches

Looking at Table 3, Approach A (separate resources) shows the most promising features. Compared to the other two candidates, it promises little complexity on the video server. Having some complexity on the client side of the system seems unavoidable in any case.

Like approach C, it allows for the storing of HM data and video separately, and this allows for these data elements to be retrieved from separate sources. If at any point the system should be extended or altered, this will give a lot of freedom in doing so. Since the design of the EIS is in this stage still subject to many changes, this scenario is not unrealistic.

What's more, is that the operational load on the server when handling requests from many clients at the same time scales much better with approach A than with approach C. This is because of the complexity introduced by multiplexing the data elements in real-time for every connected client. The addition of having to handle trick-play for each of these clients on the server increases this even further.

Additionally, approach A allows for VHM merging using different transmission methods for the video stream and HM data. This means that it can be used regardless of the video streaming protocol. Because of this, we can use the same approach for video and HM data synchronization in a live setting. Approach A is the optimal approach chosen for implementation.

8.2. Video streaming technologies

The purpose of this second research is to discover what video streaming technologies exist, how they work and how they can be used for the jury application. Before starting this research, I had insufficient knowledge of video streaming to use it during development. Informing myself about the subject allowed me to make well-informed choices and decreased development time.

This chapter lists and compares video streaming technologies to answer the following research question:

How can a video server allow for the trick-play streaming of video data in a one-to-one relationship between itself and a front-end application?

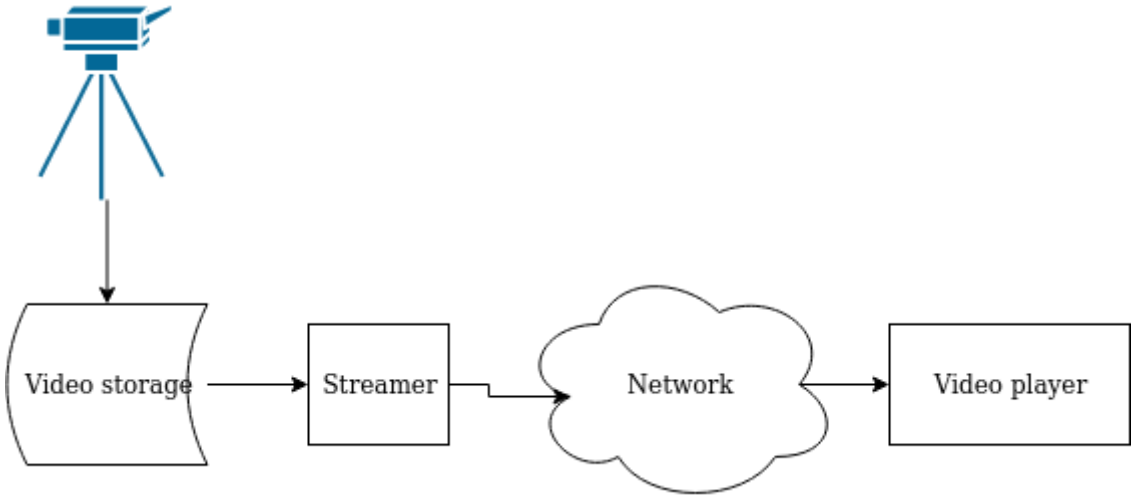


Figure 10: Video streaming flow

In the jury application, video streaming is a way of getting video from storage to a video player over a network (Figure 10). The video player can start the video without the whole file having to be obtained from the video storage. This means that the playing of a video can be started sooner than in a scenario where the whole file must be downloaded first.

Video streaming is a single functionality, but it is supported by several different technologies working together. These technologies have their own purpose and responsibilities. If we focus on video streaming, we distinguish three different categories: codecs, containers and streaming protocols. The full research report is included as an external document with of same name (“Video streaming technologies”).

8.2.1.Codecs

Codecs describe how video material is **co**mpressed and **dec**ompressed. Raw video material consists of a series of still images. To achieve the illusion of motion, these images are displayed in very rapid succession. As an industry standard, 24Hz is generally adhered to as being the frequency at which a series of still images must be displayed to be perceived as fluid motion by humans[1]. Streaming a full image 24 times per second requires an amount of bandwidth is unwieldy, impractical and unrealistic, thus the need for video compression is apparent.

Codecs do not in any other way facilitate the actual storing or streaming of video data. Depending on the streaming protocol, encoded frames might first need to be packed in a *container* in order to reach a client over a network.

8.2.2.Containers

Containers are what describe how video frames are stored, and they provide metadata about the media it contains. This allows a video player or streamer to know how to read the container out, and how to decode its contents. Some containers allow for additional media streams alongside video. A very common example is a synchronized audio stream, but also subtitles are a popular addition.

8.2.3.Streaming protocols

Having compressed a video – and perhaps even having packed it in a container – it is ready to be sent to a client. Streaming protocols take care of getting the video from host A to host B. This could be done over generic application-level networking protocols, like HTTP, or video-streaming specific protocols, like RTMP. Each streaming protocol offers a different approach and uses different underlying technologies to achieve its goal.

8.2.4.Constraints

Implementations of streaming solutions must on either the server or the client side of the system comply with the following constraints:

- The implementation must support trick-play operations.
- The incoming live stream must enter the system as an RTP stream over multicast. The implementation must allow for an RTP input, and must support multicast.
- The implementation must be available under a license which allows it to be used free of charge, and for commercial goals.

[1] "Frame Rate: A Beginner's Guide": https://www.techsmith.com/blog/frame-rate-beginners-guide/

8.2.5.Candidates

The different video streaming technologies were extensively described in the research report. For brevity, a short summary of these technologies is listed here.

RTP (RTP standard, 2003)

Real-time Transport Protocol (RTP) is a protocol designed for delivering media streams over a network connection. It sends media as (multiplexed) elementary streams. It uses the RTSP protocol for trick-play operations.

MPEG-DASH (International Organization for Standardization, 2019)

Dynamic Adaptive Streaming over HTTP (DASH) breaks content into sequence of small HTTP-based files segments, each containing a small portion of the full media file. It uses MPD manifest files to describe the available media.

HLS (HLS standard, 2017)

HTTP Live Streaming (HLS), like MPEG-DASH, breaks content into a sequence of segments which all contain a small part of the full media file. It uses a M3U “playlist” file to describe the available media.

WebRTC (Real-time communication for the web, n.d.)

Web Real-Time Communication (WebRTC) is an open source standard for sending video, voice and generic data between peers. WebRTC is a web standard which is implemented in all modern web browsers. It allows for low latency exchange of data between browsers without requiring an intermediary server.

Progressive download (HTTP/1.1 standard, 1999)

Progressive downloading is a term which describes behavior which is to a user very similar to video streaming, but is at its core slightly different. When streaming a video over progressive download, the video file is what is actually being sent over the network. In a way, this is similar to segment-based streaming protocols such as MPEG-DASH and HLS. However, during a progressive download it is the HTTP protocol which is leveraged by the video player directly obtain segments of a single file on server. This is standardized (Hypertext Transfer Protocol (HTTP/1.1): Range Requests, 2014) and supported in all modern web browsers.

8.2.6. Requirements

Here, the different technologies are weighed against the same criteria to ensure they meet the requirements set out for the system.

The target web browser for the user application is Google Chrome. Since the jury application exists in a controlled environment (managed network, full control of all entities), there are no criteria related to firewall issues.

	RTSP/RTP	MPEG-DASH	HLS	WebRTC	Progressive download
H.264 codec support	Yes	Yes	Yes	Yes	Yes
Web browser support	Scripted	Scripted	Scripted	Native	Native
Low latency	Yes	Configurable (MPEG-DASH packetization, 2019)	Configurable (Configure Apple HLS packetization, 2019)	Yes	Configurable (Progressive download, 2020)
Trick-play support	Yes	Yes	Yes	No	Yes

Table 4: Requirements for video streaming protocols

As shown in Table 4, WebRTC does not satisfy all listed requirements. It does not support trick-play, and can therefore not be used in the system. Aside from this disqualifier, WebRTC's intended use cases are not related to the streaming of video from a server to a client. Even though these design restrictions could theoretically be bypassed, the lack of trick-play support makes it unusable for the purpose of video playback in the jury application.

8.2.7. Comparison

	RTSP/RTP	MPEG-DASH	HLS	Progressive download
Streaming server availability	Poor	Good	Good	Good
Web browser video player availability	Poor	Good	Good	Native
Latency	Low	Configurable	Configurable	Configurable

Table 5: Comparison of video streaming protocols

In this comparison (Table 5), candidates which have passed the requirements are scored. It should be noted that the availability of server streaming tools which implement RTP is good. However, to support trick-play, RTP relies on the RTSP protocol. Even though some servers do implement *part* of this protocol, it is generally used to initiate the RTP stream – not to perform trick-play. The same lack of support for RTSP trick-play is found in libraries and tools for web applications (through a proxy). The libraries that do allow for RTSP do generally not allow for trick-play.

8.2.8.Conclusion

The video streaming technologies which have made it through the requirements check are compared in Table 5. In this comparison, it becomes clear that RTP is not a viable solution for use in this project. In theory it would work well – especially when considering latency – but the lack of available implementations for both the video server and the web application means that a lot of time and effort would have to be spent on creating an RTP suite which can be used to 1) stream to a web browser and 2) handles trick-play. It would cost too much time to build this, and doing so would seem especially pointless if other technologies are readily available.

Note that this same statement (1) applies to live streaming. The live video stream provided by camera nodes can not be displayed directly in a web browser, due to web browsers – by design - providing very limited control over transport layer protocols (UDP and TCP) out of security considerations. This makes it so that live video must in any case be transcoded on the video server and then streamed from there to the client.

The lack of support for RTP brings the choice of technologies down to three. Looking at development-time efficiency, as well as providing a consistent look and feel for users, the ideal solution would be having a single video player implementation for displaying both live streams and video playback.

This means that progressive downloading is not an appropriate technology. It is technically possible to use it for a live stream, but this would require bending the processes that enable it to a point where it's nowhere near its intended use. This could also cost some considerable time to implement, and eventually yields a solution which is non-standard and hard to maintain.

Between the two remaining technologies, MPEG-DASH and HLS, the differences are smaller. They both use segmented media files, and allow for both live streaming and media playback. They are both modern, well-documented solutions which enjoy good web browser support.

While both of these technologies would perform great, MPEG-DASH was chosen for the following nuance; it's an open-source technology, as opposed to HLS which is developed and maintained by Apple.

9. Implementation

The functional design provided the basis for implementation. It described what the user needed and provided ideas for what the product may eventually have looked like. Mockups helped maintain a clear objective throughout the implementation phase of the project. Early decomposition of the system also provided clarity and has allowed for the system to be split up and divided over multiple sprints. Figure 11 shows this decomposition.

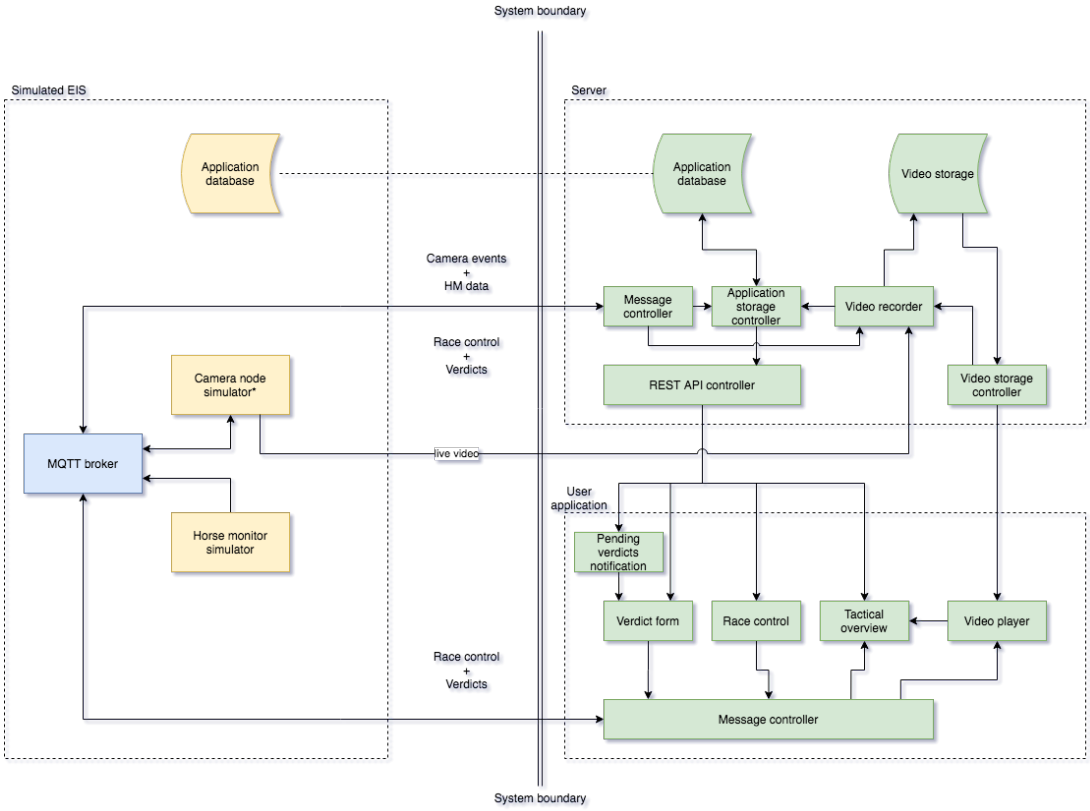


Figure 11: System decomposition overview

In Figure 11, the system is decomposed and placed in its context. The green components are fully within scope of the project. Blue components are provided by the AMI research group to act as a simulation for the EIS. Yellow components have also been provided by the research group, but have been subject to modifications by the graduate as this project required so (see chapter 9.3).

The simulated EIS is deployed in Docker. The separate services each have their own container. They are run and managed with Docker Compose, and they are defined in a YAML configuration file. This simulation runs locally, and communication is done over network ports on the host machine. The jury application is integrated with the simulated EIS. It runs in two separate containers (one for front- and one for backend) and is started with the other EIS components in the same Docker Compose script.

9.1. Server

The server is responsible for providing a REST API, recording video and streaming video- and HM data to the user application. Spring Boot's approach to dependency injection makes it so that there are barely any composition relationships between components; all components are instantiated and maintained by the Spring framework.

This means that the system architecture diagram may seem quite monolithic, but is in fact highly adaptable. All components are coded against interfaces, allowing for easy maintenance and extension, or – with a little work - even splitting the application up into micro services (if such a thing seems necessary in the future). The maintainability of this system is a very important factor in making design choices.

9.1.1. Back-end framework

The server is written in Java on the Spring Boot framework. Spring Boot is chosen as the result of a comparison between different programming languages and back-end frameworks. The conditions for a framework are:

- The framework must be able to provide a REST API;
- The framework must allow for the recording of video over RTP;
- The framework must be able to provide an MPEG-DASH video stream;
- The framework must be familiar to me, or must not have a steep learning curve. This reduces the risks which come from working with (and relying on) something you're not familiar with.

Spring Boot meets all these conditions. I did not have prior experience with Spring Boot, but I am familiar with the Java programming language and the principles of the Spring structure. I expected to be able to use the Spring Boot framework successfully. This choice came with a slight risk factor, as the learning curve could have been steeper than expected, but I had done plentiful preliminary research to feel confident in using the framework in this project. This did not end up hindering me in any way.

Since the server must at the same time be able to handle a REST interface, video recording and video streaming, it is important that these can happen in parallel. Multi-threading is supported in Spring Boot, as opposed to the next most interesting alternative - Node.js - which uses an event loop to handle asynchronous operations.

Spring Boot is backed by a large amount of libraries and tools. This cuts down on development time by making clever use of what is readily available.

I had more experience using the Node.js framework, but given the limitation of not being able to introduce threads on demand, it is not suitable for providing the aforementioned features simultaneously[2].

Alternative server frameworks are just as foreign to the graduate as Spring Boot is, so a good understanding of the Java programming language is what makes Spring Boot the best candidate. This framework is also known to the AMI research group, which benefits future maintenance and development.

The dependency and build toolchain for the server application is Maven. This was chosen because it's well-documented for use with Spring. I also personally like its project structure.

9.1.2. Application database

A relational SQL database is used for storing application data. The choice for an SQL database (instead of a NoSQL one) was made because of the structure it imposes on the data it stores, as well as allowing for the use of a "universal" query language. With SQL, it doesn't matter which database manager is used; they all speak the same language.

The structural nature of SQL means that a database can be designed and instantiated before storing any data, and that the database will always follow these design constraints. This is imperative in the jury application system, as "business" data is archived for later reference (i.e. for checking if a verdict was taken justly).

Separation of the data storage and the data access layers of the database through SQL queries allows for a free choice of which SQL database management software to deploy.

This database satisfies requirements **r11** and **r12**.

The database manager of choice during development was PostgreSQL. This database has a nice interface for looking into the structure and data inside the database while it's running. This tool is called pgAdmin. PostgreSQL is also the database found in the EIS simulation, so future integration into the EIS or the merging of databases should be a breeze for colleagues at the AMI research group. The jury application database schema is defined in Appendix C.

[2] "Don't Block the Event Loop ...": https://nodejs.org/uk/docs/guides/dont-block-the-event-loop/

9.1.3. Message Controller

Incoming and outgoing messages from and to the EIS are broadcast over MQTT. The server uses a Message Controller component to handle this communication. It satisfies functional requirement **r13**, and contributes to **r12**.

9.1.3.1. MQTT

The Eclipse Paho MQTT Client library is used to implement this component. This library is the most prominently available MQTT client for Java development. It is recommended by many sources, amongst which Oracle[3] tutorials. The library allows for both synchronous and asynchronous MQTT communication, and is well-documented.

All messages over the MQTT broker in the EIS are in the JSON format. They are received by the Message controller component, and parsed using the Gson library. This parser is chosen for its simple interface which allows for the de-serialization of JSON into a POJO directly.

9.1.3.2. Operation

During initialization, the Message controller connects to the MQTT broker and subscribes itself to the MQTT topics of interest, as described in Table 6 below. Upon receiving an MQTT message, the Message Controller parses this into their POJO representation. Depending on the type of message, it then invokes appropriate methods of the Application Storage Controller and Video Recorder components.

The system must archive all jury publications (requirement **r12**). When a jury member takes action, the User application broadcasts this to the EIS over MQTT and is received by the Message controller. This component will then use methods exposed by the Application storage controller to save these messages in the Application database.

Starting and stopping the recording of live video is done upon receiving the *approach* and *leave* messages from the EIS. The message controller therefore instructs the Video recorder component to start or stop respectively.

[3] "Simple Messaging with MQTT": https://www.oracle.com/corporate/features/simple-messaging-with-mqtt.html

Topic	Description	Message

position/**[participant_id]**	The latitude/longitude position of the specified participant is published to this topic at a 1 second interval.	{
		“participantId”: int,
		“lat”: float,
		“lon”: float,
		“time”: long
		}

heartrate/**[participant_id]**	The heart rate of a participating horse is published to this topic at a 1 second interval.	{
		“participantId”: int,
		“heartrate”: int,
		“time”: long
		}

verdict/**[obstacle_id]**/ **[participant_id]**	The verdict a jury member took relating to a participant crossing an obstacle.	{
		“participantId”: int,
		“obstacleId”: int,
		“juryId”, int,
		“penalty”: int,
		“description”: string,
		“time”: long
		}

participant_removal/ **[participant_id]**	The decision of a jury member to remove a participant from the race.	{
		“participantId”: int,
		“juryId”, int,
		“description”: string,
		“time”: long
		}

race_control	Any decisions made by a jury member which relates to the stopping or continuing of the race (0 = in progress, 1 = ended, -1= halted).	{
		“juryId”, int,
		“raceStatus”, int,
		“description”: string,
		“time”: long
		}

approach/**[obstacle_id]**/ **[participant_id]**	The “approach” event of a participant coming close to an obstacle, triggering a camera live stream.	{
		“participantId”: int,
		“obstacleId”: int,
		“time”: long
		}

leave/**[obstacle_id]**/ **[participant_id]**	The “leave” event of a participant having passed an obstacle, stopping a camera live stream.	{
		“participantId”: int,
		“obstacleId”: int,
		“time”: long
		}

Table 6: MQTT topics to which the Message controller subscribes

9.1.4. Application storage controller

The application storage controller is responsible for connecting to the Application Database and abstracting data access for the rest of the system. It will allow for other components to store data at or access data from the database without those components having to connect to the database themselves. It uses the SQL query language to perform operations on the Application database component. This component satisfies functional requirement **r12**.

The database driver, host, and user credentials are defined in the *src/main/resources/application.properties* file. This can be overridden in an external configuration file (see chapter 9.4: Deployment and Configuration).

9.1.5. REST API

The REST API component allows for the user application to interact with the server. It follows a RESTful design paradigm where connections are stateless, and requests are centered around resources. Because of the controlled environment in which the system runs, connections and requests are not secured, and any client can make any request to the API.

9.1.5.1. Endpoints

All REST API endpoint responses are in the JSON format. The Spring framework has the Jackson JSON parser on the class path. This automatically converts POJOs returned by request mappings to JSON. The resulting JSON string is then sent to the client over HTTP.

The API only exposes endpoints for HTTP-GET request methods. A summary of these endpoints can be found below in Table 7. The full list of endpoints can be found in Appendix D.

9.1.6. Exceptions

If an API request could not be completed, a HTTP response code should specify to a client what went wrong. Spring offers the option to automatically send a HTTP response code when an exception is thrown in Java. The REST API internally throws the following exceptions, corresponding to the following HTTP response codes:

Exception	HTTP response	Description
RecordNotFound	404 (not found)	Thrown when a single specific resource was requested, but not found in the data store. Never thrown when a collection of resources is requested (the collection may simply contain 0 elements).

Endpoint	Response
/api/videos	Returns information about all video clips. The file URI path in the response body does not specify a host. Response may be an empty JSON array.
/api/videos/{participant_id}	Returns information about video clips relating to the specified participant. The file URI path in the response body does not specify a host. Response may be an empty JSON array.
/api/videos/{participant_id}/ {obstacle_id}	Returns information about a single video clip relating to the specified participant and obstacle. The file URI path in the response body does not specify a host. May result in a HTTP 404 response code if there is no video clip information relating to the specified participant and obstacle. If multiple records were found, this endpoint returns the first result.
/api/positions/{participant_id}	Returns a list of positions relating to a specified participant. By default returns all position data available. Optionally returns only position data within a specified time range. Response may be an empty JSON array.
/api/heart_rates/{participant_id}	Returns a list of heart rate measurements relating to a specified participant. By default, returns all heart rate data available. Optionally returns only heart rate data within a specified time range. Response may be an empty JSON array.
/api/verdicts/{participant_id}	Returns a list of verdicts relating to a specified participant. By default, returns all verdicts available. Optionally returns only verdicts within a specified time range. Response may be an empty JSON array.
/api/verdicts/{participant_id}/ {obstacle_id}	Returns a list of verdicts relating to a specified participant and obstacle. By default, returns all verdicts available. Optionally returns only verdicts within a specified time range. Response may be an empty JSON array.
/api/verdicts/{participant_id}/ pending	Returns a list of pending verdicts relating to a specified participant. By default, returns all pending verdicts available. Optionally returns only pending verdicts within a specified time range. Response may be an empty JSON array.
/api/participant_removal/ {participant_id}	Returns a list of participant removal events relating to a specified participant. Returns all participant removal events available. Response may be an empty JSON array.
/api/race_control	Returns a list of race control events relating to a specified participant. Returns all race control events available. Response may be an empty JSON array.
/api/race_control/latest	Returns a list of race control events relating to a specified participant. Returns all race control events available.

Table 7: REST API endpoints

9.1.7. Video storage controller

Recorded video clips are stored on the server. Because the system records video files using an external tool (see chapter 9.1.9), it is not possible to store them in a database. Therefore, the files are stored on the host machine's file system. They are distributed over multiple directories. The structure is as follows:

- Root video storage directory
  - Year of the recording (directory)
    - Obstacle ID (directory)
      - Participant ID (directory)
        - Video clip manifest:
          - *video_manifest_**[obstacle_id]_[participant_id]_[time]**.mpd*
        - Initial video segment:
          - *video_init_**[obstacle_id]_[participant_id]_[time]**.mp4*
        - Video segments:
          - *video_segment_**[obstacle_id]_[participant_id]_[time]-[segment_number]**.mp4*
          - …

The root video storage directory can be set in the server configuration file using the jury_application.server.video.root-directory property. See chapter 9.4: Deployment and Configuration for more information about how this is done.

The *obstacle_id* and *participant_id* fields are integers, and the *time* field is an integer UNIX epoch timestamp in milliseconds. The *segment_number* integer field increments by one for each consecutive segment, starting at one.

The MPEG-DASH *.mpd* manifest file contains information about the video segments which are part of this video clip. These segments are stored on disk alongside the manifest. There is one initial video segment, followed by any number of additional video segments.

The video is stored across these multiple segments in the ISOBMFF[4] media format. This media file format is the basis for the MP4 container format, and video clip segments (including the initial segment) carry the .mp4 file extension. The initial segment contains metadata which relates to the whole recording, following the DASH specification. All video segments contain H.264 encoded video frames. MPEG-DASH's Adaptive Bitrate (ABR[5]) feature is not used.

[4] "ISO/IEC 14496-12": https://www.iso.org/standard/68960.html

[5] "Adaptive bitrate streaming": https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming

The video storage controller is also responsible for providing video clips upon request. Given that MPEG-DASH works over HTTP, the video storage controller must provide HTTP GET endpoints for a video player to request video clip manifests and -segments. It exposes all files in the root video storage directory over the following HTTP path:

/video/

An example of a GET request URL for a video clip manifest may then look as follows:

```
http://host_address:port/video/2020/14/8/
      video_manifest_14_8_1601724661381.mpd
```

9.1.8. Video recorder

This component records incoming live streams, which are external inputs coming from camera nodes in the EIS. The video recorder records all the live videos which any of the cameras stream.

In order to record live streams, the video recorder listens to the *approach* and *leave* events which are produced by Camera Nodes and published over MQTT. The Video recorder is notified of a camera starting their stream by the message controller. Camera Nodes listen to these same messages to start and stop their live stream.

It is possible that the approach message is received on the server well before a camera node starts the actual video stream. This is because the camera node, after receiving the *approach* event, will use computer vision techniques to determine if there is a horse in frame. Only when a horse is detected in frame does the Camera Node starts streaming video. The video recorder is currently configured to expect a maximum delay of 25 seconds between receiving the *approach* event, and the camera starting their stream. If no video is received within 25 seconds, the recording is flagged in the database as “unavailable”.

The video recorder creates a new thread for each *approach* event to record the specified live stream. The component keeps track of which recordings are currently running based on which participant is being recorded, because there is at any given time at most only one recording being made of a certain participant: they can physically not be at two obstacles at the same time.

9.1.9. Recorder tasks

Recorder tasks can run in parallel on their own thread. They are created by the video recorder when an *approach* event is received. The thread is stopped by the video recorder when the *leave* event is received, or the video stream is stopped. It requires the external *FFmpeg* binary, which allows for the receiving of an RTP stream and saving the received video frames in the MPEG-DASH format. Each Recorder task invokes and manages a process instance of the FFmpeg tool by interfacing with the host environment through Java's *java.lang.Runtime* and *java.lang.Process* objects.

Stream consumers are added to the error- and input streams from the process. These are in place to prevent blocking or deadlock in scenarios where operating systems only provide limited buffer sizes for standard output streams. Failure of a process to write to these output streams may cause it to block[6].

The FFmpeg tool is started using the following command:

```
ffmpeg -abort_on empty_output -fflags +genpts -protocol_whitelist
file,rtp,udp -analyzeduration 25000000

      -i {input_SDP_file} -r 24 -vcodec copy -g 12 -keyint_min 12 -b:v
5000k -map 0:v -start_at_zero

      -f dash -streaming 1 -flush_packets 1 -seg_duration 0.5 -
use_template 1 -use_timeline 1 -minimum_update_period 1
-init_seg_name {init_seg_name} -media_seg_name {seg_name} {manifest_name}
```

This is one command, but since it's quite long, it is here split up into three sections; general options, input and output.

The general options describe that FFmpeg must 1) stop when writing to the output file has stopped, 2) generate its own timestamps for outputting frames, 3) allow for using a SDP file and RTP over UDP as an input, and 4) stop expecting to receive input if after 25 seconds nothing has been received yet.

The input section instructs FFmpeg to 1) use a framerate of 24 frames per second and generate timestamps at this frequency, 2) not transcode the video frames, but copy them over to the output file directly, 3) set the *group of pictures* size to 12 frames, 4) inject a keyframe at an interval of 12 frames, 5) use a bit rate of 5Mbit/s for the video, 6) map the first input stream to the output video stream, and 7) start relative timestamps at 0.

6 "Process javadoc": https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html

The output section starts by specifying the DASH muxer, and describes that 1) the input is a live stream source, 2) FFmpeg must write input frames to the output segments immediately, 3) DASH segments have a duration of half a second, 4) the manifest file must use a template representation of segments, numbering them incrementally, 5) the manifest file must use a segment timeline, and 6) the manifest should suggest video players to update the manifest file every second. The next options just specify the names of the initial segment, all subsequent segments, and the manifest name.

It should be noted that because we are a file system, we must ensure a unique name for each segment. The "use_template" option helps us with this by replacing a placeholder in put segment name with an incremental number for each new segment. This placeholder is $Number$ such as this:

```
Segment_name-$Number$.mp4
```

Another thing is that a low input keyframe interval is key to the ability of FFmpeg to create small-size segments (and therefore offer a low latency video stream) without using a transcoder. Using a transcoder consumes a lot of computer resources and should be avoided; especially because multiple recordings may occur at the same time. To be able to create segments of half a second, the keyframe interval must be half the number of frames in a second, because every DASH segment must start with a keyframe. For the scenario described above, the live stream is 24 frames per second. The keyframe interval is 12 frames, allowing for $\frac{12}{24} = \frac{1}{2}$ second segment lengths.

9.2. User application

The user interface of the jury application is displayed through a web application. This allows for platform-independent use, which makes it easy to deploy. There is a degree of uncertainty regarding the devices available to the end user. Web applications are also generally easier to maintain than native applications. There is no product requirement about the nature of the user interface.

The target web browser for this application is Google Chrome version 79.0+ (current latest). It is available for the Windows, Mac, Linux and Chromebook platforms, and undergoes active development. It is kept up to date with the latest web standards, and so offers wide support for technologies.

The application is written in React. This was chosen because I had some previous experience with the framework. It is also maintainable by the AMI research group.

9.2.1. The React framework

React is a framework which handles the states and rendering of React Components. A React Component is a combination of HTML markup and JavaScript logic. All functional components which have a visual aspect are implemented using React Components. Other functional components are invoked from these React components.

One of React's core principles is the idea of having components exist in a hierarchical tree. Each component maintains its own *state*, and may have its properties defined by their parent component – the so-called *props*. In essence, React encourages a one-way information flow (from parent to child). This is not always possible, however, and it is quite common to have a parent provide its child with a callback property for receiving information back from it.

9.2.2. Component hierarchy

The component hierarchy of the user application is insufficiently represented in the system overview of Figure 11. Below, Figure 12clarifies the composition of the system before getting into the specifics of each component. Read this diagram as you would a tree data structure. "App" is the root component, and all others are children to it. The relationship between components is always parent-child (in the diagram from top to bottom). Child components are generally displayed in front of their parent. A Switch component means "display one of these". This is how multiple screens are organized.

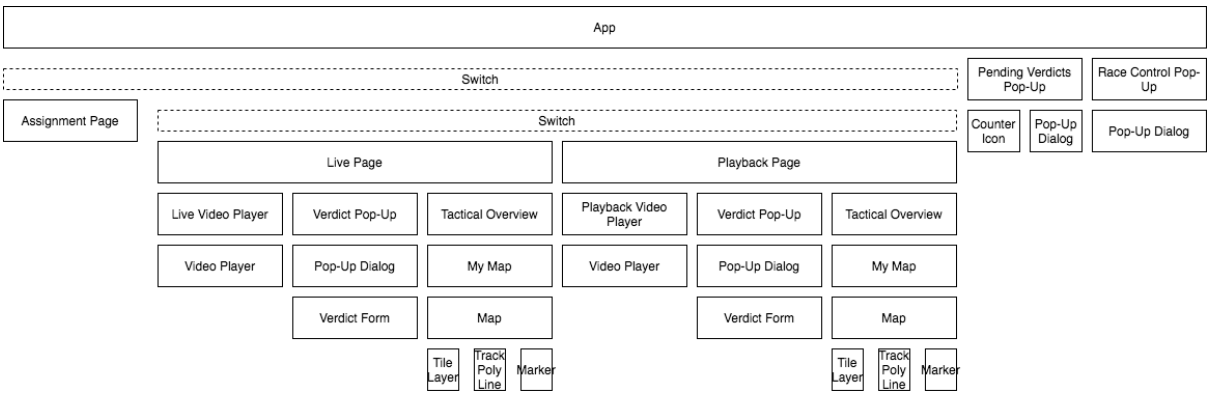


Figure 12: User application hierarchical decomposition

9.2.3. Component re-use

As shown in the component hierarchy of Figure 12, there are several components which are used multiple times throughout the user application. These components are the exact same, and their difference in behavior is defined by the properties passed down by their parent component.

9.2.4. Routers and Switches

React applications can consist of multiple pages. In web applications, it is common for each page to have their own URL path. Using this approach in our user application allows for a uniform way of navigation, and for the use of hyperlinks.

Each page is in fact another component, and specifying which component is displayed when a certain URL is visited, is done by *wrapping* them with React's *Router* and *Switch* components.

These URL paths can contain variable levels. The values of these are passed into the page component. This means that a single page can dynamically change based on which URL path is currently visited. This is used by the Live- and Playback Pages to determine which participant is currently assigned to the user, and – in case of the Playback Page – which obstacle is being requested by the user.

The associations between paths and components (the routes) are as follows:

Path	Component
/	Assignment Page
/jury/{participant_id}	Live Page
/jury/{participant_id}/playback/{obstacle_id}	Playback Page

Table 8: Associations between paths and components

9.2.5. MQTT client

The user application uses the mqtt.js dependency for interacting with the EIS. This dependency offers an MQTT client which can subscribe and publish to topics on the MQTT broker. Since this is a web application, the connection between the client and the broker must be done over WebSockets. The Mosquitto MQTT broker of the simulated EIS supports WebSockets.

The MQTT client instance is initiated when the application starts, and it tried connecting to the MQTT broker immediately. If this fails, it will keep retrying until its successful. If the connection is dropped while the application is running, it will again continuously attempt to reconnect.

One MQTT client is shared by all components of the application. It is passed down from the App component to all child components that need it, or whose children need it. The client is subscribed to all topics in which the application may be interested, and each component that wants to be notified when a message arrives on a certain topic can add an event handler to the client.

The MQTT client can be configured in the application configuration file. More on this in chapter 9.4: Deployment and Configuration.

9.2.6. Components

This chapter describes interesting or complex components in more detail. Not every component requires explanation because their functioning is quite simple to infer from either their name or by glancing over the code. A pop-up dialog, for example, is a little dialog window which pops up. The components which are less straight forward are described here.

9.2.6.1. Video Player

The video player component is a component which can play DASH video streams. It has trick-play control options which allow for rewind/fast forward, play/pause and move forwards or backwards per frame. This component takes the following properties as inputs (Table 9):

Property	Description
videoSrc	This is a string with the URL to the DASH manifest file. When this video source changes the video player will load the video immediately.
startTime	The start time of the video in UNIX epoch milliseconds.
onTimeUpdate	An event handler (function) which gets called at possibly irregular intervals when the HTML5 video tag updates its playing time. This interval depends on the web browser video tag implementation, video playback rate, video framerate and user interaction. Times are in the UNIX epoch in milliseconds. This function gets passed three values: start time, current time, and end time.
onVideoStarted	An event handler (function) which gets called when a certain video first starts playing. This is not called every time the play button is pressed.
onVideoEnded	An event handler (function) which gets called when a certain video stops playing by reaching the end of it.
controls	This property is a Boolean value which specifies whether or not controls should be shown in the video player. The full screen button is always visible, regardless of this property.
autoPlay	This property specifies whether or not the video should start playing immediately when it's done loading.
poster	This is a string which contains a URL to a poster image. The poster will be shown when no video is currently loaded.

Table 9: Input properties of the Video Player component

The Video Player component has two components which contain it. These components provide the player with specific behavior. Re-using the Video Player component for live streams and playback provides the user with a consistent look and feel. It also cuts down on development time. This was made possible by the choice to use MPEG-DASH for live streaming as well (instead of RTP through a proxy).

The Live Video Player is a specialized video player which keeps track of when the participant arrives at the next obstacle and starts loading and playing the new incoming live stream. The obstacle ID is provided by its parent, the Live Page component, which observes the *approach* MQTT topic to change the obstacle ID it passes down when a participant approaches the next obstacle. For the input proterties of the Live Video Player component, see Appendix F.

On the Military track, there is a gap between obstacles, and a participant is only filmed at obstacles. This means that, when following a participant live, there are moments when no video is being played. The tactical overview does keep showing positions and heart rates, and since the latency of live video is much higher than that of live positions and heart rates, the tactical overview must be "delayed" when a live video is playing. This delay is not applied when no video is playing.

The Playback Video Player is used for playing back videos after a live stream has ended. This component takes as inputs a participant- and obstacle ID, and loads the corresponding video. For a table containing the Playback Video Player's input properties, see Appendix F.

9.2.6.2. Tactical Overview

To provide jury members with a complete overview of what is happening with participants during the cross-country, the application displays a tactical overview next to any video player. This provides the jury with the context they may need to come to a well-informed decision.

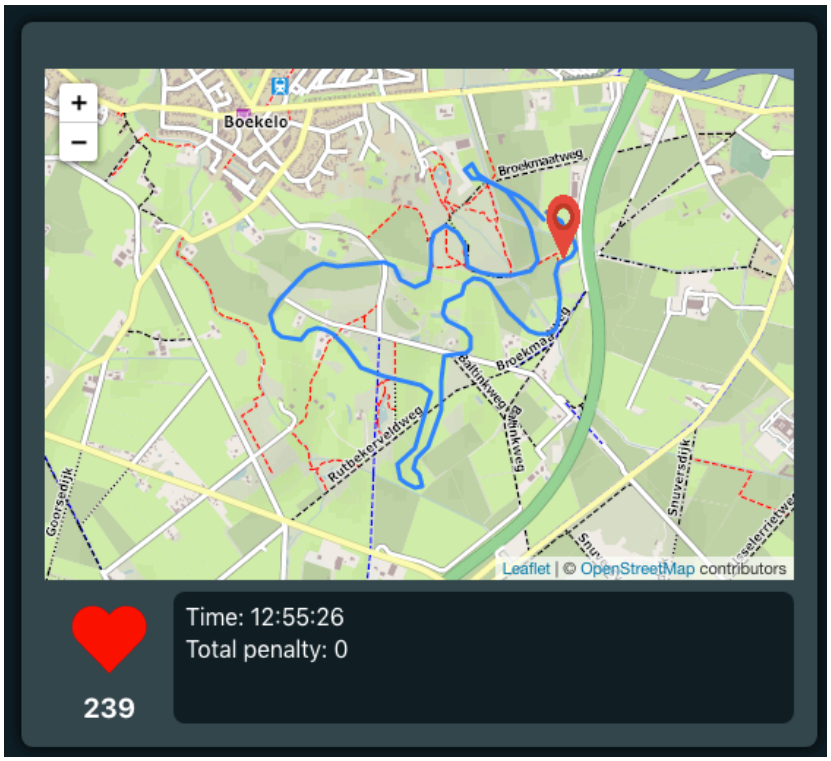


Figure 13: The Tactical Overview user interface component

The tactical overview component displays the position of an assigned participant on a map of the cross-country track, together with their heart rate and penalty score. It has two modes of operation; live and playback. In a live view, the map shows all participants currently on the track, and the assigned participant is highlighted. In a playback setting, the map only shows the participant to which the recording relates.

The tactical overview is synchronized with the video player when it's playing a video. In a live setting, however, it happens frequently that the video player is not playing a video for a period of time. Synchronizing with a live video player ensures that the jury member sees matching events on the tactical overview and in the video. If a live video stream has a high latency in it, an unsynchronized tactical overview would show positions and heart rates which are ahead of what can be seen on video. On the other hand, we do not wat a delay in seeing the position and heart rate of a participant when there is no delay needed.

This makes it so that when a live video is being played, the tactical overview synchronizes with it – effectively accounting for the video latency. When a video is not being played, the tactical overview shows the current horse monitoring values with no synthetic delay.

Table 10 shows the properties which the tactical overview takes as inputs.

Property	Description
participantId	The unique identifier specifying which participant is monitored by the user.
obstacleId	The unique identifier specifying of which obstacle the participant's performance is being played back.
mqttClient	The MQTT client passed down to the tactical overview allows it to display live heart rates. It passes this property down to the MyMap component which uses it to display live positions.
live	This Boolean property specifies whether this tactical overview should display live HM data or that it should synchronize based on the specified time properties.
startTime	When synchronizing the tactical overview, this specifies the UNIX start time of the recording in milliseconds.
currentTime	When synchronizing the tactical overview, this specifies the current UNIX time of the recording in milliseconds.
endTime	When synchronizing the tactical overview, this specifies the UNIX end time of the recording in milliseconds.

Table 10: Input properties of the Tactical Overview component

The Tactical Overview component synchronizes HM data with a video recording by having its time state updated by a video player, and obtaining the corresponding HM data from the REST API. This is done using two instances of the HMDataBuffer utility class; one for heart rates and one for positions.

HMDataBuffer

The HMDataBuffer utility class allows for easily maintaining buffers of heart rate- and position measurements which can grow dynamically in size. The size of these buffers can be specified in a configuration file (see chapter 9.4: Deployment and Configuration). Specifying this size helps ensure that the amount of requests made to the jury application REST API is limited. A network manager of the EIS might find this helpful in the future.

Measurements are obtained from this buffer by specifying a "target time" – a time at which you want to know the value of a measurement. The buffer will then return the measurement closest to this target time. To do so, it uses a simple adaptation of the binary search algorithm based on an online example (Find the closest number in an array, n.d.).

9.3. EIS simulation

Changes were made to the EIS simulation to work with the jury application. These changes must be taken into account when integrating the jury application with the EIS.

Changes were made to the camera node simulation. Simulating an RTP live stream is achieved using GStreamer, but this streamer was previously started with an argument which had it send out Picture Parameter Sets only once every ten seconds. This interval was too long to be useful to the jury application. The interval was changed to match the source video (now half a second). The shorter this interval, the lower the video live stream latency.

The simulation previously also had only one camera node. The jury application works by having a jury member follow a participant as it crosses the obstacles on the track. Having only one obstacle does not allow for demonstrating this. The simulation now has three camera nodes along the track. The demo source video was also changed out for three different ones; one for each obstacle.

The simulated horse monitor now also sends out “participant started” and “participant finished” MQTT messages when a participant starts or completes the race.

The message broker was configured to only work with the MQTT protocol over TCP. The simulation uses the Mosquitto broker. Because the jury application uses a web front-end, we had to configure Mosquitto to allow for use with WebSockets.

9.4. Deployment and Configuration

Docker simulated EIS environment from AMI research group.

The jury application is deployed in two parts; a server backend and a web application front-end. These parts both require interaction with the EIS to work. We use Docker Compose to deploy both parts of the jury application and the EIS simulation at the same time. This also ensures that the container environments can be tailored to the specific needs of the system running in it. Any machine that runs these docker containers will then behave and work the same, making the solution very portable.

The simulated EIS was already containerized when it was provided for this project. The jury application was integrated with this, and the entirety of it can now be run with the following command:

```
docker-compose up
```

The private GitHub repository which contains the complete project and EIS simulation can be found here: https://github.com/SaxionAMI/Military-docker-architecture.git on the *jury_application* branch. Access can be requested at the AMI research group.

Ntagengerwa, Aimé
Version 1.0

9.4.1. Server configuration

Configuration of the server is done through an external configuration file. This configuration file is named *application.properties*, and consists of key-value pairs. Note that string values do not require double quotes. Each pair goes in a new line. There are properties which are part of the libraries, and there are those which are custom to this application. The ones which are custom are listed in Table 11:

Variable	Description
jury_application.server.video.root-directory	Specifies the root directory to which video clips are saved (see Video Storage Controller). The value must be wrapped in double quotes. This path must not have spaces in it.
jury_application.server.mqtt.broker	Specifies the protocol, host address and port of the MQTT broker as a URL (i.e. tcp://127.0.0.1:1883).

Table 11: Custom server configuration variables

Other configuration properties (Appendix E) which are important for deployment are related to the database connection. The default value for the database driver is for PostgreSQL and is not required when connecting to a PostgreSQL database.

The location of this external configuration file must be specified when running the application. Because it is specified at runtime, the server configuration can change between runs y changing the *application.properties* file. The following program argument must be added when running the server:

--spring.config.additional-location="/path/to/configuration/application.properties"

9.4.2. User application configuration

The user application can be configured with a JSON object. This configuration can be found and edited in the *config.js* file in the project source directory. This configuration is bundled with the application for deployment. It is therefore necessary to rebuild the application when the configuration file is changed. Appendix E shows the configuration options with their default values. Please note that the application requires all configuration fields to be defined.

9.5. Additional documentation

Additional documentation about the implementation of the system and its technical details is available in the Technical Design included with this report as an external document. The server project also includes the ability to generate Javadoc using Maven. This provides detailed documentation of the project's classes. To generate this, use the following command from the project root directory (where *pom.xml* lives):

mvn javadoc:javadoc

This will generate HTML pages for navigating and displaying the project's classes. The files can be found in the *target/site/apidocs* directory, relative to the project's root.

10. Integration and acceptance

For the jury application to be useful, it must be able to integrate with the EIS. Most of the EIS has not been built yet at time of writing, but the communication between its subsystems is largely defined. The simulated EIS environment uses these interfaces and allows for integration testing before the whole EIS is built.

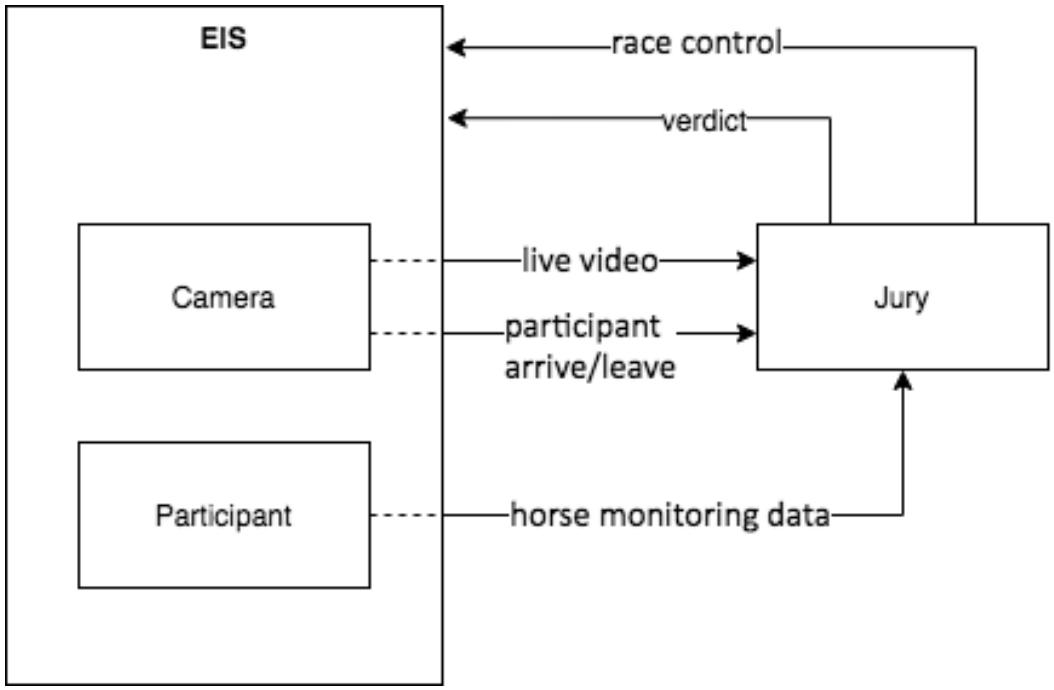


Figure 14: The jury application's external interfaces

The jury application is successfully integrated with the EIS. Taking note of Figure 14, the system communicates with other EIS components through the message broker using the provided interface definition (see chapter 9.1.3.2), and it records live streams from the simulated camera nodes. It also stores all horse monitoring data that it receives from the EIS.

10.1. Acceptance testing

The project included a lot of communication with the product owner. Weekly meetings were held in the preparation phases – when defining the product – as well as during development. During development, these meetings took the form of sprint reviews. The use cases implemented during the week-long sprint were addressed and demonstrated. This led to acceptance on a component level (Table 12). The product as a whole has not been demonstrated and explicitly accepted yet, but we do plan on doing this before the end of my time at the research group. This was not possible sooner due to my personally getting sick and the company closing its premises for the duration of the government-imposed restrictions surrounding the development of the COVID-19 viral outbreak.

Use case	Requirements	Description	Accepted
UC.1	r1	Fill in information which uniquely links a user to a jury member.	No
UC.2	r2	View which participants are taking part in the event, and what their status is.	No
UC.3	r2	Assign a participant to a jury member.	No
UC.4	r3	View the live video footage of a participant crossing an obstacle.	Yes
UC.5	r3, r4, r5, r14	View a tactical overview of horse monitoring data, showing a participant's position on a map and the heartrate and CEL of a horse.	Yes
UC.6	r5	View a recorded video clip.	Yes
UC.7	r5, r6	Perform trick-play on a video clip.	Yes
UC.8	r7	A jury member takes a verdict based on the observations of a participant.	Yes
UC.9	r9	Announce a jury member's decision regarding the continuation of the race.	Yes
UC.10	r15	View which obstacles have been crossed by an obstacle, but have not yet been subject to a verdict.	Yes

Table 12: Accepted use cases and fulfilled requirements

Table 12 shows that there are three use cases – the ones with the lowest priority – which are not implemented yet. This is because I was sick for quite a long period of time. Given experience with my development speed gathered over the course of the implementation phase, the three use cases would fit in a single sprint. I fell sick before starting these tasks, and the two weeks of development time left at that time were not put to use. These use cases are expected to be finished before the end of my time at the AMI research group.

Only the requirements which are directly tied to use cases are listed in Table 12. The remaining requirements do not have a use case attached to them directly, and it is hard for a product owner to see them in action. For the system to work, however, they must be implemented, and the degree in which the system covers these requirements is listed below in Table 13.

Requirement	Description	Covered by system
r8	The system must publish verdicts to the EIS, directly after a jury member has taken the decision.	Yes
r10	The system must publish race control decisions to the EIS, directly after a jury member has taken the decision.	Yes
r11	The system must, when information is being published, always unambiguously describe who it was – in case the information to be published is the direct result of a user's action – that generated the information.	No
r12	The system must store a permanent archive of all of its publications (verdicts and race control decisions) done during the cross, including the description of who generated the information (**r11**).	Yes
r13	The system must interface with the *horse, camera, and contest* components of the EIS.	Yes

Table 13: System coverage of non-user requirements

Requirement **r11** is not covered by the system, because it relies on the system knowing which jury member is currently using the system. Technically, the requirement is currently fulfilled by specifying a placeholder jury ID for all published information. However, this does not have any meaning until a jury member can identify themselves with the system as described in **UC.1**.

Just like **UC.1**, this requirement is expected to be met before the end of my working agreement at the research group.

11. Working practices

To create a product which solves the core business problems while keeping all stakeholders involved and satisfied is no simple thing. Having had a good plan of approach was crucial to the success of the project. This plan of approach is included with this report as an external document called “Plan of Approach”. This chapter will look back on what actually happened instead of what was planned.

11.1. Progress reports

To keep stakeholders informed about the progress of the project, a weekly summary of activities was published. This was done one day before the weekly sprint review meeting (so on Monday), giving the product owner and coworkers involved an idea of what will be discussed during the meeting.

After the meeting, a sprint retrospective will be created and distributed. This document describes in more details the tasks performed during a sprint, any problems encountered, and the plan for the coming sprints.

Distributing these documents weekly also allows for other stakeholders to read about the state of the project, without having to be actively invested every step of the way.

11.2. Task management

A sprint backlog lists the tasks which must be completed to fulfill the user stories assigned to this sprint. I used the web application Trello to manage this. Trello allows for keeping track of multiple lists and moving items between them. A collection of lists is called a board.

I used a single board for each individual sprint, containing the following six lists:

- Sprint backlog;
- In progress;
- Testing;
- Documenting;
- Done;
- Carry over;

Tasks traversed these lists in order. An example of the Trello boards can be found in Appendix G.

11.3. Quality assurance

Several measures were taken to ensure that the codebase of the system is clear and maintainable. The first is to automatic linter tools which show warning messages when code does not comply with the configured conventions.

The second has proven to be much more valuable, which was in the form of a code review. After having coded a first part of the system, I sent a request for a code review to my coworker/supervisor. He gave me good tips and insight into what practices to follow. I implemented this feedback and took it into consideration for the rest of the project (and my professional career, no doubt).

As for linters, I used the Google Java style guide for my backend and integrated this with my IDE using the corresponding IntelliJ plugin. For the front-end, I used ESLint and Prettier. These style configurations are included with the project on GitHub.

Manual tests were conducted on the backend REST API which helped will providing consistent behavior across endpoints. These tests were done using tools such as *curl* and *Postman*. Video streaming was also subject to testing, and this is described in the document named “MPEG-DASH streaming issues” included with this report as an external file.

11.4. Version control

As a version control method, I used Git. The research group has supplied me with a GitHub repository on which I could store changes to the code base of the project. It also functions as a way to make sure that the research group has access to the developed system at all times.

Every sprint started on a new branch. After discussing the results of a sprint with the product owner, the branches were merged into the master branch. The jury application repository can be found here: https://github.com/SaxionAMI/Military-JuryApplication-Graduation-Aime.git. The EIS simulation environment needed to run the jury application can be found here: https://github.com/SaxionAMI/Military-docker-architecture.git. This repository also includes the jury application itself, so for deploying the system this is recommended.

12. Conclusion

In this conclusion I look back on the project and describe why I believe it has been a success.

The project's primary goal – assisting the cross-country Grand Jury – has always been clear. A large degree of time and effort was put into formulating use cases and requirements which add up to this goal. A meeting with an end user was not possible until quite late in the project, but thanks to the attention to use cases and requirements, this meeting did not result in changes to the system's behavior and operation. Dividing use cases over sprints and reviewing these sprints every week with stakeholders have contributed to keeping the product on track. Not all use cases are implemented yet, but the ones left are expected to be finished before my graduation defense (see chapter 10.1).

Research was conducted to improve the quality of the product and to make domain-specific knowledge available to the AMI research group. This research was focused around two questions:

1. *How can horse monitoring data be synchronized with recorded video footage shot by a camera node?*
2. *How can a video server allow for the trick-play streaming of video data in a one-to-one relationship between itself and a front-end application?*

These questions were answered in chapter 8: Research. The conclusions drawn from answering these research questions have been very valuable for the implementation of the product. They provided understanding of the most challenging aspect of this project – video streaming – and offered suggestions on how implementation could be achieved. This was the foundation on which the technical design of the system was conceived.

Special care was taken to make sure that the integration of the jury application with the rest of the EIS would be successful. Some interfaces of other EIS components were provided by the AMI research group, while others were yet to be defined. This was done by me in agreement with coworkers at the research group. A simulation of the EIS was provided by the research group, and I made adjustments where needed.

The process of designing the jury application started with its external interfaces, working down to a layer of abstraction on the component-level. This design process ensured that the product's context was never forgotten.

Development had a focus on implementing the agreed upon functionality and creating a system which is easy to deploy and configurable. I chose not only frameworks and libraries with which I was familiar, but also some of which I was confident that I could successfully learn how to apply them in this project. This has added to my skillset as a software engineer.

As a whole, the project was executed very independently. This included managing the project timeline, communicating with multiple stakeholders, and convincing those people of my opinions while taking into account theirs to come to the best mutual conclusion for any discussion. This independence also came with the responsibility for making sure that the product could integrate with the context in which it will support the Grand Jury of the Military Boekelo cross-country.

13. Recommendations

This chapter describes areas around which future research and development can be centered. The reasons why these areas were added as recommendations – as opposed to being implemented in the product – vary. The items below are just pointers in the general direction this project could be going in later iterations for extension or maintenance.

13.1. Video recorder

The video recorder tool is a standard build of FFmpeg. This tool can be configured quite well, but no interaction can be had from the server codebase once FFmpeg is started. This makes it impossible have FFmpeg stop gracefully upon request, instead having to kill it or wait for it to gracefully stop itself. This results in the last segment of a live stream not being available until FFmpeg decides that it has waited long enough for more (non-existing) RTP packets to arrive.

Using the FFmpeg C library allows for using utility functions for handling the video recording process. Implementing a pipeline which takes RTP as an input and outputs MPEG-DASH files (or another format) may take some work, but it allows for great control over all aspects of video recording. Documentation on the FFmpeg C libraries can be found on their website: https://www.ffmpeg.org/documentation.html.

13.2. User application

The user application runs in a web browser. This was chosen for the portable nature that web applications have, and for ease of development. After this choice was made, the realization came that the RTP live video stream can not be received in a web browser out of their security considerations. It would be possible with the addition of a proxy server, but it is unknown how much the latency of the live stream would be affected. The solution that is used now was very efficient in terms of development time; using DASH for both live streaming and playback allowed for recycling some components. However, since the latency of live streams in the system is ideally zero, being able to display an RTP stream would be desirable.

To achieve this, one may consider creating a user interface of a different nature, such as native or hybrid. A hybrid solution might be able to use most of the user interface created for this project and just swap out the live video player for one that takes RTP streams as an input.

13.3. Security

Even though security is not a requirement for components in the EIS because this is supposedly managed on a network level, it is still advisable to equip every sub-system with correct security measures. For the jury application, features of interest could include:

- User credentials for jury members;
- HTTPS over TLS for the REST API;
- Digitally signed verdicts;
- MQTT client authorization.

14. Versions

Date	Description	Version
02-03-2020	Initial draft	0.1
08-03-2020	- Described project planning in more detail. - Added mockups and use cases to the report.	0.2
17-03-2020	- Restructured the report in accordance with feedback by PO. - Updated writing style to a more formal one. - Added requirements. - Added relation between use cases and requirements. - Added descriptions of progress reports and task management. - Added system overview and decomposition. - Added research chapters.	0.3
18-03-2020	- Added to system overview. - Added to research chapters.	0.4
21-03-2020	- Added implementation chapter.	0.5
22-03-2020	- Added to the implementation of user application. - Reviewed by company and academic supervisors.	0.6
02-04-2020	- Partially implemented feedback on version 0.6. - Changed writing perspective to first person. - Updated captions on figures and tables. - Restructured report chapters, moving the focus of the report from “process” to “product”. - Sent to company supervisors for review.	0.7
05-04-2020	- Implemented remaining feedback on version 0.6. - Removed chapter on MPEG-DASH troubleshooting, referred to the external document instead. - Added chapter on integration and acceptance. - Added conclusion and recommendations.	1.0

15. Bibliography

RTP standard. (2003, July). Retrieved from IETF: https://tools.ietf.org/html/rfc3550

International Organization for Standardization. (2019, December). *ISO/IEC 23009-1:2019*. Retrieved from International Organization for Standardization: https://www.iso.org/standard/79329.html

HLS standard. (2017, August). Retrieved from IETF: https://tools.ietf.org/html/rfc8216

Real-time communication for the web. (n.d.). Retrieved from WebRTC: https://webrtc.org/

HTTP/1.1 standard. (1999, June). Retrieved from IETF: https://tools.ietf.org/html/rfc2616

Hypertext Transfer Protocol (HTTP/1.1): Range Requests. (2014, June). Retrieved from IETF: https://tools.ietf.org/html/rfc7233

Configure Apple HLS packetization. (2019, December). Retrieved from Wowza: https://www.wowza.com/docs/how-to-configure-apple-hls-packetization-cupertinostreaming

MPEG-DASH packetization. (2019, November). Retrieved from Wowza: https://www.wowza.com/docs/how-to-configure-mpeg-dash-packetization-mpegdashstreaming

Progressive download. (2020, January). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Progressive_download

Best Video Codec for HTML5 Live Video Streaming. (2018, March). Retrieved from dacast: https://www.dacast.com/blog/best-video-codec/

Find the closest number in an array. (n.d.). Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/find-closest-number-array/

Appendix A Use cases

Appendix A.1 Use case descriptions

Identifier	UC.1
Name	Identify as a jury member
Summary	Fill in information which uniquely links a user to a jury member.
Rationale	Linking a user to a jury member will allow the system to keep track of which jury member took what actions. For example; when a verdict is reached, it must be recorded who the jury member was who made that decision (**f.11**). The same applies to race control events.
Actors	User
Preconditions	N/A
Scenario	1. The user fills in information which uniquely identifies them. 2. The system publishes the information to the EIS.
Alternatives	N/A
Postconditions	- The user is identified as a jury member. - The jury member can access the rest of the system.

Identifier	UC.2
Name	View participants
Summary	View which participants are taking part in the event, and what their status is.
Rationale	Information such as knowing which participants are in the race facilitates taking action on a certain participant. For a jury member to execute any use case involving a participant, they must be able to find this participant in the system.
Actors	Jury member
Preconditions	N/A
Scenario	1. The jury member requests information about which participants are taking part in the race.
Alternatives	N/A
Postconditions	- Participant information is displayed to the jury member.

Identifier	UC.3
Name	Assign participant
Summary	Assign a participant to a jury member.
Rationale	In the military, each participant gets a jury member assigned to them. This jury member is responsible for judging this participant from start to finish. Every participant has one jury member, and a jury member can at most have one participant assigned to them. A jury member can choose to who they are assigned.
Actors	Jury member
Preconditions	- The participant is available. - The jury member is not yet assigned a participant.
Scenario	1. The jury member views which participants are available (**UC.2**); 2. The jury member selects a participant to assign to themselves.
Alternatives	2a. No participants are available; i. No participant is assigned to the jury member. ii. End of use case. 2b. The selected participant is no longer available; i. System informs user. ii. The participant is not assigned to the jury member. iii. Return to step 1.
Postconditions	- The participant is assigned to the jury member. - The participant is no longer available to be assigned.

Identifier	UC.4
Name	View live stream
Summary	View the live video footage of a participant crossing an obstacle.
Rationale	Viewing a live stream of a participant crossing an obstacle is key to the value proposition of this system. It allows jury members to see how a participant performs at an obstacle. This helps them make better judgements.
Actors	Jury member
Preconditions	- The participant is assigned to the jury member. - The EIS is broadcasting live video.
Scenario	1. The system displays live video footage of the participant.
Alternatives	N/A
Postconditions	N/A

Identifier	UC.5
Name	View tactical overview
Summary	View a tactical overview of horse monitoring data, showing a participant's position on a map and the heartrate and CEL of a horse.
Rationale	For a jury member, it is essential to be able to determine if the health of a horse is at risk because of exhaustion. If the CEL exceeds a certain threshold, the jury member will take action. Seeing the position of a participant provides the jury with context, which helps them reach their verdict.
Actors	Jury member
Preconditions	- The participant is assigned to the jury member. - The participant is in the race. - Horse monitoring data is available.
Scenario	1. The system displays the position of the participant on a map. 2. The system displays the heartrate and CEL of the horse.
Alternatives	N/A
Postconditions	N/A

Identifier	UC.6
Name	View video clip
Summary	View a recorded video clip.
Rationale	For a jury member to make a decision, they need detailed of information. Sometimes, live observations alone are not sufficient for a jury member to reach their verdict. In this case, the jury will review the recording. Horse monitoring data is also recorded, and a tactical overview is displayed next to the video recording. When a verdict is being reached by a jury member, the video clip provides evidence which justifies the decisions made.
Actors	Jury member
Preconditions	N/A
Scenario	1. The jury member requests a video clip. 2. The system plays the video recording. 3. The system displays a tactical overview, synchronized with the video recording.
Alternatives	1a. The video clip is not available. i. The system prompts the user to wait while the video recording is being made available. ii. When the video is available, proceed to step 2. 2a. The end of the video recording is reached. i. The system pauses the video on the first frame. ii. The tactical overview synchronizes accordingly. 2b. The jury member performs trick-play on the video clip (**UC.7**).
Postconditions	- The video clip is being displayed.

Identifier	UC.7
Name	Perform trick-play
Summary	Perform trick-play on a video clip.
Rationale	For a jury member, performing trick-play on a video clip helps to make better observations about a participant's performance, and therefore makes for a fairer judgement.
Actors	Jury member
Preconditions	- A video clip is being displayed.
Scenario	1. The system displays a trick-play control panel. 2. The user issues trick-play operations.
Alternatives	2a. The user does not issue any trick-play operations. i. The video clip plays at a normal speed.
Postconditions	- The video recording changed to the playback mode issued by the jury member. - The tactical overview synchronizes accordingly.

Identifier	UC.8
Name	Take a verdict
Summary	A jury member takes a verdict based on the observations of a participant.
Rationale	Having observed the participant, the jury member is well-informed and confident in taking a verdict. This decision can later be reviewed and compared to a video clip in case of a dispute.
Actors	Jury member
Preconditions	- A participant is assigned to the jury member. - The participant has crossed an obstacle. - The participant has not received judgement for crossing the obstacle.
Scenario	1. The jury member observes the performance of the participant at an obstacle. 2. The jury member enters their verdict into the system. 3. The system informs the jury member that the judgement process is completed.
Alternatives	1a. The jury member takes a verdict based on CEL observations. i. Proceed to step 2. 3a. The system informs the jury member that insufficient details have been entered. i. Return to step 2.
Postconditions	- The verdict is published to the EIS.

Identifier	UC.9
Name	Announce race control decision
Summary	Announce a jury member's decision regarding the continuation of the race.
Rationale	In case of a calamity or unforeseen event, jury members can make the decision to halt the race. When a race is resumed, they will also announce this.
Actors	Jury member
Preconditions	- The race is in progress.
Scenario	1. The jury member observes or is informed of a calamity. 2. The jury member makes an announcement regarding the continuation of the race. 3. The jury member announces the reason for taking their decision regarding the continuation of the race.
Alternatives	3a. The jury member does not announce a reason for taking their decision regarding the continuation of the race. i. End of use case.
Postconditions	- The decision regarding the continuation of the race is published to the EIS.

Identifier	UC.10
Name	View pending verdicts
Summary	View which obstacles have been crossed by an obstacle, but have not yet been subject to a verdict.
Rationale	A jury member must reach a verdict about the performance of a participant at every obstacle – even if the verdict is that the performance was flawless. To ensure this, it's necessary to know which performances have not yet been evaluated.
Actors	Jury member
Preconditions	- A participant is assigned to the jury member. - The participant has crossed an obstacle. - The participant has not received judgement for crossing the obstacle.
Scenario	1. The system notifies the jury member that a participant has not yet received judgement. 2. The jury member clicks on the "show pending verdicts" button. 3. The system displays a list of recordings of the performance of the participant at obstacles which have not been evaluated yet. 4. The jury member selects a recording. 5. The jury member views the corresponding video clip, conform **UC.6**.
Alternatives	2a. The list of recordings is empty. i. End of use case. 3a. The jury member does not select a recording. i. End of use case.
Postconditions	N/A

Appendix A.2 Requirements relationships

The tables below (Table 14 and Table 15) show the relation between requirements and use cases. This two-way relationship is listed in both directions for clarity.

Functional requirement	Fulfilled by use case(s)
r1	UC.1
r2	UC.3, UC.2
r3	UC.4, UC.5
r4	UC.5
r5	UC.6, UC.5, UC.7
r6	UC.7
r7	UC.8
r9	UC.9
r14	UC.5
r15	UC.10

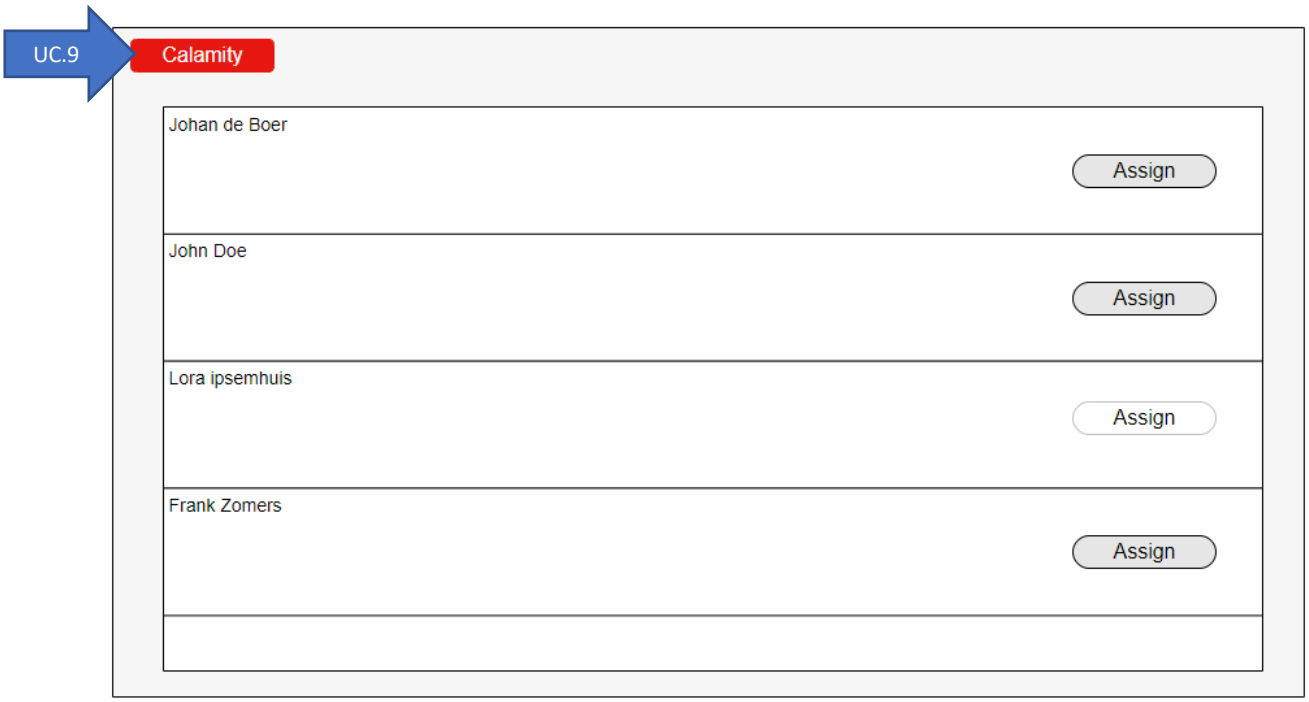
Table 14: Requirements versus Use Cases

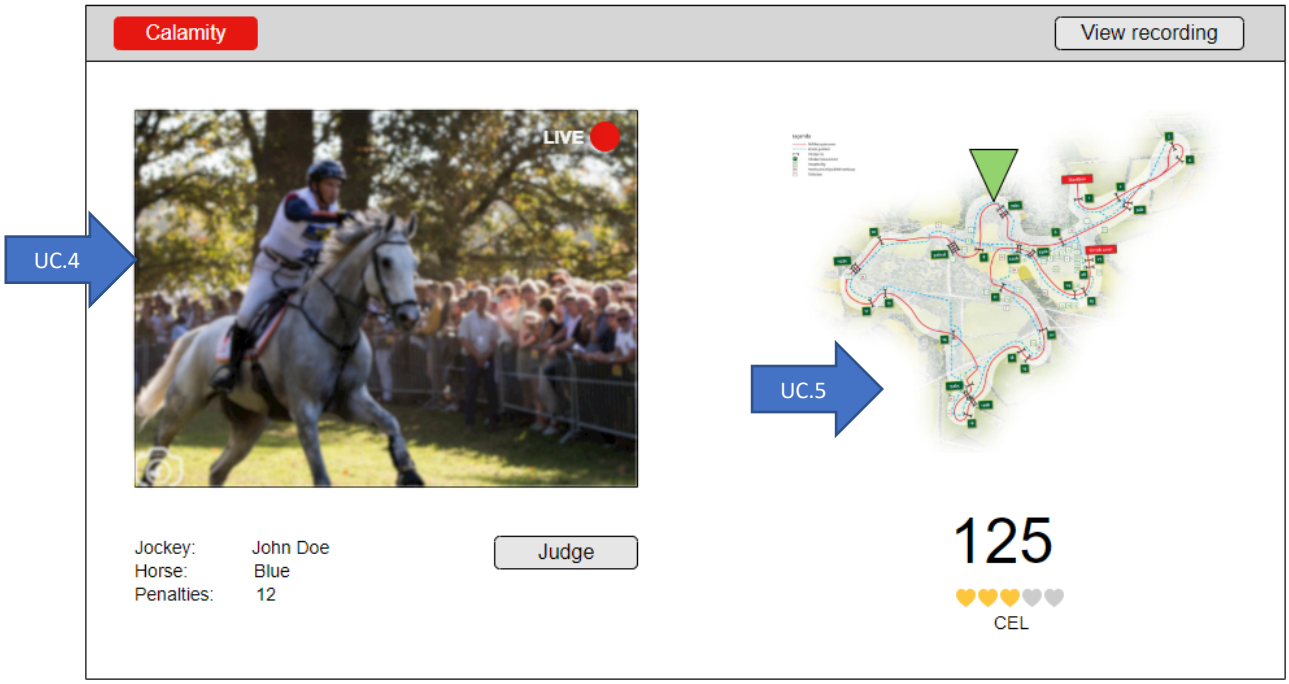
Use case	Fulfills / Contributes to
UC.1	r1
UC.2	r2
UC.3	r2
UC.4	r3
UC.5	r3, r4, r5, r14
UC.6	r5
UC.7	r5, r6
UC.8	r7
UC.9	r9
UC.10	r15

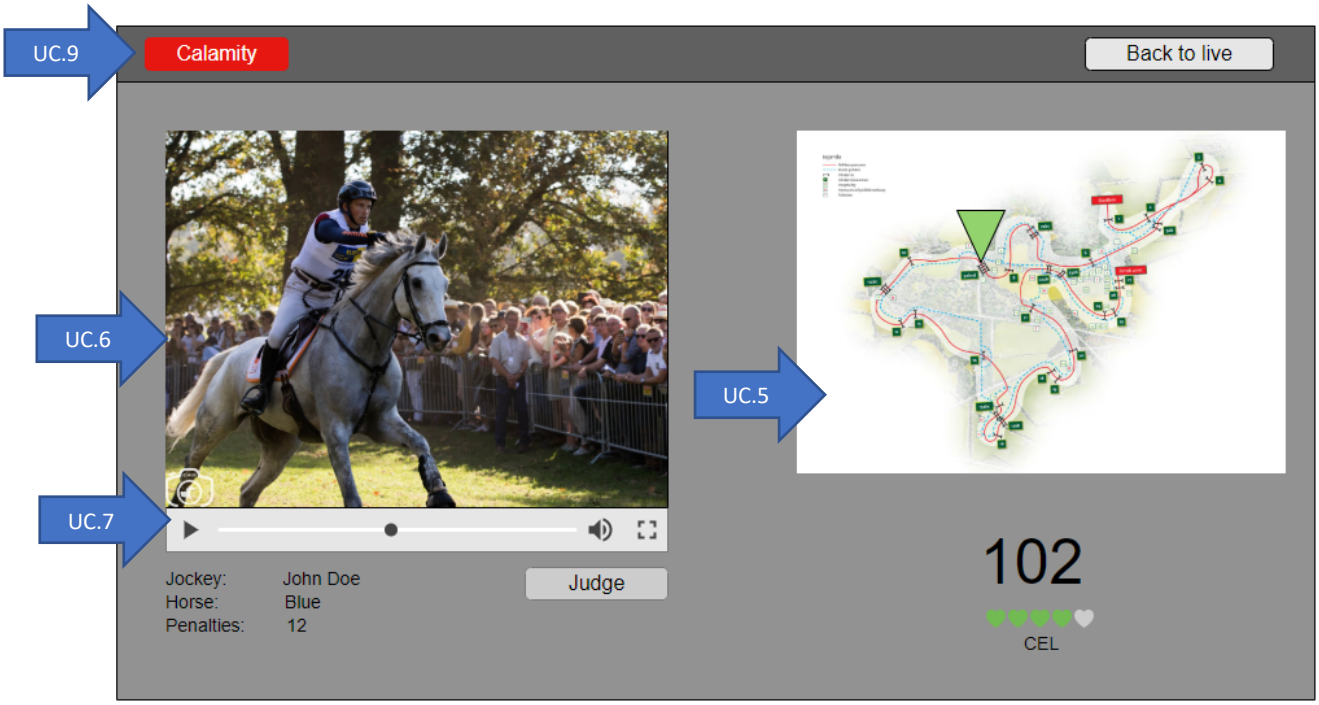
Table 15: Use Cases versus Requirements

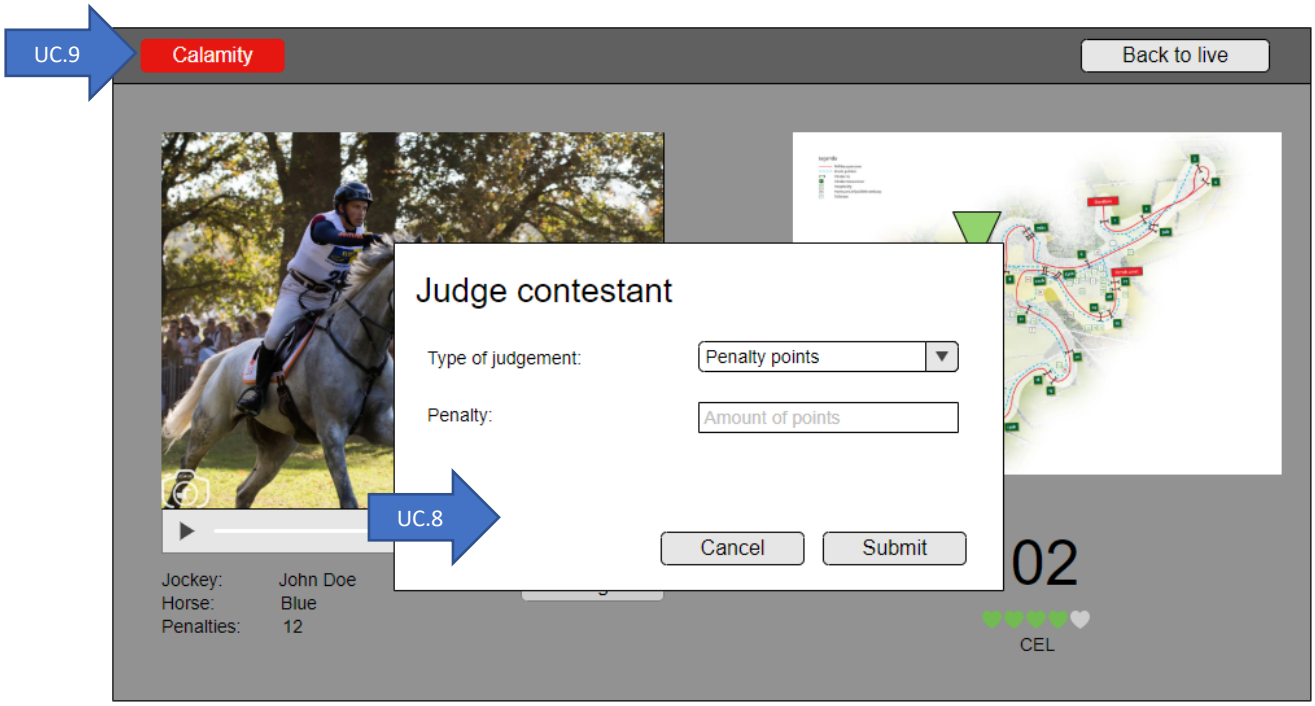
Appendix B Mockups

The mockups show the initial rough design of the user interface (UI) of the system. They have helped with communication during that critical period of finding out what the user needs and wants. The blue labels show to which use case the UI elements of the mockups relate.






UC.9
Calamity
Back to live
UC.6
UC.7
UC.5
Jockey: John Doe
Horse: Blue
Penalties: 12
Judge
102
CEL


UC.9
Calamity
Back to live
Judge contestant
Type of judgement:
Penalty points
Penalty:
Amount of points
UC.8
Cancel
Submit
Jockey: John Doe
Horse: Blue
Penalties: 12
02
CEL

Appendix C Database schema

The relational database schema of the jury application can be found in Table 16.

Table	Fields
position	id (serial), participant_id (int), time (big int), latitude (decimal), longitude (decimal)
heart_rate	id (serial), participant_id (int), time (big int), heart_rate (int)
verdict	id (serial), participant_id (int), obstacle_id (int), jury_id (int), penalty (int), description (varchar), time (big int)
participant_removal	id (serial), participant_id (int), jury_id (int), description (varchar), time (big int)
race_control	id (serial), race_status (int), jury_id (int), description (varchar), time (big int)
video_clip	id (serial), file_uri (varchar), participant_id (int), obstacle_id (int), status (int), time (big int)
approach	id (serial), participant_id (int), obstacle_id (int), time (big int)
leave	id (serial), participant_id (int), obstacle_id (int), time (big int)

Table 16: Relational database schema

Appendix D API endpoints

This appendix contains the REST API endpoints specification for the jury application. All the endpoints are GET requests and used to obtain information. No information can be posted to the system through HTTP. Timestamps are in milliseconds in the UNIX epoch.

/api/videos	
Returns information about all video clips. The file URI path in the response body does not specify a host. Response may be an empty JSON array.	
Request parameters	**Response body**
-	[{ “participantId”: int, “obstacleId”: int, “fileURI”: string, “status”: int, “time”: long }, …]

/api/videos/{participant_id}	
Returns information about video clips relating to the specified participant. The file URI path in the response body does not specify a host. Response may be an empty JSON array.	
Request parameters	**Response body**
-	[{ “participantId”: int, “obstacleId”: int, “fileURI”: string, “status”: int, “time”: long }, …]

/api/videos/{participant_id}/{obstacle_id}	
Returns information about a single video clip relating to the specified participant and obstacle. The file URI path in the response body does not specify a host. May result in a HTTP 404 response code if there is no video clip information relating to the specified participant and obstacle. If multiple records were found, this endpoint returns the first result.	
Request parameters	**Response body**

-	{
	“participantId”: int,
	“obstacleId”: int,
	“fileURI”: string,
	“status”: int,
	“time”: long
	}

/api/positions/{participant_id}	
Returns a list of positions relating to a specified participant. By default (from = 0, until = *now*) returns all position data available. Optionally returns only position data within a specified time range. Response may be an empty JSON array.	
Request parameters	**Response body**
- **from=timestamp** ◦ **[integer]** ◦ **[optional, default=0]** - **until=timestamp** ◦ **[integer]** ◦ **[optional, default=now]**	[{ “participantId”: int, “lat”: float, “lon”: float, “time”: long }, …]

/api/heart_rates/{participant_id}	
Returns a list of heart rate measurements relating to a specified participant. By default (from = 0, until = *now*) returns all heart rate data available. Optionally returns only heart rate data within a specified time range. Response may be an empty JSON array.	
Request parameters	**Response body**
- **from=timestamp** ◦ **[integer]** ◦ **[optional, default=0]** - **until=timestamp** ◦ **[integer]** ◦ **[optional, default=now]**	[{ “participantId”: int, “heartrate”: int, “time”: long }, …]

/api/verdicts/{participant_id}	
Returns a list of verdicts relating to a specified participant. By default (from = 0, until = *now*) returns all verdicts available. Optionally returns only verdicts within a specified time range. Response may be an empty JSON array.	
Request parameters	**Response body**
- **from=timestamp** ◦ **[integer]** ◦ **[optional, default=0]** - **until=timestamp** ◦ **[integer]** ◦ **[optional, default=now]**	[{ “participantId”: int, “obstacleId”: int, “juryId”, int, “penalty”: int, “description”: string, “time”: long }, …]

/api/verdicts/{participant_id}/{obstacle_id}	
Returns a list of verdicts relating to a specified participant and obstacle. By default (from = 0, until = *now*) returns all verdicts available. Optionally returns only verdicts within a specified time range. Response may be an empty JSON array.	
Request parameters	**Response body**
- **from=timestamp** ◦ **[integer]** ◦ **[optional, default=0]** - **until=timestamp** ◦ **[integer]** ◦ **[optional, default=now]**	[{ “participantId”: int, “obstacleId”: int, “juryId”, int, “penalty”: int, “description”: string, “time”: long }, …]

/api/verdicts/{participant_id}/pending	
Returns a list of pending verdicts relating to a specified participant. By default (from = 0, until = *now*) returns all pending verdicts available. Optionally returns only pending verdicts within a specified time range. Response may be an empty JSON array.	
Request parameters	**Response body**
- **from=timestamp** ◦ **[integer]** ◦ **[optional, default=0]** - **until=timestamp** ◦ **[integer]** ◦ **[optional, default=now]**	[{ “participantId”: int, “obstacleId”: int, “time”: long }, …]

/api/participant_removal/{participant_id}	
Returns a list of participant removal events relating to a specified participant. Returns all participant removal events available. Response may be an empty JSON array.	
Request parameters	**Response body**
-	[{ “participantId”: int, “juryId”, int, “description”: string, “time”: long }, …]

/api/race_control	
Returns a list of race control events relating to a specified participant. Returns all race control events available. Response may be an empty JSON array.	
Request parameters	**Response body**
-	[{ “juryId”, int, “raceStatus”, int, “description”: string, “time”: long }, …]

/api/race_control/latest	
Returns a list of race control events relating to a specified participant. Returns all race control events available.	
Request parameters	**Response body**

-	{
	“juryId”, int,
	“raceStatus”, int,
	“description”: string,
	“time”: long
	}

Appendix E Configuration variables

Server database configuration variables are shown in table Table 17.

Variable	Description
spring.datasource.url	The URL of the database to connect to. This database must contain the schema specified in chapter 9.1.2: Application database. This takes the form of "jdbc:**{protocol}**://**{host}**:**{port}**/**{database}**", i.e.: "jdbc:postgresql://127.0.0.1:5432/jury_application".
spring.datasource.username	A the username of a user with read/write access to the database tables specified in chapter 9.1.2: Application database.
spring.datasource.password	The user's password.
spring.datasource.driver-class-name	The Spring database driver used by Hibernate for connecting to the database (default is org.postgresql.Driver).

Table 17: Server database configuration variables

Client configuration variables are shown in Table 18.

Variable	Default value	Description
api.method	http	The REST API communication protocol.
api.host	127.0.0.1	The host address of the REST API.
api.port	9090	The network port of the REST API.
mqtt.host	127.0.0.1	The host address of the EIS MQTT broker.
mqtt.port	8080	The WebSocket port of the EIS MQTT broker.
videoPlayer.segmentBufferSize	4	The number of segments the video player should buffer before playing a video. Affects streaming latency.
videoPlayer.liveDelay	2	The target amount of latency in seconds the video player should have in a live stream. Using a value that is too low may result in choppy video.
videoPlayer.rewindScale	2.0	The timescale for rewinding and fast-forwarding video with trick-play.
videoPlayer.rewindFPS	12	The target framerate of the video during rewinding. A higher framerate requires more computer resources and too high of a value may cause choppy video, depending on the host machine.
tacticalOverview.mapRoot	map/ military_boekelo	The path to the root folder containing map tile images. This path is relative to the "public", "html" or "www" webserver directory.
tacticalOverview.locationCenter	[52.196265, 6.809011]	The geolocation on which the map centers. Value is an array of two numbers

		representing latitude and longitude respectively. Default value centers on the Military Boekelo cross-country track.
tacticalOverview.zoom	14	The starting zoom level of the map.
tacticalOverview.buffer	20	The dynamic HM data buffer growth in seconds. The tactical overview will load this amount of time worth of HM data.
verdictForm.maxPenalty	100	The maximum penalty a jury member can issue when publishing a verdict.

Table 18: User application configuration variables

Appendix F UI component properties

Table 19 shows the input properties of the Live Video Player component. Table 20 shows the input properties on the Playback Video Player component.

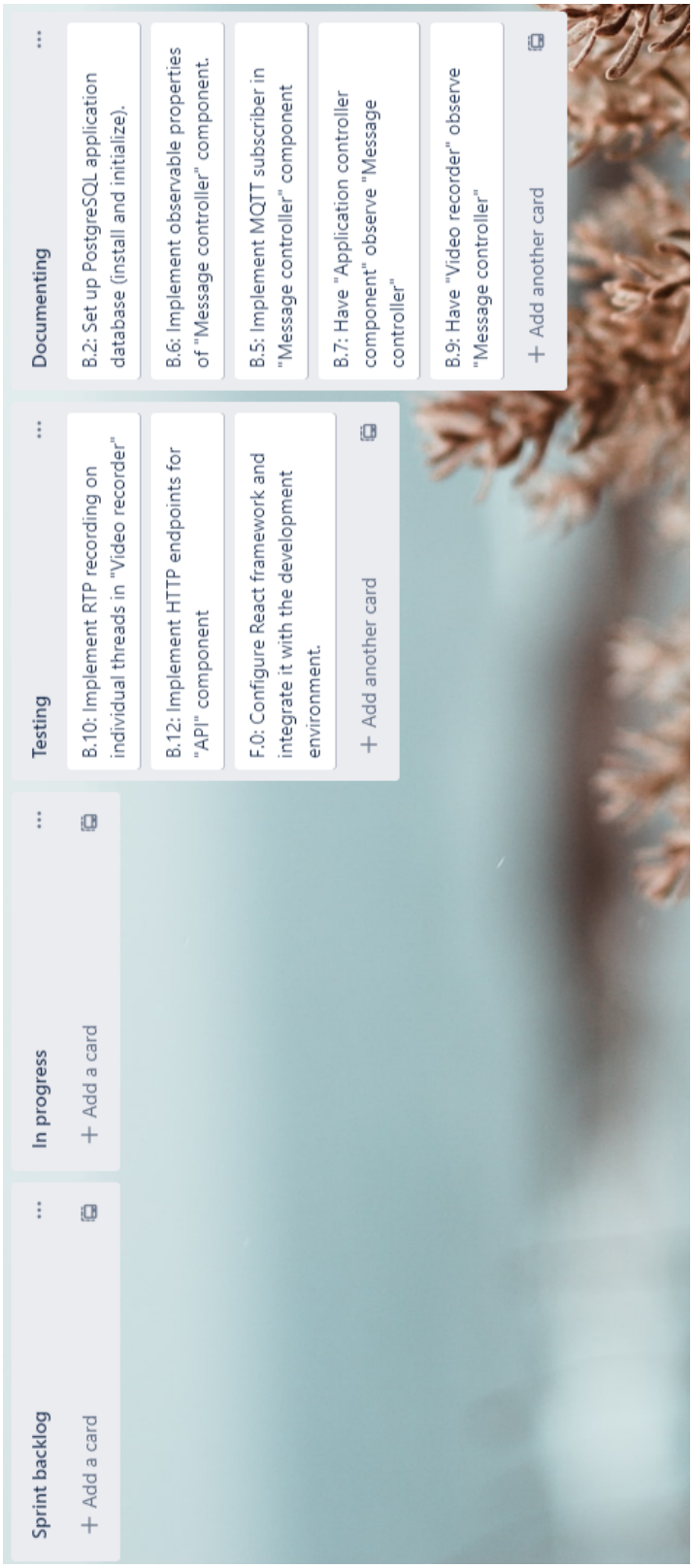
Property	Description
participantId	The unique identifier specifying which participant is monitored by the user.
obstacleId	The unique identifier specifying which obstacle is being approached or crossed by the participant.
onTimeUpdate	An event handler (function) which gets called every time the child Video Player component updates its current playing time in the UNIX epoch in milliseconds. This function gets passed three values (all in UNIX time): start time, current time, and end time.
onSyncStart	An event handler (function) which gets called when a live stream starts playing. This means that any component that wants to synchronize with the Live Video Player should start listening to onTimeUpdate.
onSyncEnd	An event handler (function) which gets called when a live stream stops playing. This means that any component which wants to be in sync with the Live Video Player should “stop”.

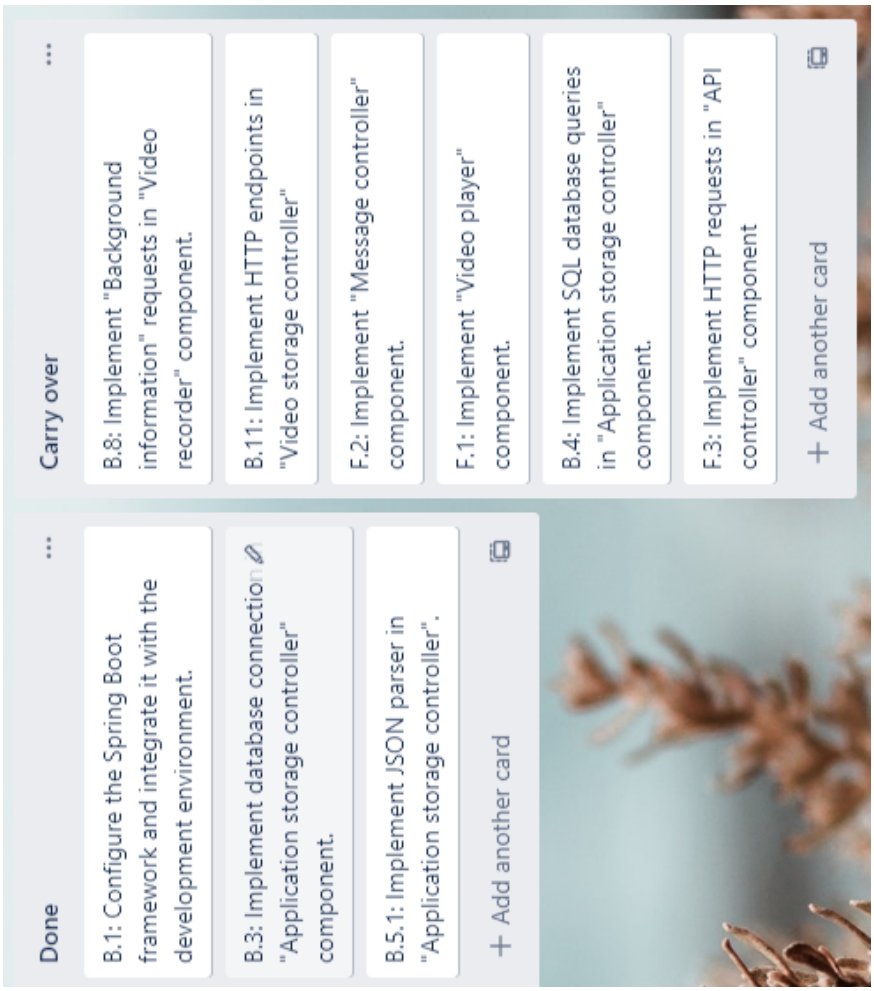
Table 19: Input properties of the Live Video Player component

Property	Description
participantId	The unique identifier specifying which participant is monitored by the user.
obstacleId	The unique identifier specifying of which obstacle the participant’s performance is being played back.
onTimeUpdate	An event handler (function) which gets called every time the child Video Player component updates its current playing time in the UNIX epoch in milliseconds. This function gets passed three values (all in UNIX time): start time, current time, and end time.

Table 20: Input properties of the Playback Video Player component

Appendix G Sprint backlogs




Done
B.1: Configure the Spring Boot framework and integrate it with the development environment.
B.3: Implement database connection "Application storage controller" component.
B.5.1: Implement JSON parser in "Application storage controller".
+ Add another card
Carry over
B.8: Implement "Background information" requests in "Video recorder" component.
B.11: Implement HTTP endpoints in "Video storage controller"
F.2: Implement "Message controller" component.
F.1: Implement "Video player" component.
B.4: Implement SQL database queries in "Application storage controller" component.
F.3: Implement HTTP requests in "API controller" component
+ Add another card