Data Management System Proposal

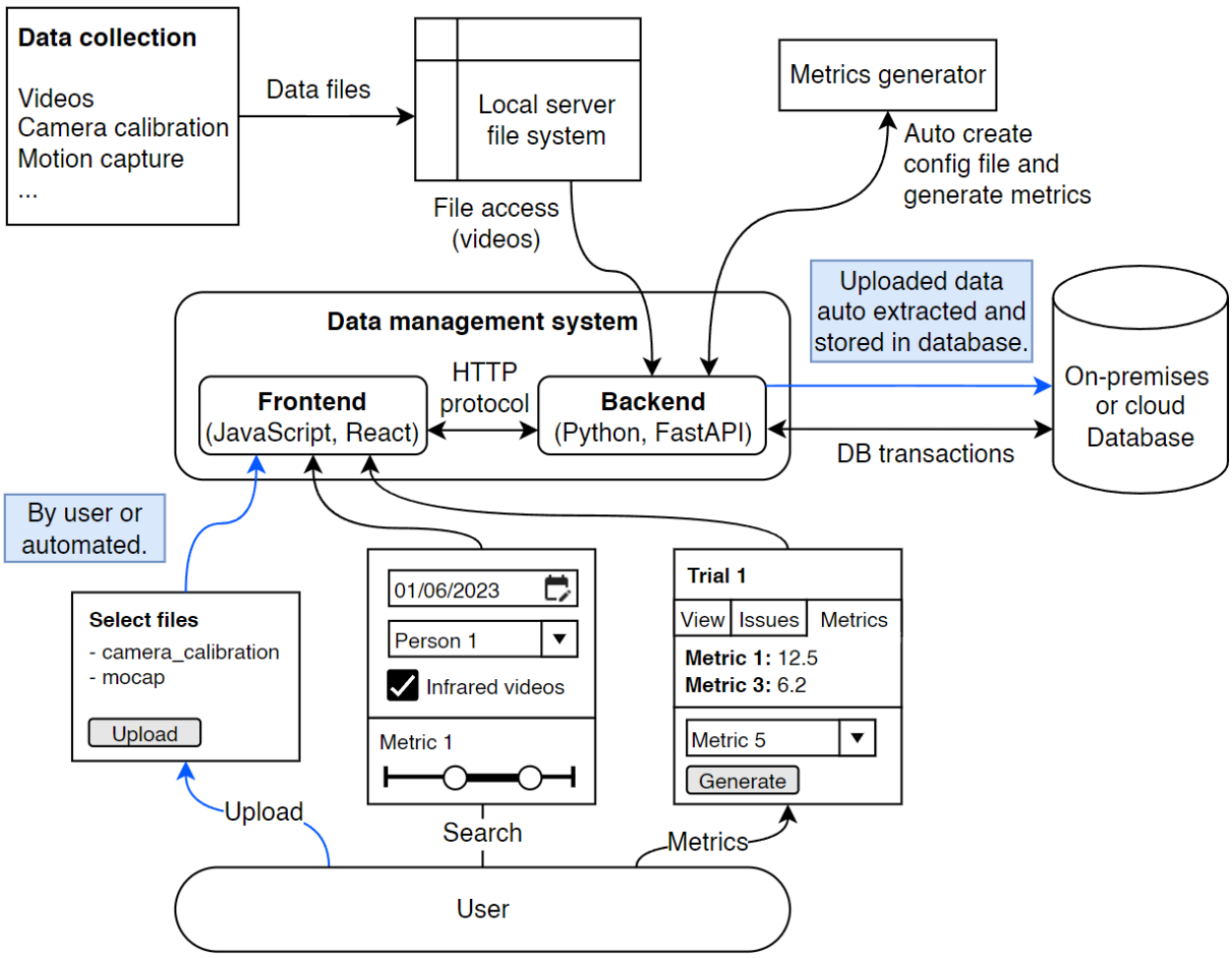
Overview

I would like to propose an on-premises internal data management (DM) system, that serves as a central data hub that machine learning (ML) engineers (users) can use, which facilitates the storage, retrieval, and analysis of the data collected during motion capture trials. The DM system will provide a seamless and efficient interactive experience for users to perform various DM tasks, including querying trials, viewing trial data, generating metrics, and exporting relevant trial data in multiple formats for the user's exact needs. There should be no need for direct interaction with the file system, as all the raw data, for example, the camera calibration and motion capture files data will be automatically extracted and imported into a structured database, enabling powerful querying and data processing ability. All DM tasks will be streamlined and conveniently carried out through the data DM system using a practical intuitive user-friendly graphical user interface (GUI), tailored specifically for the ML engineer's use cases.

A possible rough workflow and data flow might look like this:

1. Data collection: Camera calibration and motion capture files are created as well as any other data for a given trial.
2. Data upload and storage: The files from the data collection stage are uploaded to the database using the DM system, where the data will be automatically extracted to a compatible format to import into the database. The upload of trial files can either be done by a user through the DM system GUI or be a fully automated process.
3. Data view and retrieval: At this stage, all the raw data for all trials, including the one just uploaded, is in a structured database for easy processing and data retrieval, which allows users through the DM system GUI, to view trial data, query trials, generate metrics, and export trials in various formats as needed.

Below is a simple illustration of how the DM system might possibly be integrated based on the brief task description.



Proposal summary

Before diving into the details, here is a summary of the key elements of the proposal for your convenience.

Storage solution for the raw data

Currently, the data is stored in a file system, which is limiting and inefficient for querying, processing and working with the data. To overcome this, a more suitable solution is to store the data in a structured and organised database. Initially, a relational SQL-based database was considered, but it was quickly found out that it would be extremely impractical due to all the inapplicable relationships and scalability issues especially when dealing with data such as motion capture. So for the proposed system, I propose a NoSQL database, specifically a document-based database, which provides flexibility, adaptability, and scalability for storing motion capture data for an on-premises DM system. By transferring the raw data to a database and utilising a structured document-based format, the data becomes easily accessible, queryable

at any level efficiently, and ready for further processing within the proposed DM system or any other applications needed.

Existing technologies used in the proposed system

The proposed DM system is based on a simple full-stack web application architecture, aiming for efficiency, maintainability, and a practical GUI. The frontend utilises JavaScript, for its wide adaptation, compatibility, and extensive ecosystem, with React as the GUI development framework for its component-based architecture and versatility. On the backend, Python is selected for its widespread use in ML applications, and will possibly provide seamless integration with already existing internal tools. FastAPI is chosen as the backend server API framework due to its high performance and seamless integration with Python's ecosystem. By combining only these four technologies, a powerful, expandable, and high-performance DM system can be developed. Accommodating not only the required applications for the given task, but also provides immense potential for future expansion to meet the exact evolving needs of the ML engineers. Please note, these technologies were solely chosen based on the brief task description and may completely change after deeper understanding of the data, processes and system requirements.

Metrics generation and storage

In the proposed DM system, generating and storing metrics for a trial can be streamlined and fully automated. Through the GUI, the DM system allows users to simply choose a metric from a selection menu in a specific trial view. The system then sends a request to the backend, which uses the necessary data from the connected database to generate the selected metric using the existing program that generates the metrics. The result is automatically stored in the database and displayed in the GUI for the user alongside other generated metrics for the trial. This eliminates the need for the config file setup by the user and ensures persistent metrics for future reference and eliminates re-generations of metrics for the same trial. Additionally, the system can offer options like bulk metric generation and exporting metrics in specific formats. This approach simplifies and assists the metric generation and storing process significantly.

Search and data retrieval

The proposed DM system offers intuitive and efficient search functionality for retrieving specific data. By storing the raw data in a structured database, using the DM system GUI, users can interactively search for trials using various filters such as date range, certain person presence, video type, and metric value ranges. The system translates the user-selected filters into queries which are executed by the database, providing real-time search results. Users may choose to explore specific searched trials or export all queried trials or a subset of the data in a variety of

predefined and customisable formats to meet their exact needs. The proposed search platform enhances the efficiency of data retrieval and improves the user's workflow and overall user experience within the DM system.

These four main topics are all further expanded upon in their corresponding section of the proposal.

Storage

In this section, I go over how the raw data may be stored in the proposed DM system. Currently all the data as described is stored on the local server in a file system. Having the data in a file format makes it extremely limiting and inefficient to query and work with the data itself, it is not an optimal way of storing such data. If needed, a DM system can still be developed that directly interacts with the file system without any new infrastructure. However, querying and processing will be very limited and extremely inefficient and hard to scale, as it will be bound to fixed file formats and file system structure, and will be burdensome to adapt to new or changing data. The most important aspect of the DM system is to allow users to query trials based on data within a trial, so it is important for a DM system to allow a high-performance robust way of querying data.

A more suitable and feasible solution is to transfer and store the raw data in a structured organised database, where all the inner data within the files (camera calibration and mocap files for example) are directly stored in the database. This would allow much more efficient and powerful querying ability across all the data, and allow fine-tuned queries based on any small piece of the data very easily. Not only will querying ability be much better, but data organisation, data processing, and data extraction will be much easier to perform as all data is stored as individual data points in the database and can be easily accessed.

So now the question is, what type of database, as there are many database solutions to consider. Most databases can be categorised as either SQL or NoSQL. I initially considered a relational SQL database, however, after further evaluation, I realised that it would not be feasible or practical for storing all the motion data for all the trials in that way. The overworked relationships between the tables would result in possibly thousands of rows across multiple tables for a single trial, creating a cumbersome structure to maintain and evolve. This would not only hinder scalability but also impose strict adherence to a fixed scheme, adding overhead and making it very difficult to accommodate any changes to the data or for new data forms in the future. The use of an SQL-based database in this context would quickly become impractical and troublesome to manage.

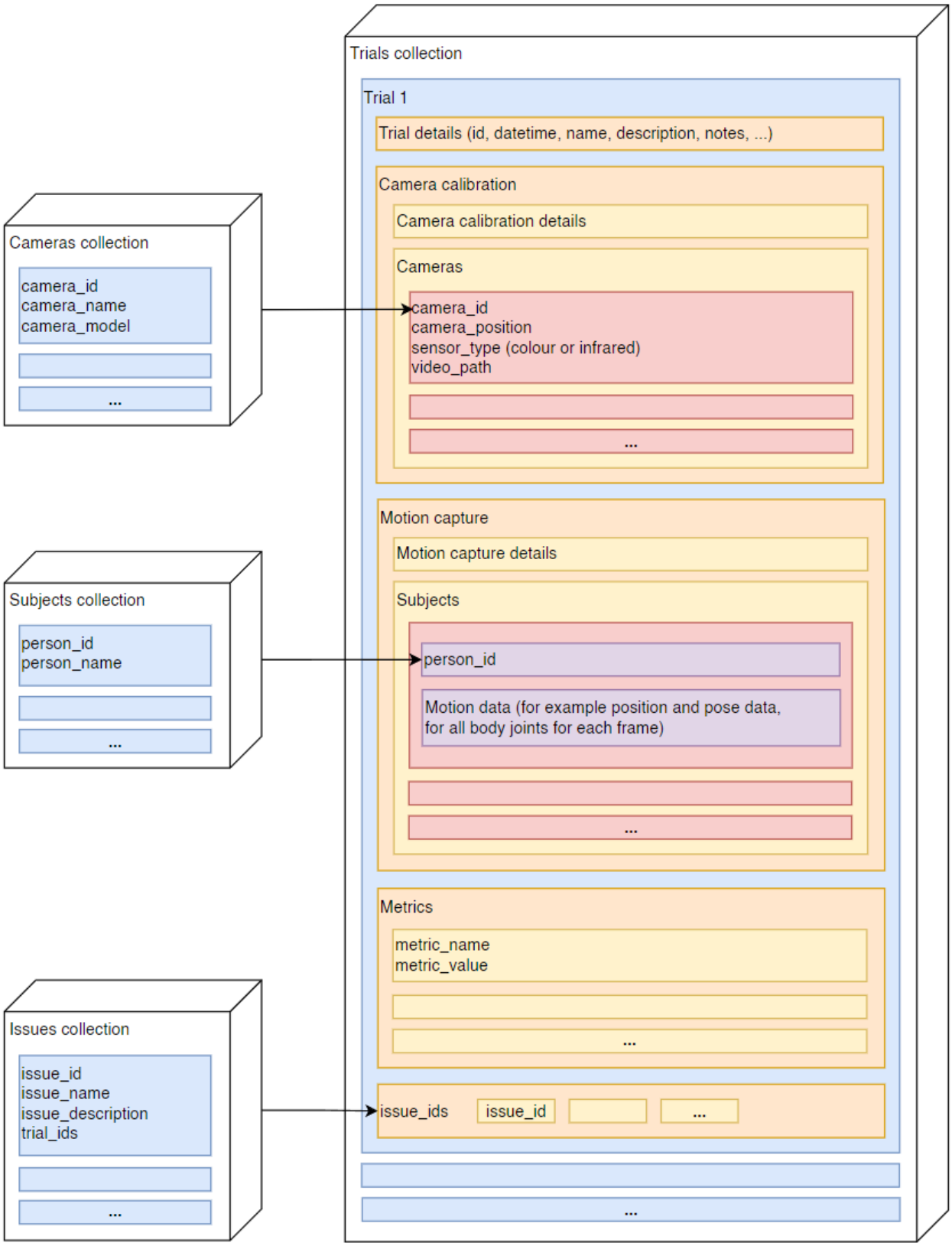
An SQL database would be a great option for certain types of data, however, it is not practical for motion capture data that may include for example a large number of joint positions and rotations

for just a single timestep, where in turn you may have thousands of timesteps for a single trial. Storing such data across multiple tables is not constructive. However, if a relational SQL-based system is required, it might be feasible, if only the file paths are stored in the database for the files stored in the local server. In this way, an SQL-based solution could work, and in that case, these are some great options to consider, MySQL PostgreSQL, and Oracle Database. However, this will greatly limit the DM system capability and potential, as well as the querying ability as you are not able to query the data within the files efficiently or effectively.

This naturally led me towards a NoSQL database solution, and this is what is used for the proposed DM system as the storage solution for the raw data. In a NoSQL document-based database, all data related to a trial can be stored within the same document, eliminating the need for trial data to be spread across multiple tables. This design allows for easy adaptability to changes in the data for different trials, or additions and modifications to the data in the future. Additionally, NoSQL databases excel in horizontal scalability, as trials can be treated as separate documents that can be distributed across multiple shards, allowing long-term and stable performance as data grows. With the flexibility of a NoSQL database, there is no strict schema enforcement to maintain as the data evolves, yet data can still adhere to a general structure.

There are many NoSQL databases to choose from, such as Google Firebase Firestore Database, Amazon Web Services (AWS) DynamoDB, MongoDB, and Apache Cassandra. Now some of these options are solely cloud-based which is not suitable for an on-premises DM system solution. For this proposal, I have chosen to go with the MongoDB NoSQL database, as it supports a production-level local environment setup, powerful querying capabilities, and indexing for improved query performance, which is especially needed when dealing with large datasets such as motion capture data.

Based on the brief task description, below is an illustration of one possible way of how the collections for the data might look for storing data related to a trial in a NoSQL document-based database. For a more detailed possible structure of the data, please refer to this JSON format of the same structure or the Pydantic model representation as an example.


Trials collection
Trial 1
Trial details (id, datetime, name, description, notes, ...)
Camera calibration
Camera calibration details
Cameras
camera_id
camera_position
sensor_type (colour or infrared)
video_path
...
Motion capture
Motion capture details
Subjects
person_id
Motion data (for example position and pose data, for all body joints for each frame)
...
Metrics
metric_name
metric_value
...
issue_ids
issue_id
...
...
Cameras collection
camera_id
camera_name
camera_model
...
Subjects collection
person_id
person_name
...
Issues collection
issue_id
issue_name
issue_description
trial_ids
...

Using a document-based NoSQL database, all the current raw data from for example the camera calibration and mocap files, as well as any other data, can be automatically extracted and stored in the database through the DM system GUI. A data validation pipeline between the data collection and the database can also be implemented to ensure data consistency and data quality is maintained, and only appropriate and correct data is stored in the database to possibly detect errors or anomalies within the collected data.

In summary, for how the raw data should be stored in the proposed DM system. Using a document-based NoSQL database, then to initially populate the database, all the current raw data can be automatically extracted and transferred to a format that is compatible with the database, using for example, the camera calibration and motion capture files. In terms of the videos, they can be stored on the local server as they are now, with video path references available in the database. Then from there on, the possible workflow discussed in the overview section might be conducted for new data collected. New data can be simply uploaded to the DM system through the GUI by a user, or done by an automated process, by accessing the files directly from the local server file system after trial data has been collected and files added to the server. Ultimately storing all the data in an organised and structured format in the database, where it will be readily and easily accessible for querying and processing for any DM task.

Existing technologies

Provided that the data is transferred to a database as proposed in the storage section, there are many existing out-of-the-box DM systems which can be used for querying and exporting data. Systems such as MongoDB Atlas/Compass, Amazon Redshift, and Studio 3T. As well as Amazon Athena, which does not require the data to be transferred to a database, it operates with the files directly. They all offer querying and exporting capabilities, and this might be enough and what is needed. However, ML engineers will be very limited by the lack of specific functionality, flexibility, integration, and control. Hence why I am proposing an internal system custom-built from the ground up, which will be completely tailored and designed specifically with the ML engineers in mind. Made to maximise productivity, with seamless integration when it comes to managing, processing, querying, and using the data at full capacity. Whilst still providing full control, eliminating reliance on third parties, and allowing iterative expansion to create a powerful internal data hub to meet the current and future DM requirements.

Technologies used in the proposed system

The proposed DM system is based on a simple full-stack web application architecture that can possibly be hosted on the same current local server. It is made up of small core building blocks, to use the least amount of frameworks possible, to make the system stack as lean as possible.

Using the bare minimum tools and frameworks for easy maintainability, expansion, and maximum performance, whilst still having an attractive and practical GUI. Below is the set of existing technologies used in the proposed DM system, separated into frontend and backend.

Frontend

- **JavaScript** is chosen for frontend language for its wide adaptation, compatibility across different browsers/devices, extensive ecosystem, easy integration with various APIs, plus many other reasons, after all, it is JavaScript... TypeScript is also a consideration, which is also an excellent choice for the frontend language for its static-typed nature.
- **React** is chosen for the main GUI development framework for its reusable component-based architecture and performance, allowing the DM system to remain consistent, easy to learn, and highly adaptable.

Backend

- **Python** was an obvious choice for the backend language, as it is most likely one of the main languages used for the internal tools at Vicon, and is widely used in ML applications. This would make integrating any existing internal tools to the DM system much more coherent, for example, the Python scripts currently used to extract data from the raw data files.
- **FastAPI** is chosen for the backend server API framework for its integration with Python's native type hints through Pydantic, contributing to making it one of the fastest API frameworks, which for the proposed DM system I am aiming for maximum performance. FastAPI also offers comprehensive and well-organised documentation. Its high performance, great documentation, and great support for document-based databases, makes FastAPI an ideal choice.

Using these simple building blocks, a very powerful internal DM web application system can be created that can be developed in any way required. However, with a better and deeper understanding of how data flows and processes within Vicon, a whole different tech stack might be needed. But these are the general technologies that can be used to accommodate all the applications needed for this given task, which also allows great potential for expansion and full flexibility for a DM system.

I think it is worth noting that a DM system purely based on Python was considered, using for example a GUI library. However, the GUI options will be very limited and not be able to accommodate comprehensive UI elements, will not be as easily accessible on different platforms and devices, and possibly be a bottleneck for the DM system's potential. Hence why I went with the web-based application route, since I believe a comprehensive, versatile, and easy-to-learn GUI in a DM system is extremely important for maximum practicality and ease of use.

Metrics generation and storage

In this section, I will go over how a user might use the DM system to generate and store metrics for a given trial. From the way I have understood it, the process of generating metrics currently involves the user setting up a config file that contains the required data to generate the specified metrics using a C++ command line executable program. In the proposed DM system, this process can be standardised and automated.

For example, after the user has navigated to a particular trial using the DM system GUI, in the trial view, there could be a selection menu, where the user can select a metric to generate from all the available metrics to generate. The DM system would then send a request to the backend, and would use the same C++ command line executable program to generate that specific metric. The backend will use the subset of data it needs from the trial using the connected database, since the data needed to generate each metric would be known. There would be no need for a config file from the user, since for each metric the data needed to generate the metric can be directly accessed from the database, hence the config file can be automatically generated by the backend system, which is fed into the program that generates the metrics.

The result from the program that generates the metrics, is then optionally automatically stored in the database in the corresponding trial document in the appropriate location. With the result also sent to the frontend, to be displayed to the user in the DM system GUI, probably in a simple visualisation or in any specific form the metric may have. The main reasons why the metrics are stored in the database are so the metrics are persistent and the same metric does not have to be re-generated for the same trial, as well as to allow all users who may navigate to the same trial in the future, to view all the generated metrics for that trial.

In summary of how a user may generate metrics for a trial, focusing only on the user point of view. The user navigates to a trial using the DM system GUI, for example through a search, the user selects a metric to generate from a selection menu, the GUI will then update to include the metric results visually depending on the form of the metric, alongside all other generated metrics for that trial. In addition to the selection of the metric to generate, there could be other options the user can select that might be added as per the user's needs, for example, generate metrics in bulk, or for a subset of trials selected, etc. The user may also have the option to download all or a subset of the generated metrics in a specific format. This is just one possible way, how a user may generate and store metrics for trials in the proposed DM system.

Search and data retrieval

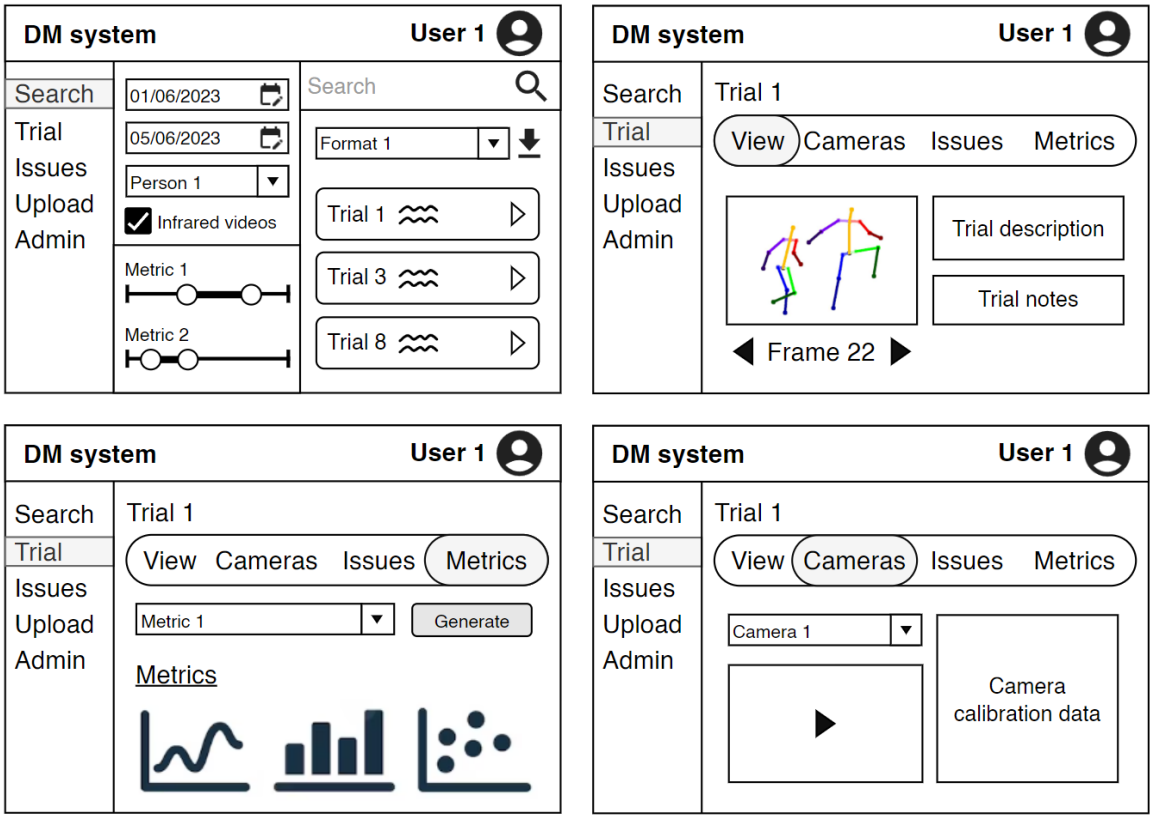
In this section, I will go over how a user might execute searches for trials and how they can retrieve the data using the proposed DM system. Since the raw data in the proposed DM system is stored in a database, searching for trials using any piece of data is now extremely easy and efficient.

All the user has to do is open up the DM system, and use interactive visual querying tools to search all trials based on for example, date range, a certain person, video type, and any metric range value, as well as any other filters that the ML engineer may need. Such system can be easily expanded and extended for a very powerful querying platform. The DM system will use all the interactive filters set by the user, to send requests to the backend, where it will simply convert all selected filters to a query format that can be executed by the database to ultimately obtain the search results. The search result is then communicated back to the frontend, to display to the user all the relevant trials in real-time as the search filters are changed.

From there, the user may choose to explore a specific trial by simply navigating to the trial view for a specific trial in the search result. Or the user may choose to export all the trials that are part of the search. There could be a number of format options the user can choose from to export the data, it could be for example JSON, XML, predefined formats agreed upon, simply the paths to all the data files stored in the local server, or any subset of the trial data. The export format options can be greatly customised to meet the exact needs of the ML engineers to make their workflow as efficient as possible. Since I believe that you may have an extremely large number of trials to query, to ensure the system operates as efficiently as possible and ensure minimal loading times, techniques such as lazy loading or pagination can be implemented for smooth consistent operation.

System wireframes

Below are some examples of the GUI wireframes of how the system might look like based on the task description, ideally this would be closely defined with the machine learning engineers to ensure it is as easy to use and as useful as possible for their specific tasks.



Conclusion

In conclusion, the proposed on-premises internal data management system aims to create a central data hub for automatically handling data management tasks, specifically designed for the machine learning engineers and their workflow. Allowing users to easily and efficiently perform DM tasks such as, querying trials, exporting relevant trials, and generating metrics, all intuitively through a practical GUI. The mission is to streamline DM processes and make it easier for ML engineers to access and work with the raw data, while adhering to best practices and ensuring a solid foundation, for continuous iterative expansion to adapt and meet the evolving needs of the ML engineers and DM tasks in general through feedback or any other means. It is important to note that the proposed DM system is based on the brief task description and may not fully capture the complexity of the actual data and processes. A deeper understanding of the data and processes will enable the development of a much more robust and effective DM system, that might require a completely different set of solutions and technologies to meet the exact system requirements. I believe team collaboration, creativity, and versatility will be key in successfully developing such system, and I am excited about the potential to be able to contribute to enhancing data management processes, ML engineer's workflow, and ultimately the overall project success at Vicon if this is a snapshot of what the role involves!