BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

ONLINE VIDEO ANNOTATION TOOL

NÁSTROJ PRO ONLINE ANOTACE VIDEA

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **NIKITA MOISEEV**
AUTOR PRÁCE

SUPERVISOR **Ing. ONDŘEJ KLÍMA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

156724

Institut:	Department of Computer Graphics and Multimedia (DCGM)
Student:	**Moiseev Nikita**
Programme:	Information Technology
Title:	**Online video annotation tool**
Category:	Web applications
Academic year:	2023/24

Assignment:

1. Get familiar with existing online applications for object annotation in traffic videos.
2. Study current frameworks and libraries for building custom web applications.
3. Prepare the design of a web tool, including the functions provided for efficient work with video, review and editing of annotations.
4. Implement the designed web application using appropriate frameworks and libraries.
5. Evaluate the created application with the help of real usage scenarios, users and provided data.
6. Present the achieved results.

Literature:

- https://react.dev/

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Klíma Ondřej, Ing., Ph.D.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	9.5.2024
Approval date:	9.11.2023

Abstract

The goal of this thesis was to develop a web-based tool for object annotation in videos. The tool is designed to provide an interface for annotating objects in order to further use the annotation data in other applications. The tool utilises modern frameworks and libraries for creating user-friendly web applications. This thesis contains information about existing tools, libraries, and tools used in the development process, the user interface development process, and implementation details. The implemented tool includes features for video management, annotation editing, and data import or export. Thanks to the application, the user is able to annotate their videos for further usage in different areas.

Abstrakt

Cílem bakalářské práce bylo vyvinout webový nástroj pro anotaci objektů ve videích. Nástroj je navržen tak, aby poskytoval rozhraní pro anotaci objektů za účelem dalšího využití dat anotace v jiných aplikacích. Nástroj využívá moderní frameworky a knihovny pro vytváření uživatelsky přívětivých webových aplikací. Tato práce obsahuje informace o existujících nástrojích, knihovnách a nástrojích použitých v procesu vývoje, o procesu vývoje uživatelského rozhraní a podrobnosti o implementaci. Implementovaný nástroj obsahuje funkce pro správu videa, editaci anotací a import či export dat. Díky aplikaci má uživatel možnost anotovat svá videa pro další využití v různých oblastech.

Keywords

video annotation tool, web application, React, machine learning tool, data annotation tool, JavaScript, TypeScript, user interface, web design, NestJS, video annotations, JSON, Postgres

Klíčová slova

nástroj pro anotaci videa, webová aplikace, React, nástroj pro strojové učení, nástroj pro anotaci dat, JavaScript, TypeScript, uživatelské rozhraní, web design, NestJS, anotace videa, JSON, Postgres

Reference

MOISEEV, Nikita. *Online video annotation tool.* Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ondřej Klíma, Ph.D.

Online video annotation tool

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Klímy. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Nikita Moiseev
May 6, 2024

Contents

Chapter 1

Introduction

Video processing is one of the most well-known applications of machine learning, which has gained popularity as a field in computer science. This includes teaching computers to evaluate and extract information from videos, which frequently requires the difficult process of annotating each frame and describing what it contains. To get over this problem, companies have invested in developing specialist tools that make the annotation of videos easier and enable efficient frame-by-frame annotation with user-friendly interfaces.

Video annotation was not seen as a crucial activity in the past, and machine learning projects were not as common in the computer science community. The value of video annotation has increased dramatically, and for many engineers, it is now a crucial aspect of their machine learning workflow. As a result, a greater variety of video annotation tools have been created; many of them are widely used by companies that mostly specialise in video processing. Even though these tools have unique advantages, they are often commercial in nature and charge a price to use them. Chapter 2 presents an extensive review of the existing tools, including their advantages and disadvantages.

In order to create this open-source application, Chapter 3 dives into the specific tools and libraries chosen to aid the development process. This chapter will identify the most suitable technologies and provide an analysis of the chosen tools and libraries. It will explain how their functionalities contributed to the application's features, laying the groundwork for the in-depth implementation details covered in Chapter 5.

Chapter 4 dives into the design of the implemented tool. Here, we'll analyse the development process, focusing on the design choices made to create a user-friendly interface that facilitates efficient video annotation. This chapter will also present a comparison between the implemented tool and existing applications.

Essentially, the goal of this thesis is to present a thorough analysis of the video annotation tool, highlighting its advantages and contributions to the field of machine learning. Also, developers looking for an effective and open-source tool for their video processing projects will find it to be a useful resource.

Chapter 2

Existing tools

Before, even if machine learning algorithms could solve problems that previous algorithms could not solve or could only solve much more slowly, they were too heavy on resources to be used. A decrease in computing time are now possible thanks to modern hardware and software developments. Machine learning algorithms have quickly and widely been used as a result of this. They currently provide more effective solutions to a wide range of problems.

A significant field of use for these techniques is computer vision. They are employed in the classification, identification, or recognition of certain persons, things, or events in images or videos.

These algorithms are then used in a variety of fields, such as security, to evaluate photo data and identify individuals who may be a threat to those who are nearby [4].

The medical field is where computer vision is also often used. Computer vision algorithms are used for analysing the data of various patients in order to improve the quality of healthcare by making more accurate diagnoses [6].

The third use, computer vision for autonomous vehicles, will be covered in more detail in this thesis. The use of machine learning algorithms is made to enhance traffic safety and efficiency in many populated locations [3].

For all of these applications, accurate and truthful data are important for the machine learning algorithms to use, since this can decrease the mistake probability to almost zero.

Software programmers annotate their data using a variety of tools. The main focus of this thesis is on video annotation process research and new solution proposal.

Analysis of the current video annotation process tools is required in order to assess and evaluate the results that have been achieved.

2.1 Label Studio

Label Studio is an open source data labelling platform. It enables users to create labelled data for machine learning models. This platform allows the user to annotate various types of data, including images, text, and videos. To annotate the data, the user must first create a new project, import their data, and configure labelling [9].

Project setup

To set up a new project, it is enough to set its name. The two remaining steps can be completed later. The project description can also be specified, making it easier to identify projects from each other. The project setup interface is presented in the Figure 2.1.

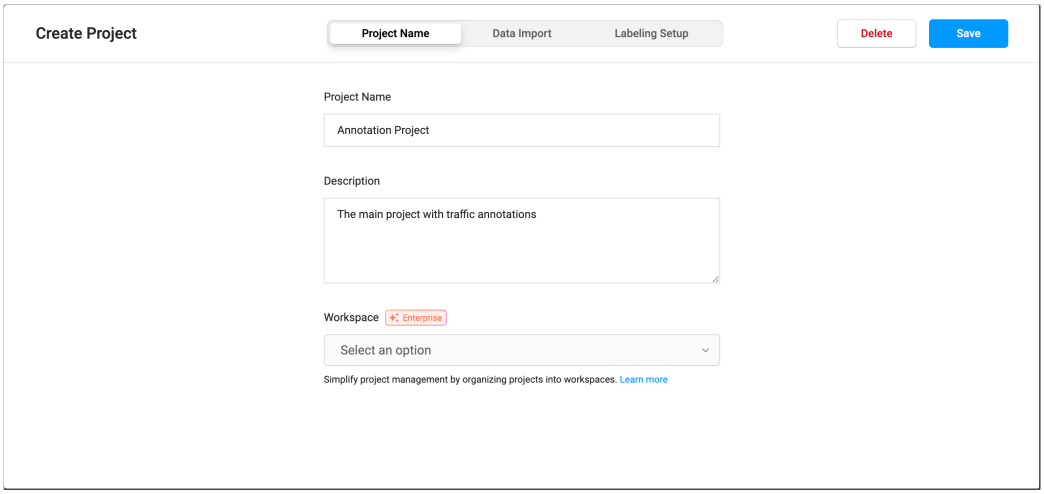

Figure 2.1: Basic project setup window.

It is also possible to select a workspace for the project at this step. Users may arrange projects with the use of workspaces. Additionally, they can be used to define a user's access to several projects. However, this functionality can only be accessed through the application's enterprise edition, which requires payment.

Data import

In order to save user-provided files, Label Studio requires setting up internal or cloud storage. The user may configure the following storage formats for the project:

- Amazon S3[1]
- Google Cloud Storage[2]
- Microsoft Azure Blob Storage[3]
- Local Disc Storage

To use cloud storage services, their providers demand payment. By default, while setting up the label studio platform on a server, the user needs to provide the local storage folder. If cloud storage is not configured by the user, local disc storage is used by default. If it is needed, the cloud storage configuration can be set up later. Existing projects can then be transferred to a newly configured type of storage.

The user can import their data after configuring the platform's file location. There are two available methods of importing data for the project: the first involves uploading each file manually, and the second involves providing the URL for the data file. The application supports various file formats, including video files. The upload interface is presented in the Figure 2.2.

[1] `https://aws.amazon.com/s3/`
[2] `https://cloud.google.com/storage`
[3] `https://azure.microsoft.com/en-us/products/storage/blobs`

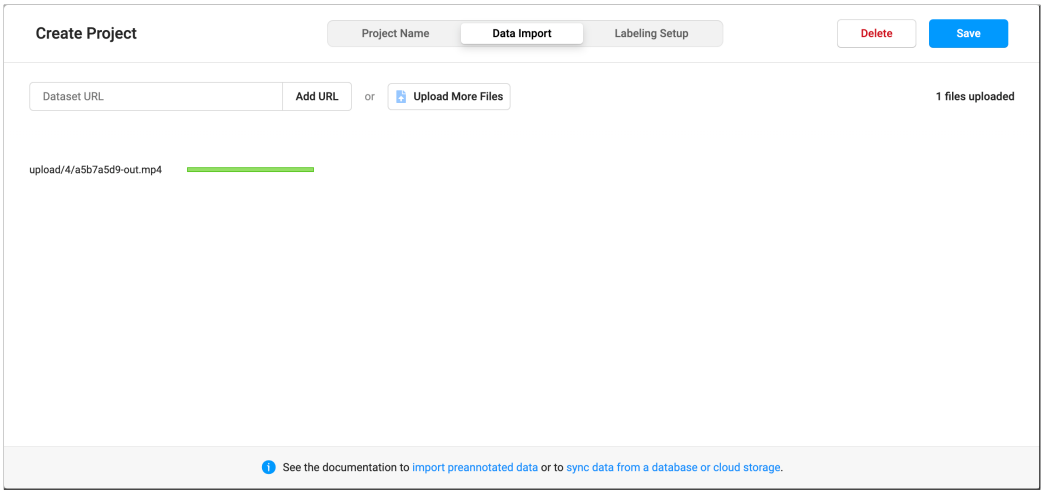

Figure 2.2: Data import during the project creation.

Annotations setup

The third step in creating a new project in the application is defining the annotation format. The application includes some predefined templates for these settings, which can be used to ease and speed up project creation. Additionally, there are two methods for manually defining the format. One approach is to use the XML language and components found in the Label Studio documentation[4] to create configuration. The user may efficiently customise the annotation formats in this way. The second method is to specify all the settings using a user interface. Though this method is not as flexible as the previous one, it is still useful in situations where specific customisations are not needed for the project. The two configuration interfaces are presenter in the Figure 2.3.

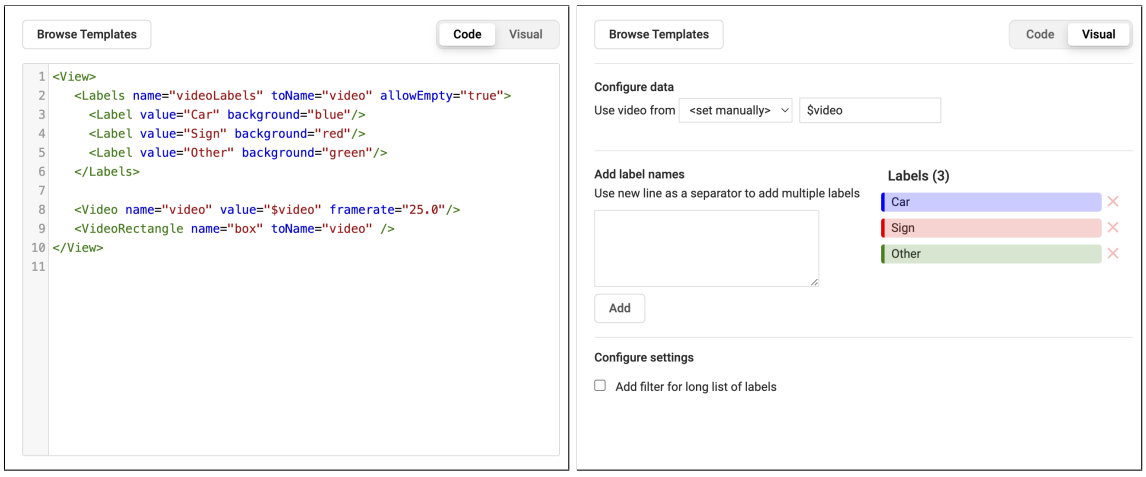

Figure 2.3: Code editor for annotation configuration on the left. User interface for the configuration on the right.

A prepared template for video annotation tasks called “Video Object Tracking” is already included in the application. This template sets up the video source and several

[4] `https://labelstud.io/tags/`

predefined annotation types. The predefined annotation types should be adjusted according to the requirements of the annotation task because the ones provided by the template may not meet the task requirements. The path of the video file imported during the second step should also be updated as the source of the video parameter.

The user can start working on adding annotations to the imported video after finishing the three steps above.

Annotation creation and export

The user must draw a rectangle above the necessary region in order to create an annotation on the video frame. The created annotation shows up in the list of all annotations and immediately receives a name. The type of annotation may be adjusted by the user. The newly created annotation gets interpolated to all frames that come after the frame it was created on. This interpolation can be disabled for each annotation individually. The only shapes the user may create to annotate the video are rectangles. The application does not support more complex polygons for video annotation. The application interface is presented in Figure 2.4.

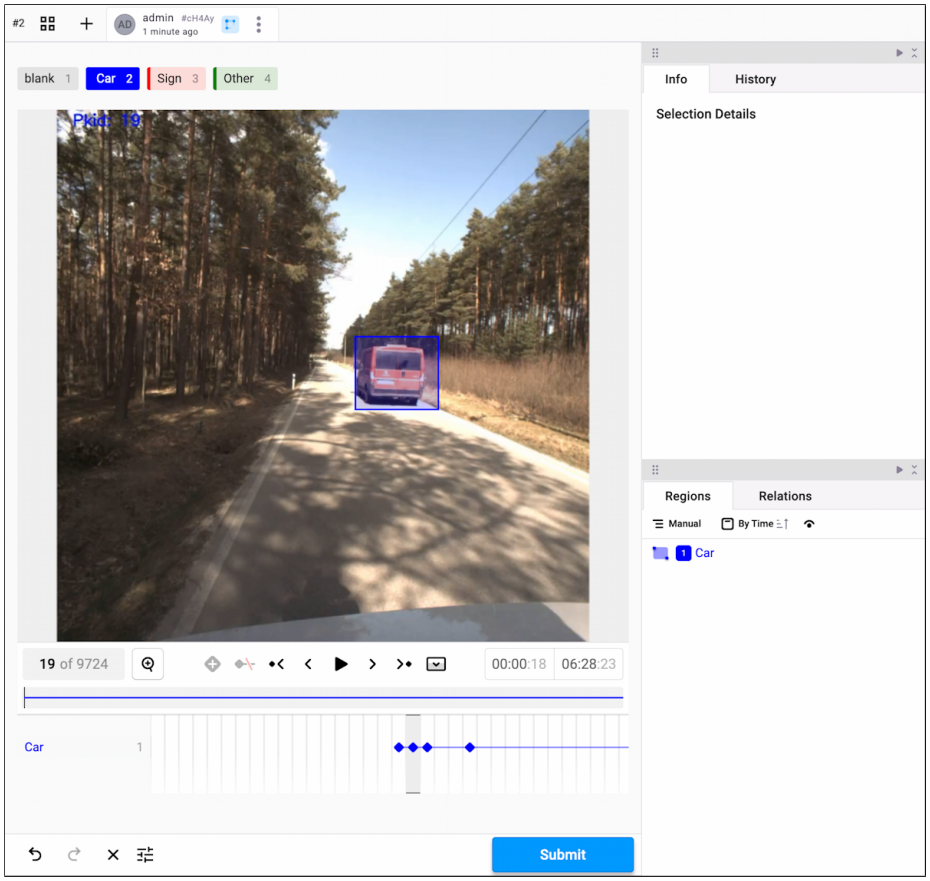

Figure 2.4: Annotation interface.

After creating all of the annotations, the user can export them in several file formats. The exported file contains an array of annotations for the project's video. Every annotation contains an array of objects arranged in the frame order, each of which specifies a rectangle in the frame. This file can then be used for machine learning purposes.

2.2 V7 video annotation platform

The V7 video annotation platform allows users to create pixel-perfect video annotations. This platform supports various types of video files and multiple export formats for the annotations created. To start the annotating process, the user needs to create a dataset, import their files, and set up the classes of the annotations [10].

To begin the annotating process, the user needs to create the dataset and configure it.

Dataset configuration

Importing files to the dataset is the first step after creating a dataset in the application. There are various options to set the location of uploaded files. The application supports integration with the following cloud storage platforms: Amazon S3, Google Cloud Storage, Microsoft Azure Blob Storage, and MinIO[5]. The first three cloud storage platforms were already mentioned in the Label Studio section (reference here). Those platforms require payments to access their storage services. The MinIO platform is an open source platform, which means that it can be deployed to the user's server. The V7 application can be configured to use this self-hosted solution, which ensures data privacy.

The user can also upload their files without configuring integration with cloud storage services. In this case, the file storage process is managed by the V7 platform. This platform is not open source, which means there is no way to know how the user files are being stored. To upload files to the dataset, the user needs to manually upload them to the application. The import modal interface is presented in Figure 2.5.

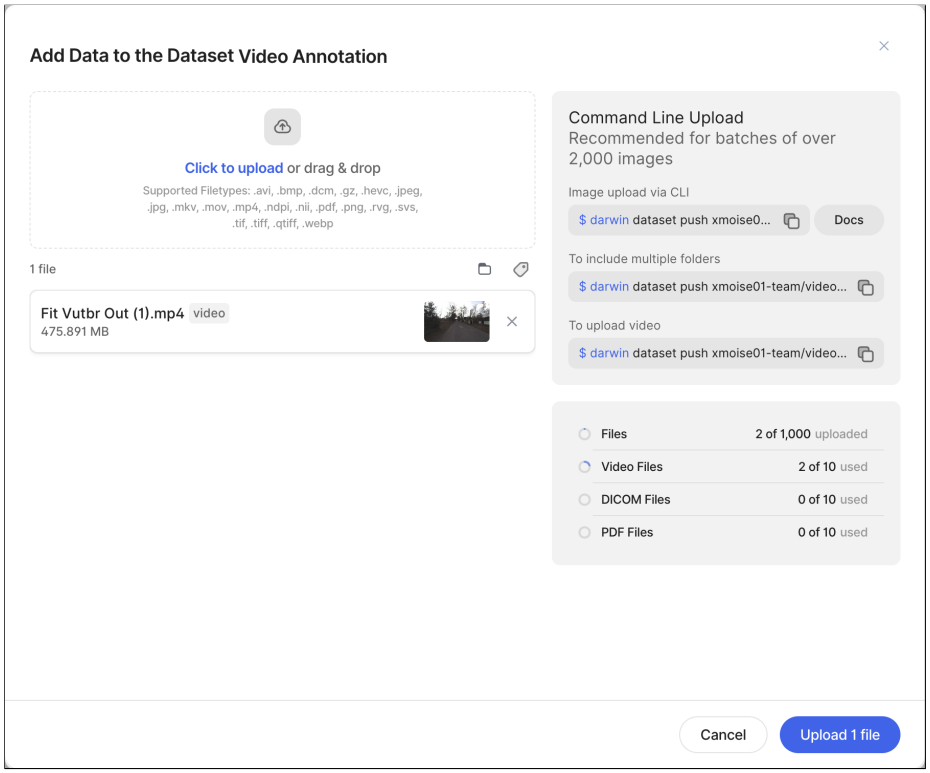

Figure 2.5: Files import modal window.

[5] https://min.io/

The next step after uploading files is to configure annotation classes, which will be used in the annotating process. The V7 platform supports multiple types of classes. The user can select one of the following class types: bounding box, ellipse, keypoint, line, mask, polygon, skeleton, or tag. Each of those annotation classes may be suitable for some specific annotating process. The user can also specify such information about the class as name, colour, and instructions for other users on what the class should be used for. The classes configuration interface is presented in Figure 2.6.

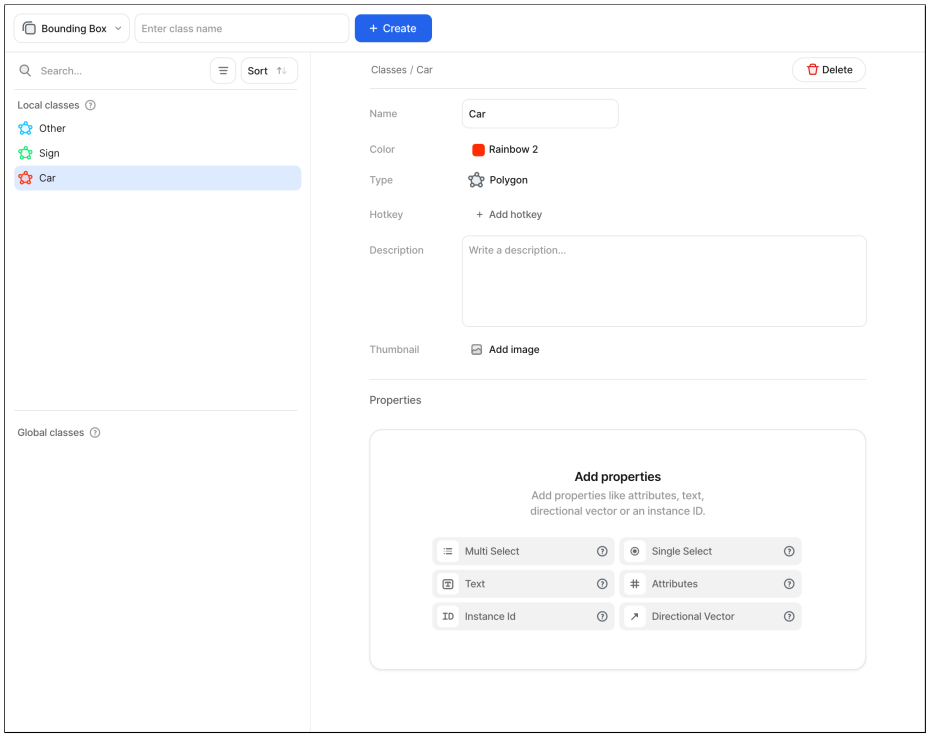

Figure 2.6: Annotations classes configuration interface.

After uploading video files and configuring annotation classes, the user can start creating annotations.

Annotation creation and export

The user is expected to use the application interface to start annotating the imported video file. Figure 2.7 presents the interface, which consists of the following components:

- The video player component displays the frame that is currently selected and provides a visual representation of the annotations on the frame.
- The timeline component is used to display the created annotations along with their class names. The component also includes a representation of the frames. This component also enables a smooth and simple way for the user to navigate around the video.
- The annotations list component enables extra configuration settings and annotation interactions in addition to displaying all of the annotations in the current frame.

- The tools panel, which is expected to be utilised by users in order to select a tool to create an annotation.

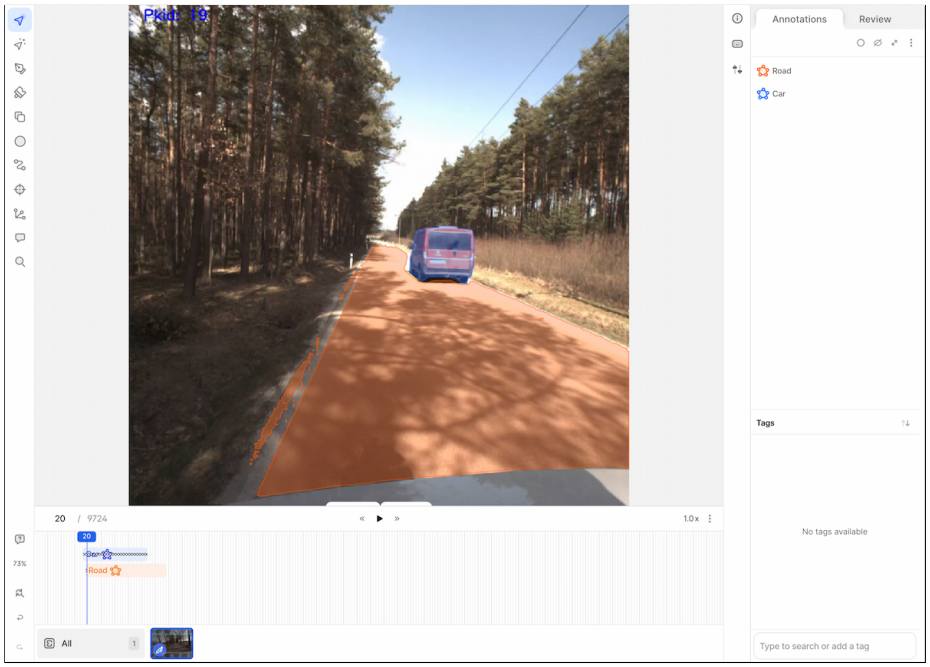

Figure 2.7: Main annotating interface.

The user has to select the tool they want to use to annotate the object in the current frame before they can create an annotation. Inside the application, the tools panel provides a variety of annotation tools. Among the tools offered are an auto annotate tool and a polygon tool. It is possible to annotate classes with polygon types using these tools. All the tools in the panel are expected to work with the corresponding annotation classes. The auto annotate tool relies on computer vision algorithms and enables highly precise annotation of complex shapes. These algorithms are able to identify the borders and shapes of objects in the video. This provides smooth and accurate polygons, which might be quite beneficial for certain annotation tasks. With the polygon tool, the user may manually annotate a video object by creating a polygon above it. This tool can be used for annotation tasks that do not require accurate annotation shapes and produce annotations that are not as precise as the auto annotate tool's.

After annotating the necessary frames and objects on the video, the user can export their annotation data. The V7 application can export the annotation dataset in a variety of formats that are compatible with other annotation tools. These formats are presented in Figure 2.8. A badge with information about the format itself is present on almost all annotation formats. The "Most Versatile" badge, for instance, means the format that holds it can be used in more tasks than others. After exporting, the file may be used for machine learning training tasks or imported to other platforms providing video annotation functionality.

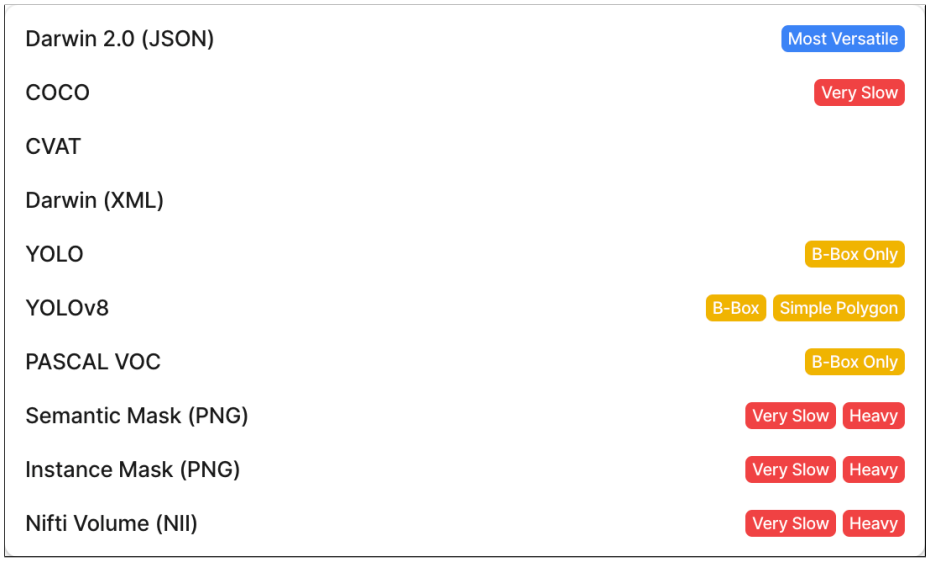

Figure 2.8: Annotation export formats.

2.3 CVAT

CVAT is a free, online, interactive video and image annotation tool for computer vision. It is being developed and used by CVAT.ai to annotate millions of objects with different properties [2]. The application enables broad customisation of the annotation and project settings, as well as a flexible annotating approach. Since the application is open-source, the entire source code may be accessed on the company's Github repository[6]. Additionally, this enables developers to modify the application according to their requirements.

The user must set up the project, add annotation tasks, and upload videos to be annotated before they can start the annotation process.

Project configuration

The user must provide the project's name in order to set it up. Additional configurations might be finished afterwards. The user can additionally define labels using the create project interface; these labels will be utilised in the annotation process. There are two methods to do this: either by utilising a user interface or by defining the annotation labels using JSON code according to the schema defined by the application developers. Each of these methods offers the same functionality. However, the user may prefer the JSON configuration method in order to share or store the configuration code if necessary. The project creation interface is presented in Figure 2.9.

The label configuration includes specifying the label name, colour, type, and attributes. The annotation may be identified by its name and colour attributes in a collection of other annotations. The annotation's shape is defined by the type attribute. This type can be any two-dimensional shape, such as an ellipse, a polygon, or any other shape. It may also be a straight line or a polyline. The application also supports three dimensional shapes for annotation label types. The label's attributes allow the categorisation of annotations of the same type without the need to create several annotation labels. This allows the user to define more detailed properties for the label.

[6] `https://github.com/cvat-ai/cvat`

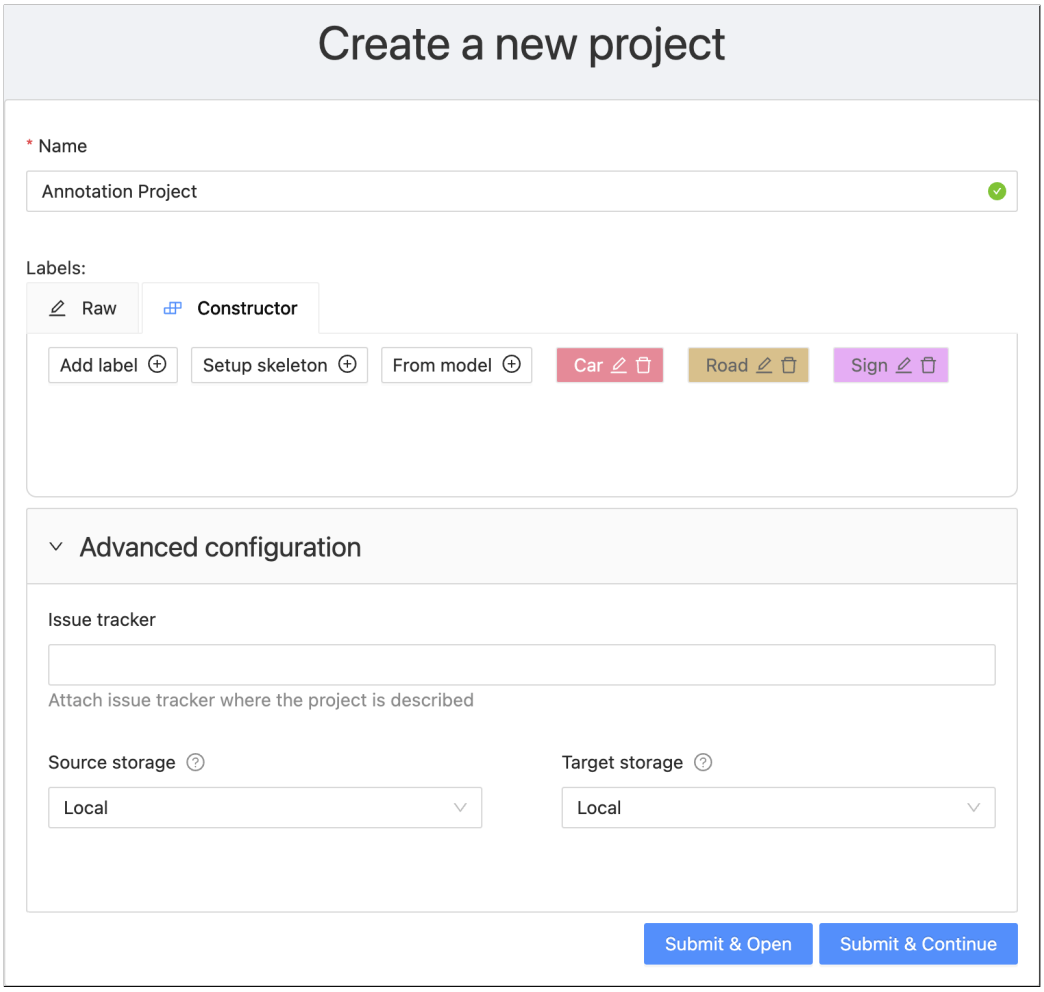

Figure 2.9: Project configuration interface in CVAT application.

The user is also able to specify the location of the output files, which include annotation datasets, exported frames with annotations, and any other files that may be exported depending on the export format, as well as the location of the video files used as the input for the annotation process during the project creation process. For this, the user may utilise local storage, which includes downloading output files to the client side and importing video files to the application from the client side. The user also has the option of using cloud storage services like Microsoft Azure Blob Storage, Google Cloud Storage, and Amazon S3. Using cloud provider services means exporting the application's output files to the cloud provider's servers and uploading files from the cloud provider's servers to the application.

To start annotating videos, the user has to create an annotating task after completing the project creation process. This process involves specifying the task name, project, and files. The video files that have been uploaded may be coming from the cloud or local storage. The user does not need to specify the labels for the task again because they are already specified for the project and will be used for this task. The task creation interface is presented in Figure 2.10.

Once the task has been created, the user can start annotating the video that was uploaded for the task.

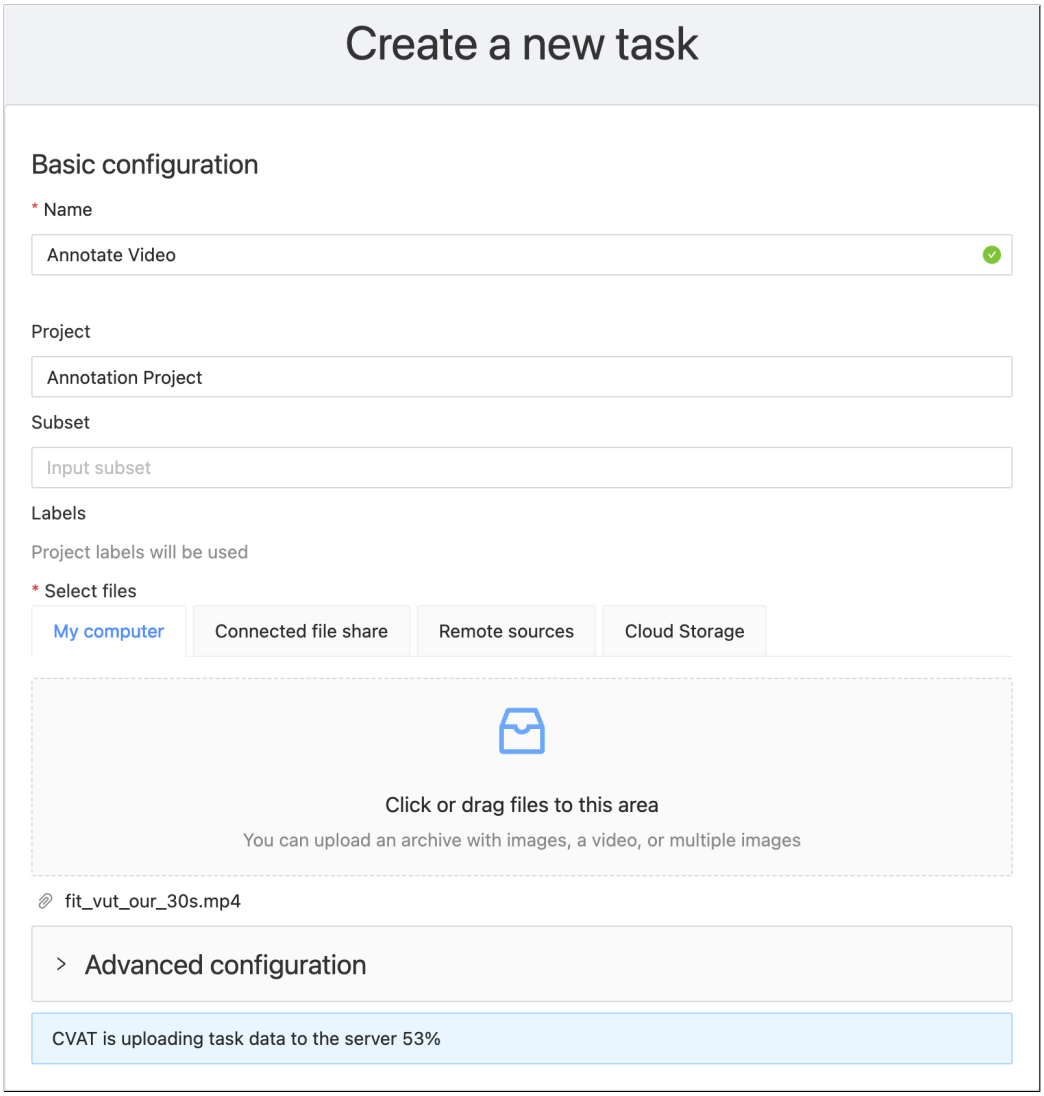

Figure 2.10: Task creation interface in CVAT application.

Annotation process

The annotating interface of the application consists of the video player, the tools panel, and the annotations list in the right sidebar. The complete annotating interface is presented in Figure 2.11.

The video player component displays the frame that the user selected to work on. The created annotations are also displayed, along with their labels and colours. To make it easier for the user to annotate the frame's small objects, the frame can be moved and scaled as needed.

In order to create annotations, the user has to select a tool from the tools panel. The panel includes various tools that allow the user to create different types of annotations or to manipulate the video frame itself. The annotation can be created using the tools that correspond to each label type that was set during the project creation process. To create an annotation with the selected tool, the user must draw a shape above the desired object in the chosen frame.

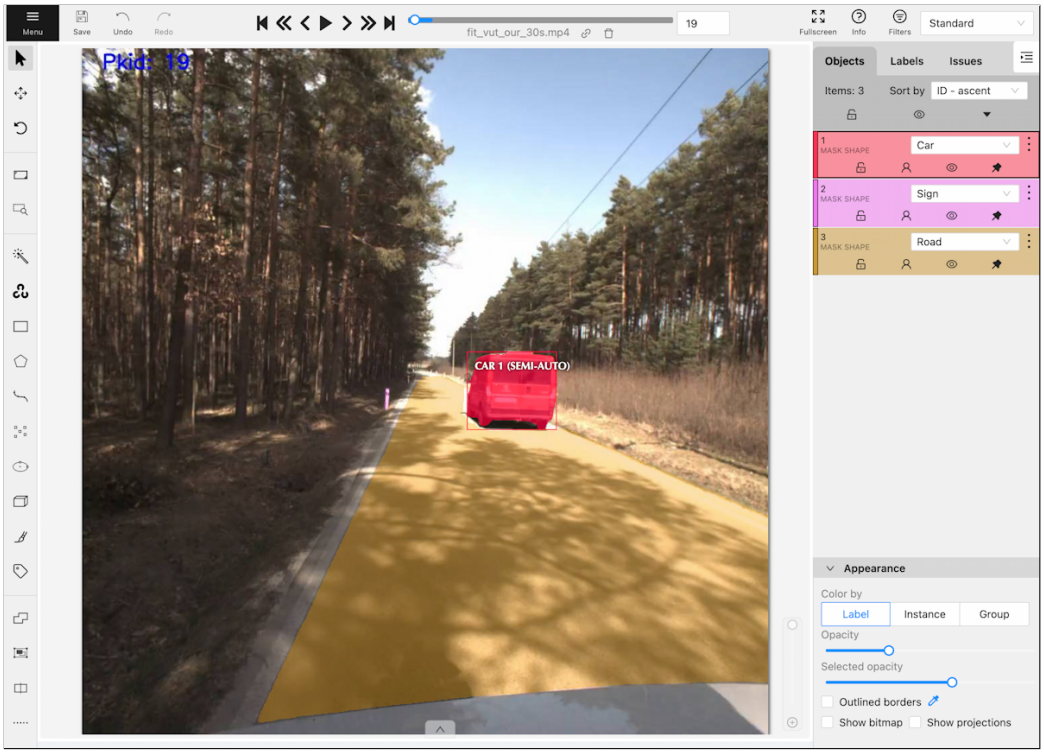

Figure 2.11: Main annotation interface in CVAT application.

After the creation of the polygon, the annotation is displayed in both the currently selected frame and the sidebar component, which shows all of the annotations in that frame. Every created annotation can be modified at any time in case the annotation's shape does not meet the requirements of the annotating task.

The additional interface components enable the user to work with the annotation data or the video. This includes filtering annotations, navigating across time inside the video, and specifying annotation display settings in the video frame.

Annotation dataset export

The user is expected to export the annotated data after finishing the annotation process so that it may be used in the training process of the machine learning model. The application allows the user to export their annotation dataset in multiple formats compatible with other programming libraries or annotating platforms. The export interface is presented in Figure 2.12.

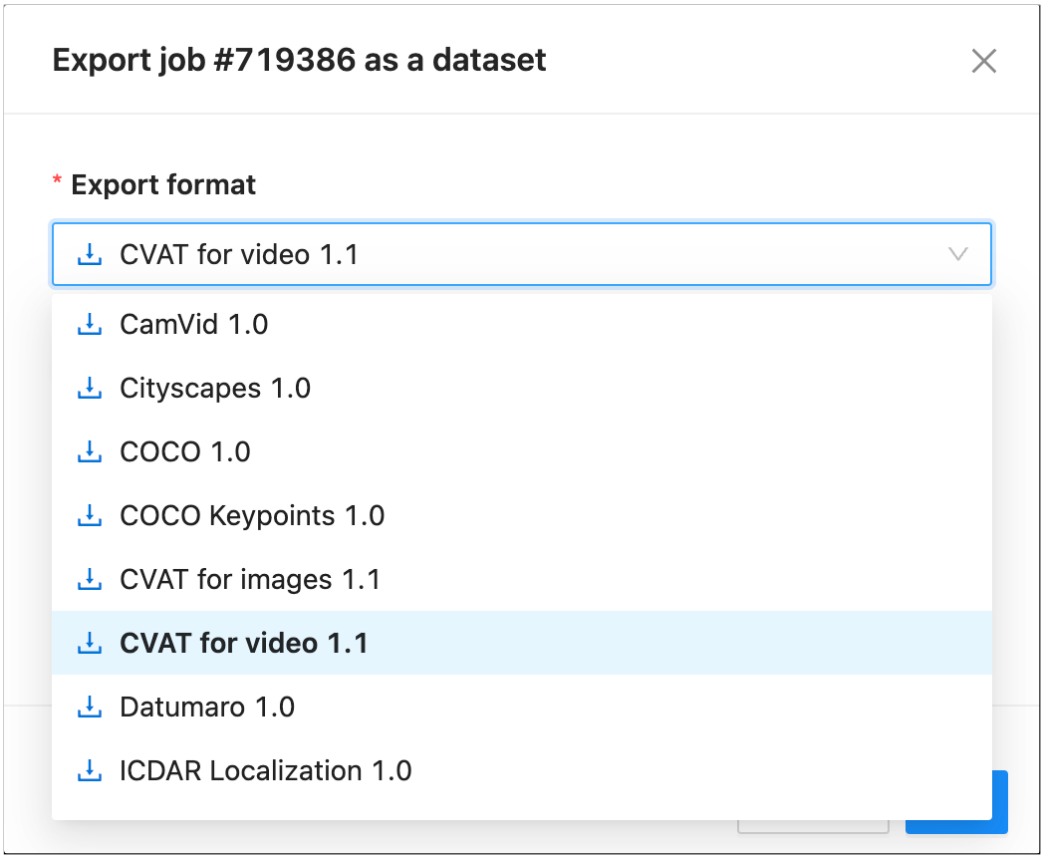

Figure 2.12: Annotation dataset export interface in CVAT application.

Chapter 3

Used languages, tools, and libraries

There are various tools and libraries that were used in the development process of the video annotation tool. The entire project is planned to work as a web application accessible to the public with an open-source code base. It was important to choose tools and libraries that are well supported in order to make the implemented tool more reliable.

3.1 Languages

TypeScript

TypeScript[1] is a strongly typed programming language that builds on JavaScript. I considered using this language because of the implemented tool's extensive functionality, and the implementation plan included handling a variety of data types. Developers can have better control over project types by using the TypeScript programming language. This is done for the purpose of making sure that the data is managed and stored as planned. Another reason for using this tool is to ensure that code base modifications will not cause unexpected errors, which might be overlooked when using non-typed programming languages like JavaScript.

JSX

JSX[2] is a syntax extension for JavaScript that enables writing HTML code in JavaScript code. This extension is used as the default markup language in React applications. Also, this extension lets developers create code blocks inside tags and pass variables as attributes for components. With the help of this JSX feature, developers can write conditional expressions or even render arrays of data in components.

3.2 Libraries

React

React[3] is an open-source library that is used to create complex user interfaces using the component principle. This is because React makes it possible for developers to divide their

[1] `https://www.typescriptlang.org/`
[2] `https://react.dev/learn/writing-markup-with-jsx`
[3] `https://react.dev/`

interface into reusable components, which simplifies organising code and maintenance. The implemented tool was made using this principle to prevent the code from becoming too complex and difficult to maintain in the future. Based on the functionality and design of the already existing tools, I decided that using components would be an effective way of implementing the tool's functionality. Every component of the implemented tool is responsible for its own functionality.

Material UI

Material UI[4] is an open-source library that offers a variety of React components designed according to Google Material Design guidelines[5]. The goal of the library for the tool implementation was to ease the interface development process and give the user, as a result, a modern and visually appealing user interface. Many web applications are designed according to the Google Material Design guidelines, resulting in a universal and user-friendly experience for users across different platforms. Using components that offer a comprehensive and well-balanced interface without distracting from the user's main goal when using the application was another reason to use this library.

Redux

Redux[6] is a JavaScript library for managing application state. The implemented tool was designed to consist of multiple independent components. Most of the components are meant to work with the same set of data. This led to a need to send the same data sets to the tool components, which might not even be related to one another in the code base. The Redux library is an appropriate choice for this purpose because it works as a single source of truth[7]. This principle guarantees that every component in the tool is operating with the same data and that changes to the data affect every component simultaneously.

KonvaJS

KonvaJS[8] is a JavaScript library that is used to draw graphics with the help of the Canvas API[9]. I considered including this library during the video annotation tool development because it contains built-in components for React applications that are highly configurable from the code and can be used directly in JSX markup during component rendering. This was one of the key points that had an influence on the decision making process of choosing the library for annotation rendering. The project uses this library to render each annotation that the user adds.

vis-timeline

vis-timeline[10] is a JavaScript library that offers the ability to create interactive components for data visualisation across time. I consider this library to be one of the most important libraries in the project. The library is used to give the user visible feedback about their

[4] https://mui.com/material-ui/getting-started/
[5] https://m2.material.io/design/guidelines-overview
[6] https://redux.js.org/introduction/getting-started
[7] https://redux.js.org/understanding/thinking-in-redux/three-principles
[8] https://konvajs.org/docs/overview.html
[9] https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
[10] https://visjs.github.io/vis-timeline/docs/timeline/

annotations in the video as well as annotation metadata, such as kind, name, and position in time. All of the tools that were covered in Chapter 2 of this thesis have timelines as a common component. The fact that this library provides a more familiar and intuitive interface for the video annotation process is another reason for using it.

Video.js

Video.js[11] is a JavaScript library that is used to show and configure a video player. This library extends the functionality of the HTML5 video element[12]. The project utilised this library to show and set up video visualisations for the user. It has several features, including compatibility for many video formats and customisable controls. These features enabled the application to include custom video controls, such as navigating between annotations and playing video frame by frame.

NestJS

NestJS[13] is a framework for building efficient, scalable NodeJS server-side applications. It uses progressive JavaScript, is built with and fully supports TypeScript, and combines elements of object-oriented programming, functional programming, and functional reactive programming. This framework is implemented in the application in order to create a server-side tier of the application. This server-side tier is described in the Section 5.4.

3.3 Tools

Docker

Docker[14] is an open source platform that makes it possible for developers to work with containerised applications, which integrate application source code with the operating system libraries and dependencies needed to run that code in any environment. The platform is used in the project for both local development and deployment processes. Each containerised application in Docker uses a template that contains instructions for creating a container, which is called an image.

During the local development process, developers can put the whole application with its data and dependencies inside a containerised version of the application. This guarantees that the development environment will be the same on every developer's machine, regardless of differences in operating systems.

To build a Docker image, which is required for the deployment process, the application contains a set of instructions. The image is stored in the Docker registry of images available for public access. The process of building the image is automated via the Github Actions platform. The build process is covered in the Section 5.5

Git, Github and Github Actions

Git[15] is an open-source version control system. This software is widely used in the developer community to monitor changes in source code and collaborate on projects with

[11] `https://videojs.com/`
[12] `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video`
[13] `https://nestjs.com/`
[14] `https://docs.docker.com/get-started/overview/`
[15] `https://git-scm.com/`

other developers. The reason to use Git in the project was to track feature development progress over time and across plans. Another reason was to make the project accessible to the public so that anyone might use it and modify existing features according to their own needs. The application source code is available at `https://github.com/NickSettler/video-annotation-tool` and at `https://github.com/NickSettler/video-annotation-backend`.

Github[16] is a cloud-based platform that stores developer projects in repositories using Git functionality. The platform web interface improves basic Git functionality by adding several features like pull requests and issues, which makes the contribution process more seamless. One of the reasons to use Github was its worldwide use in the developer community.

Github Actions[17] is a continuous integration and continuous delivery platform that is maintained by Github. The platform is available to use by developers for free in their repositories. This feature was the second reason to consider utilising Github during the project's development. The entire project is utilising Docker functionality for both local development and deployment processes. The Github Actions platform automates the process of building the project and uploads the output image to the Docker images registry.

Postgres

PostgreSQL[18] is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. This database is utilised in the application in order to store users' data, such as projects they are working on and videos they upload. The database model is described in the Section 5.4.

[16] `https://docs.github.com/en/get-started/start-your-journey/about-github-and-git`
[17] `https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions`
[18] `https://www.postgresql.org/`

Chapter 4

User interface

It was important to determine user use cases before beginning the development of the user interface that would satisfy all project criteria. Because the implemented tool was made in cooperation with the company, the use cases were defined in advance and were sufficient to begin the user interface development process for the project The required use cases are presented in the Figure 4.1.

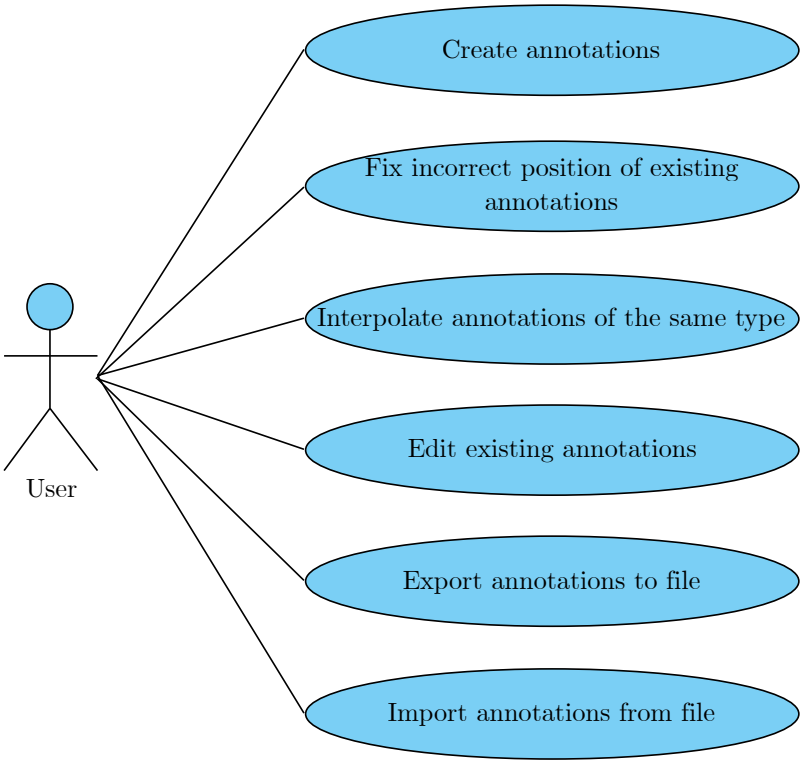

Figure 4.1: Video annotation tool user use cases.

To ensure that the implemented tool would meet the requirements, it was expected that the output of the user interface development process would fulfil each of the use cases that were provided.

4.1 Interface development process

The entire application was designed to have the same component composition as the existing tools analysed in Chapter 2 for a better user experience. The initial user interface design was created by hand as a low-fidelity prototype [11]. Before using digital design tools, this was designed to have a better understanding of the complete structure. The prototype is presented in the Figure 4.2

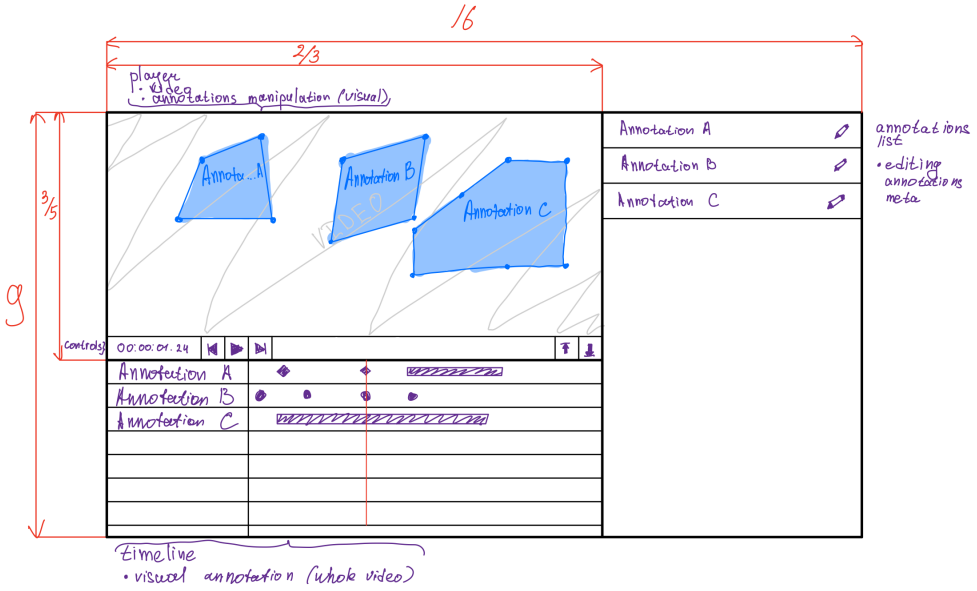

Figure 4.2: Low-fidelity prototype of the tool drawn manually.

The next phase in the interface development process was to use the Figma instrument to create a high-fidelity prototype. The prototype is presented in the Figure 4.3.

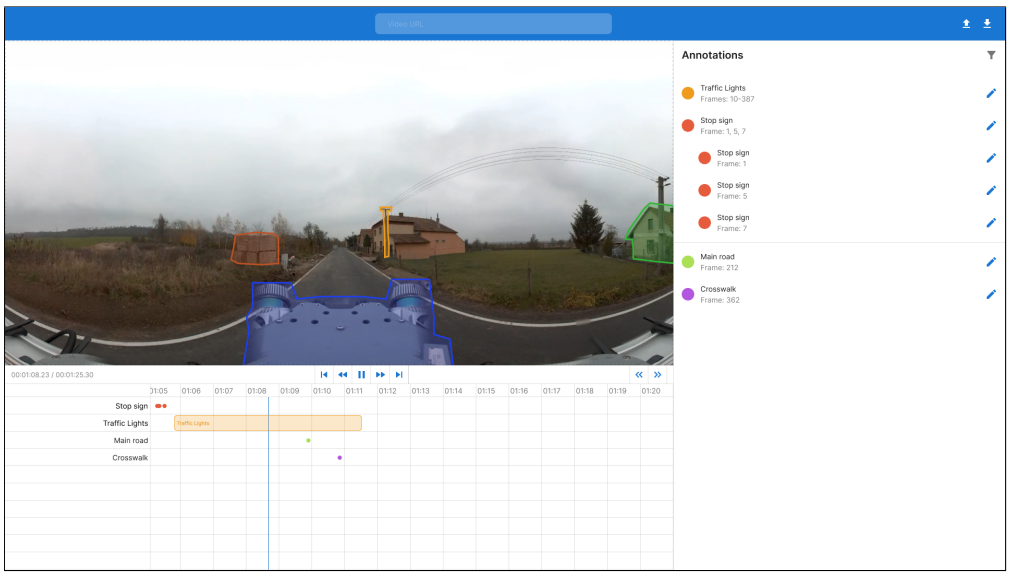

Figure 4.3: High-fidelity prototype of the tool drawn using the Figma tool.

During this iteration, some problems with the initial paper-based approach were discovered:

- Users cannot find out how they should begin working with their video. The user cannot upload their video or at least input the video URL to download it from the internet using any fields or dialog windows. It was decided to add a toolbar at the top of the page with a field for the user to enter the video URL they would like to work with to resolve this issue. The import and export buttons were also moved from the controls toolbar to the page's top navigation bar.

- The user does not receive a comprehensive overview of all the annotations in the sidebar annotation list. If a user works with annotations of the same name but of different types, they may become confused because the list does not visually separate them. The solution to this issue was to give each annotation in the list a frame number caption and colours for annotations of the same kind.

- Another issue with the annotation list was that there was no difference between multiple annotations of the same type spread over multiple frames and a single annotation in a single frame. When looking for the required annotation in the list of annotations with the same name and type, this problem may cause misunderstandings or confusion. This issue was resolved by dividing an annotation list into two separate sections. The first section is intended to display several annotations of the same type in a grouped and collapsible form. The second section focuses on providing detailed information about each annotation that appears only once in the video.

- The next problem was the absence of controls for some necessary user interactions that the user could be familiar with. These interactions were direct jumps between annotations and frame-by-frame moves through the video. It was decided to make a few blocks of logically related controls nearby, and justifying the blocks themselves across the control panel was another way to improve the control panel. Following the Gestalt concept of proximity [7], this change aims to improve users' ability to easily locate and access the controls they need.

The tool's high-fidelity design helped in discovering issues with the initial design. The application was able to be implemented with this version of the design, which covered all the stated use cases. To enhance the user experience, a few more minor design adjustments were also performed during the implementation process.

4.2 Composition components

After defining application use cases and analysing existing tools' interfaces, it was decided to use a video player, a timeline, and a sidebar as the three main application components. Those components are used in most of the existing tools designed to fulfil the same purpose.

Video player

The video player component must allow the user to work with the video content to annotate entities in the video. The whole component should be made of two blocks that overlay each other: the video element, which will display the video content, and the canvas element,

which will let the user create and interact with the annotations. The video player component is presented in the Figure 4.4

The process of adding a new annotation should include locating the feature in the video and creating a corresponding polygon using the mouse. For editing existing annotations, the flow is to identify a mistaken annotation and make changes to the polygon.

The next important feature of the video player is zooming and moving functionality. It should be easy for the user to drag the video around in the player window and zoom in and out of it. In order to help the user focus on specific parts or areas of the video, this functionality is important. This feature could be very useful when the video resolution is quite high and the important details are small or hard to notice.

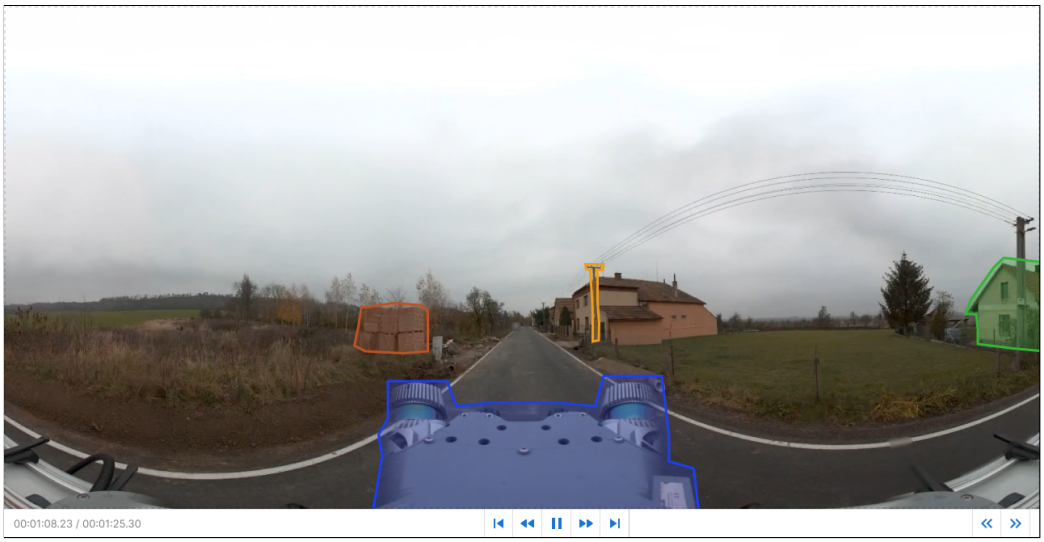

Figure 4.4: Video player component design.

Timeline

The timeline component is meant to provide the user with a visual and clean overview of all the annotations in the video. For each annotation type, it contains a row with two annotation display possibilities. If there are several annotations of the same type on adjacent frames, the user will see a rectangle spanning across each frame where the annotation is present. For single annotations that do not have the same type annotation on the adjacent frames, the user will see only a point indicating the position in time of the annotation. The timeline component is presented in the Figure 4.5

The timeline component is not supposed to satisfy any of the use cases defined earlier. Nevertheless, this component improves the user experience by providing a visual representation of the annotation data.

Sidebar

The sidebar component is designed to list all the annotations that are presented in the video. Every annotation in the video is shown in the list, with some basic details about it, such as its name, colour, frame, or a range of frames where it is located. Additionally, the editing annotation functionality is designed in this component, which enables users to modify annotation data. The sidebar component is presented in the Figure 4.6

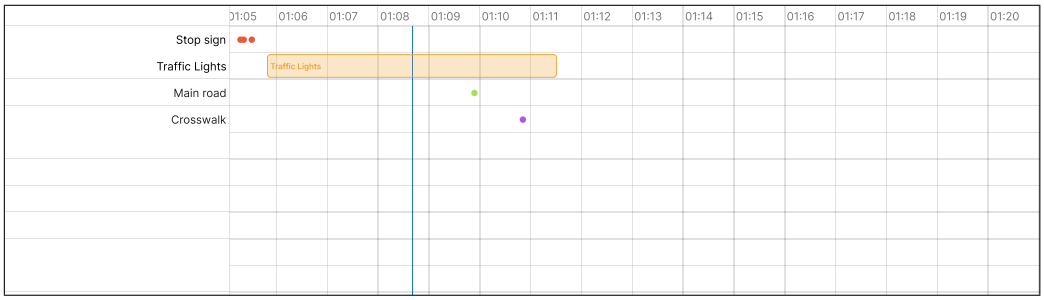

Figure 4.5: Timeline component design.

The list of annotations in the sidebar is separated into two sections: a section with annotations of the same type located in multiple frames and a section with annotations that appear only once in the video. The sidebar also allows users to easily navigate between annotations and view details about each one.

The sidebar component also allows the user to filter video annotations based on their location over time and type. This functionality simplifies working with massive sets of annotations.

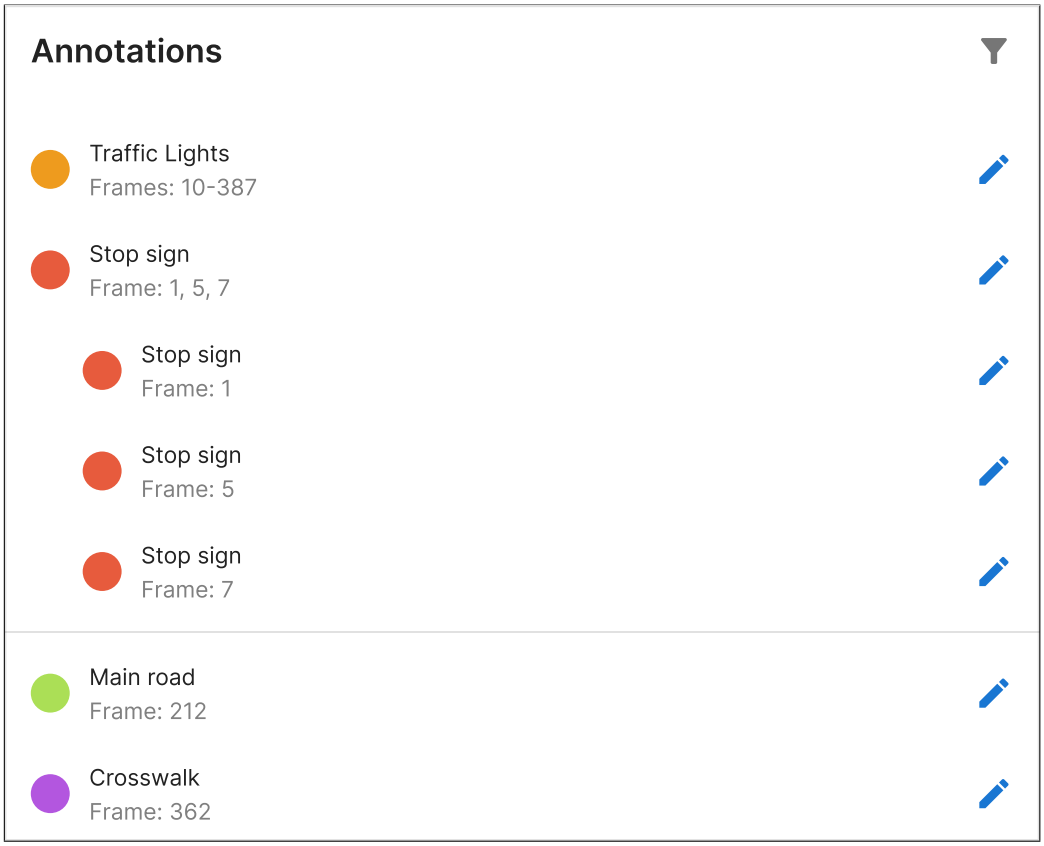

Figure 4.6: Sidebar component design.

4.3 Tools compare

There are some common criteria that may be used to compare the implemented tool and the existing tools, which are described and analysed in Chapter 2. These criteria can be categorised into four subgroups: components, annotations, deploy, and import and export. The comparison of these criteria is shown in Table 4.7.

	Label Studio	V7 Annotation Tool	CVAT	Implemented tool
Timeline	☑	☑	☑	☑
Tools panel	☐	☑	☑	☐
Annotations list	☑	☑	☐	☑
Creating polygons	☐	☑	☑	☑
Annotation interpolation	☑	☑	☐	☑
File storage options	Self-hosted / Cloud	Remote / Cloud	Self-hosted / Remote / Cloud	Self-hosted / Remote
Deploy options	Self-hosted	Remote	Self-hosted / Remote	Self-hosted / Remote
Import various formats	☑	☑	☑	☑
Export various formats	☑	☑	☑	☐

Figure 4.7: Existing tools and implemented application criteria comparison.

There are several differences between the implemented application and the existing tools. While some of them have advantages, others have disadvantages that might serve as motivation for future application improvements.

Chapter 5

Implementation

5.1 Import and export

One of the application's stated use cases was to allow users to edit a set of annotations. The implemented application aims to shorten the time required to prepare data for machine learning models by artificial intelligence developers. Therefore, the ability to import annotation data or export created annotations for training models was one of the primary features of the application.

Import

During the implementation phase, the data set of annotations generated by the external machine learning model was already provided for testing purposes. Its format had to be changed to the internal application annotation format during the import process. The initial concept of the import process was simple and easy to implement. Nevertheless, the final decision about the implementation of the import process differed from the initial concept.

One of the main concepts and features of the implemented application is that it is open-source. This concept allowed for customisation among users. It also led to a number of issues during the implementation process. One of them is the difference in data sets, which the user might wish to import during their work. The company that was collaborating on the tool's implementation provided data collection that was large and organised to be converted to the internal application format. However, there is also the possibility that the data used by the other developers might not be so extensive and might not be used in the application. Additionally, this problem also causes the issue that the data conversion process cannot be defined in the application code base and requires code changes to work with different data sets and formats.

The stated problems were the primary reasons for developing a universal data conversion tool, which would enable users to specify conversion rules for data sets in various formats in the application without involving manual code changes. As an outcome of implementing this data conversion tool, users will be able to convert data in multiple formats with ease instead of changing the application code for each conversion.

Using a dialog window, the import tool guides the user through three sequential steps of the import flow: uploading a file with annotations, choosing the data set format, and mapping[1] the uploaded file format to the internal application format.

[1]Data mapping is the process of matching data fields from one source to data fields in another source.

Step 1: File upload

The first phase of the annotation import flow is data uploading. The import tool is intended to work with the JSON file format, which is frequently used to store structured data in files. The user can use a designed drag-and-drop area to upload their file, or they can pick the file using an operating system dialog window. The first step modal window is presented in the Figure 5.1. After choosing the file, the user can move on to the second step.

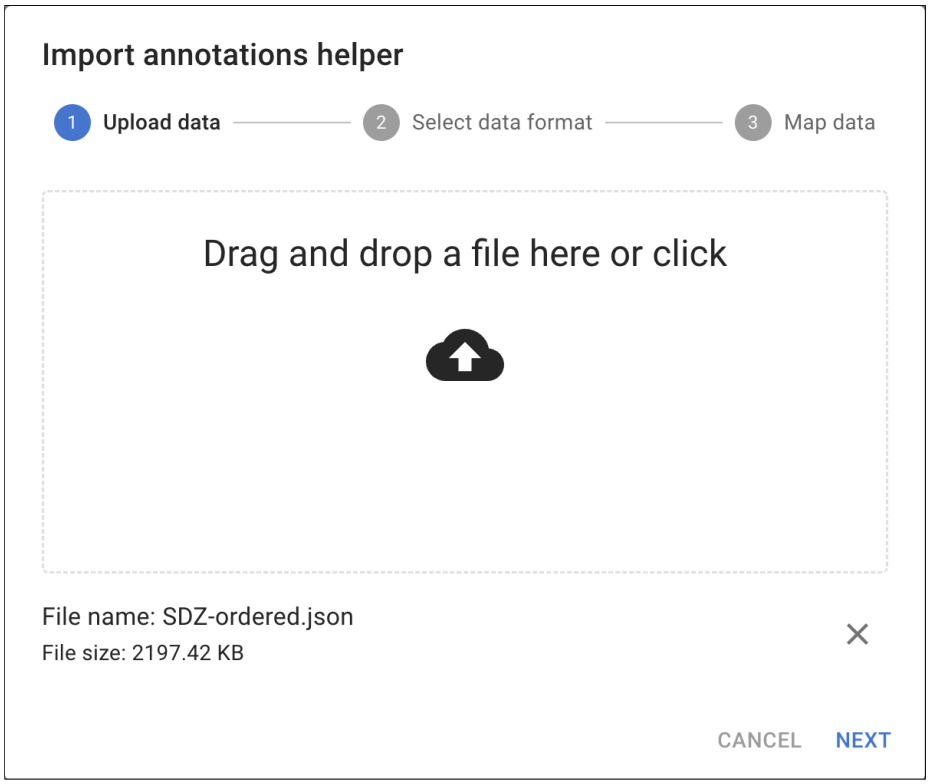

Figure 5.1: The first step of the import flow: selecting annotations file.

Step 2: Data set format selection

The file format selection is an important aspect of the import flow. The second step interface is presented in the Figure 5.2. During the tool development process, it was decided to provide the ability to import three various data formats:

- Application format: this format needs to be the same as the internal application data format. If the user wishes to make changes to the data set that was exported from the tool, they can simply import it using this format. Since this format is already in internal application format and may be imported just as is, it does not require additional conversion in the third step of the import flow.
- Array of frames: this format is meant to be used when a two-dimensional array is uploaded. The first dimension of the array represents a sequentially ordered frame. The second dimension of the frame array is the array of annotations in the frame.

- Array of polygons: this format is intended to be used with a one-dimensional array where each element of the array is a polygon. The array's annotations may be in any order.

These data formats might be suitable for most data annotation tasks. To continue with the import flow, any further formats that are incompatible with the given formats should be manually transformed to one of these three formats beforehand. To successfully finish the import process in the third step, the annotation array's elements must be homogeneous.

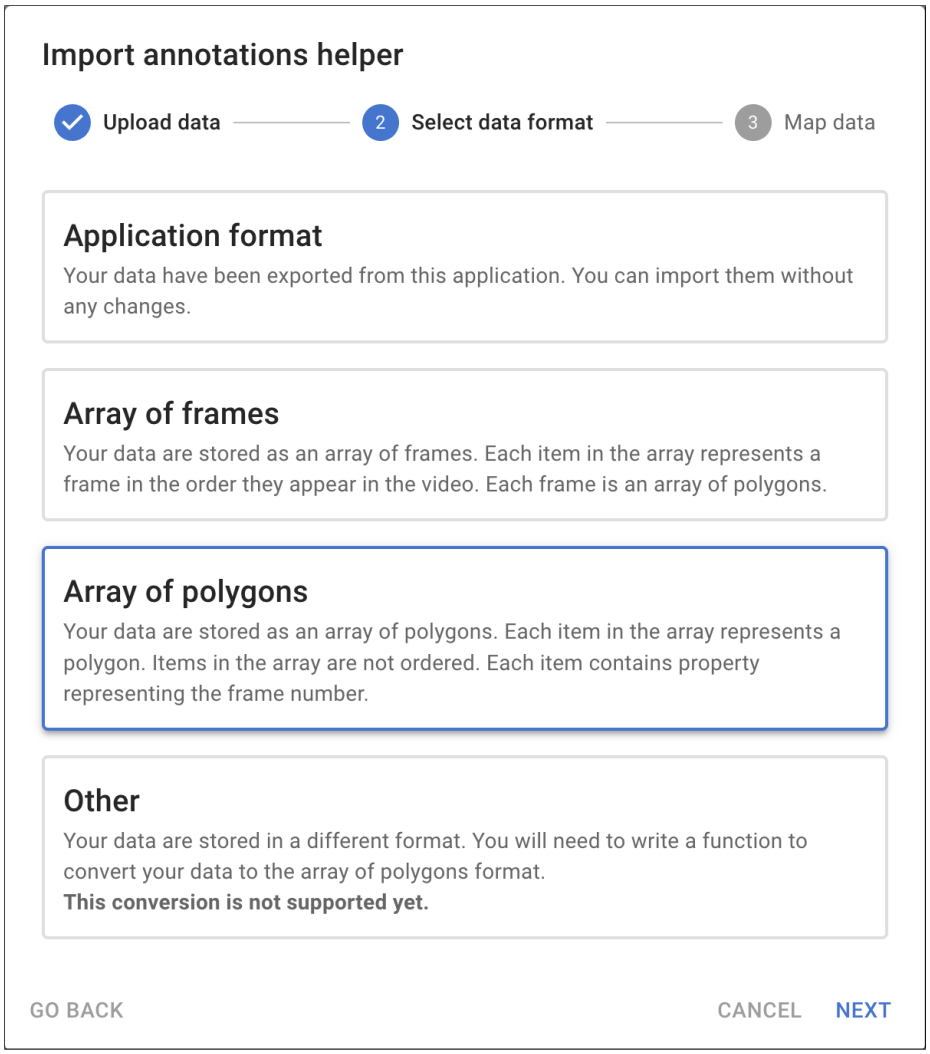

Figure 5.2: The second step of the import flow: selecting uploaded file data format.

Step 3: Mapping data

The final step of the import flow is to convert the user annotation object's format to the internal application format. This conversion requires mapping the paths of the annotation object in the user's file to the paths of the annotation object stored in the application. The mapping data interface is presented in the Figure 5.3.

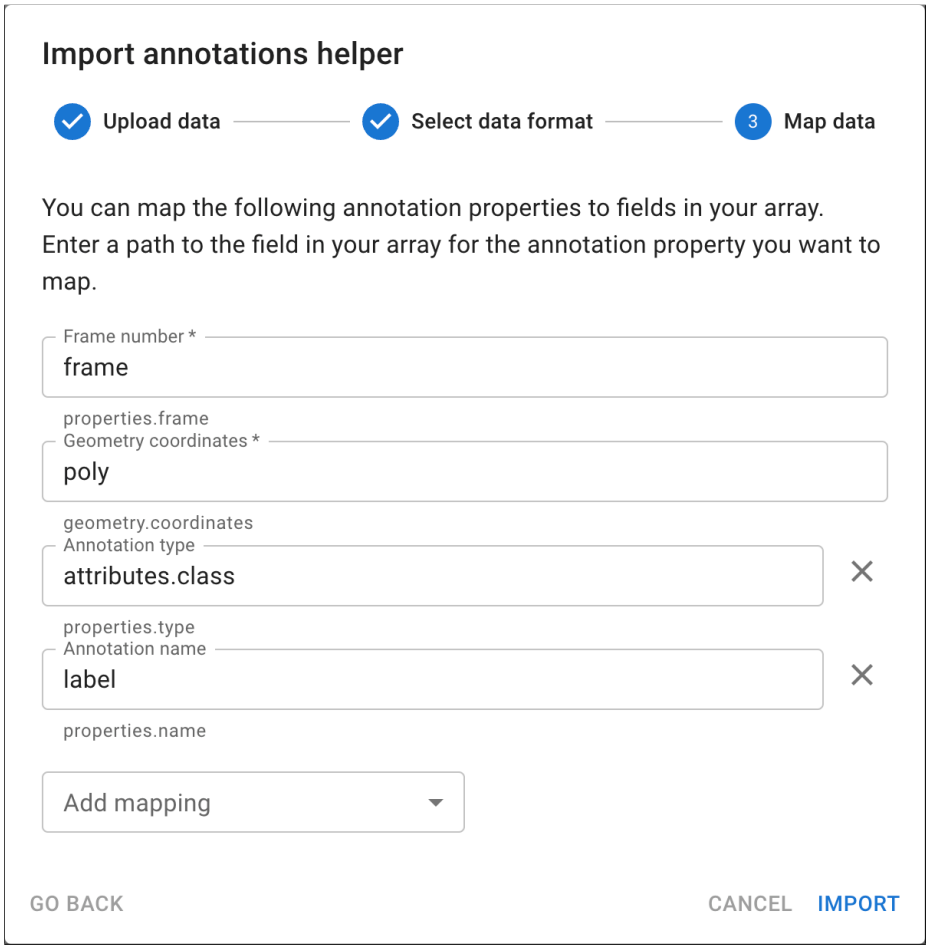

Figure 5.3: The third step of the import flow: mapping the user's annotation object fields to the application annotation object fields.

There are two kinds of fields during this step: required and optional. The import flow can be completed by mapping the required fields. If the user's file contains additional details about the annotation, optional fields may also be mapped. The following table shows the fields and how they should be mapped during this step. The fields and their kinds based on the user's file format are displayed in the Table 5.1.

	Array of frame	Array of polygons
Frame number	Optional	Required
Polygon coordinates	Required	Required
Annotation type	Optional	Optional
Annotation name	Optional	Optional
Annotation color	Optional	Optional

Table 5.1: Mapping fields and their kinds.

Export

Export functionality enables users to download annotations created for the video. This feature can be used to train machine learning models on the data that is manually val-

idated using the application. The exported file is in GeoJSON [1] format, which allows for integrating the annotation data set with other positional tools and libraries for further processing.

5.2 Data storage

During the development, it was decided to store all of the application's data in the Redux library. It allows for storing all data in one container and makes this container the single source of truth. This concept ensures that, throughout the whole application, all saved data is consistent. It also simplifies the state management process and enables easier debugging and testing.

Another advantage of the library is that it can pass data directly between multiple components simultaneously. Because the application uses multiple components, it is difficult to transmit and store data between them all without utilising a storage management framework like Redux.

Internal application data format

The primary goal of the application is to work with existing and new annotations on the video. All annotations are kept in the Redux store. Every annotation has all the properties specified in the standard and is saved in GeoJSON format.

Figure 5.4: Annotation storage structure.

Coordinates and properties are the two fields of the annotation object that are most important. The polygon's array of vertices makes up the coordinates field. Every coordinate holds a position based on pixels and represents a point on the video. Name, frame index, colour, type, and other extra information about the annotation are contained in the properties field, which is a key-value structure.

Components configuration

In order to perform particular features, the application uses certain components. To display the annotations for a project across time, a timeline is used using the `vis-timeline` library. A video player that can handle and play a variety of video formats is implemented using the `Video.js` library. For each of these libraries to function as intended, the proper configuration has to be specified.

The component configuration for the timeline component is shown in Listing 1. To make the timeline look like it was designed to fill the whole bottom area of the screen under the video player component, the setup sets the initial scale and transition properties. Additionally, it provides the function to be used in order to format labels and configures how the time labels and annotation data items should be displayed. Additionally, the

component's setup includes properties that define the user's possible interactions with the timeline.

```
{
  // Layout
  align: 'auto',
  height: '100%',
  min: new Date(0),
  zoomMin: 1000,
  zoomKey: 'ctrlKey',
  orientation: 'top',

  // Labels
  showMajorLabels: false,
  format: {
    minorLabels: timelineLabelsFormat,
  },

  // Data
  snap: null,
  stack: false,
  groupHeightMode: 'auto',

  // Interactions
  selectable: false,
  showCurrentTime: false,
  horizontalScroll: true,
  verticalScroll: true,
}
```

Listing 1: Timeline component configuration.

The component configuration for the video player is shown in Listing 2. The configuration defines the aspect ratio and the fill property, which makes the video player component fill all of the available space inside its container. These parameters determine how the video player should look. Additionally, the configuration defines the properties to preload a video and activate the behaviour that buffers more video content than it does by default, improving the quality of the video watching experience. The last property mutes the video file's audio track in order not to distract the user with sounds while working.

```
{
  // Layout
  aspectRatio: '16:9',
  fill: true,
  fluid: true,

  // Playback
  preload: 'auto',
```

```
  enableSmoothSeeking: true,

  // UI
  muted: true,
}
```

Listing 2: Video player component configuration.

5.3 Annotations creation

Users can create their annotations in the form of polygons with multiple coordinates. Each polygon must contain at least three vertices to be a closed polygon. To create them, the user only needs to start clicking with their mouse on the video, and the new vertices of the polygon being created will appear. Once the user clicks on the initial point of the polygon, it closes, and an annotation is created. The default attributes are assigned to the properties of the polygon, including the polygon's name, colour, type, and identifier. These attributes can be modified by the user after the annotation is created.

Coordinate mapping

Multiple calculations are being done when the user creates a polygon. The application allows the user to move video inside the viewport by horizontal and vertical axes and zoom the video. The video itself might also be contained in a block of a smaller size than the original size of the video. Those transformations require the following computation to be done during the annotation creation process in order to properly adjust the annotation display over the video:

1. The user cursor's horizontal and vertical positions are multiplied by the inverted zoom value. The resulting value represents the polygon's vertex position on the viewport of the video without zoom.

2. The difference between the user cursor position and video move offsets is calculated. The resulting value represents a point on the viewport without any user transformations.

3. The difference between the video's original size and the video's viewport size is computed. The user's cursor position is multiplied by that difference. The result value represents a point on the pixel grid of the original video.

The resulting value computed after all the transformations is recorded in the annotation vertecies array. This data-storage format was selected to simplify the import and export processes. Thanks to this format, no changes are needed before exporting the resulting annotations file from the application because all the vertecies coordinates are already defined as the coordinates on the video pixel grid.

The only problem with this approach is the need to always convert the coordinates whenever the user manipulates the video viewport or the polygons themselves. This means every render of the polygons causes those three calculation steps to be computed again.

5.4 Backend

The implemented application includes a server (backend) implementation. Implemented functionality ensures that users can create their own user accounts. Their accounts include videos and projects, enabling people to work on several projects at once. Videos and annotations are linked to the projects. This makes it possible for users to split the annotation process into separate stages and save their annotating progress. Furthermore, it ensures that several users may collaborate on the same annotations without exchanging exported annotation files.

Database model

The database utilised for the backend functionality contains five entities implemented in six database tables. The complete entity relationship diagram is presented in Figure 5.5.

Figure 5.5: Entity relationship server implementation diagram.

The `User` entity represents a registered user in the application. It contains such properties as email, username, password, and refresh token. The user may be uniquely identified by using the first two properties. The username property is also used for the authenticating process. The password property contains a hashed user password. The hash functionality

for password hashing is provided by the `bcrypt` library[2]. The refresh token property is also used for the authenticating workflow.

The `Role` entity contains the only property, which is named role. This entity is used to specify the roles of the user. Roles determine the rules, specifying which entities and their corresponding fields the user has access to and which type of access the user is authorised to perform.

The roles of users are specified in the `UserRole` database table. The `User` and `Role` entities use the table as a Many-To-Many connection table.

A user's annotation project is represented by the `Project` entity. It contains such properties as name, annotations, author, and video. The author property is used in order to restrict or grant access to the project to certain users. The annotations property contains a JSON array, which is used in the frontend internal storage. When the project data is obtained from the server in the frontend code, the annotations from the server response are moved to the frontend storage.

The user-uploaded video file is represented by the `Video` entity. Users can annotate data from the same video according to their needs without having to upload the same file multiple times while using the same video for multiple projects. The video entity includes attributes like the author, poster, and path to the file within the backend Docker container, along with additional information related to the video file that was acquired using the `ffmpeg` tool[3]. The author property for this entity implements the identical functionality as for the „Project“ entity authorising users' access to the video file. The poster property defines the image file used as a thumbnail image for the video. A set of posters is generated using the `ffmpeg` tool when the video is being uploaded to the server.

The video's thumbnail image is represented by the `Poster` entity. It includes such properties as a path to the thumbnail image inside the backend Docker container and a relationship to the video the image was generated from.

Authentication

Before using the application, users have to register for an account. Any unauthorised user can access the backend endpoints for authentication and registration. Users obtain an access token and a refresh token after passing the authentication process. The JWT[4] standard is being used by these tokens.

Users' requests to the backend are authorised and authenticated with the access token. This token has a seven-day expiration interval, which is a relatively short duration. If a user forgets to log out of an application on a device and is unable to access it again to revoke access, the token is supposed to expire to prevent unauthorised access to the user's data.

The refresh token can only be used to get a new access token in the event that the current one expires or becomes invalid for any other reason. This kind of token has a thirty-day expiration period. Because the access token may expire sooner than the refresh token, this kind of token has a longer expiration duration.

Both the refresh token and access token are signed using HMAC[5] and the SHA-256[6] hashing algorithm. This ensures efficient and robust functionality against multiple attacks

[2] `https://github.com/dcodeIO/bcrypt.js`
[3] `https://ffmpeg.org/`
[4] JSON Web Token
[5] Hash-based Message Authentication Code
[6] Secure Hash Algorithm 256 bits

[8]. A hashed token is sent from the client side to the server side with each request. The server validates the HMAC value using the correct key and the signing input as input to the HMAC SHA-256 function, then takes the output and determines if it matches the token signature. If it matches exactly, the HMAC is valid, and the server will extract the information contained in it and use it to determine what actions the user is authorised to perform [5].

The authentication token parameters are specified by four environmental variables used in the backend implementation. These parameters and their descriptions are presented in Table 5.2.

Parameter name	Parameter description
`JWT_ACCESS_SECRET_KEY`	Specifies the key used for signing the access token using HMAC standard.
`JWT_ACCESS_EXPIRES_IN`	Specifies the amount of time during which the access token is considered valid
`JWT_REFRESH_SECRET_KEY`	Specifies the key used for signing the refresh token using HMAC standard.
`JWT_REFRESH_EXPIRES_IN`	Specifies the amount of time during which the refresh token is considered valid

Table 5.2: Environmental variables description for backend authentication tokens configuration.

5.5 Build process

The entire application consists of three tiers: frontend, backend, and database. Docker is used to create three parts because of its ability to run images in an isolated environment.

Each part of the application requires a Docker image to be created before it can be deployed to the production environment. The Docker image is a read-only template with instructions for creating a Docker container. Every application tier has a Docker configuration file that specifies the exact flow of how these images must be created. Dockerfile is the name of this file, which defines the steps needed to create the image and run it.

With further customisation, the application tier images are based on another Docker image. The generated images are used to launch Docker runnable instances, known as containers. Containers for each application part must be created in order for the application to work properly[7].

Frontend build

The frontend build process consists of two stages. The first stage consists of installing the project dependencies required to build the project and building the project itself. The official Docker image for NodeJS[8] is used as the basic image for this step because it comes with all the tools needed to complete the build process without requiring the installation of other packages. The code of the Dockerfile for this stage is listed in Listing 3

The output of the first stage is a directory with files, which should be served by any web server. This output is used in the second stage of the frontend build process. The

[7] `https://docs.docker.com/get-started/overview/`
[8] `https://hub.docker.com/_/node`

```
FROM node:18-alpine AS build

ARG VERSION
ARG VITE_APP_API_URL

WORKDIR /build

COPY package.json .
COPY yarn.lock .
COPY tsconfig.json .

RUN yarn install --freeze-lockfile

COPY . .

RUN yarn build
```

Listing 3: Dockerfile for the frontend build step.

second stage utilises the Nginx official image[9] to build a customised frontend image. The goal of this stage is to execute the web server using the contents of the copied directory by copying the output directory from the previous stage to the folder in the web server image. The Dockerfile for the second stage is listed in Listing 4.

```
FROM nginx:1.18-alpine

COPY docker/nginx.conf /etc/nginx/nginx.conf
COPY --from=build /build/dist /frontend/build
```

Listing 4: Dockerfile for the frontend server configure step.

Once this phase is finished, a Docker container can be run using the created frontend image.

Backend build

The official Docker image for NodeJS[8] is also the basic image for the backend Docker image. Unlike the frontend image, the backend image has only one stage. It is not necessary to copy the backend directory created during the build to the web server image. A command-line interface must be available for the backend executable to use in order to interact with the operating system that is executing inside the container. To meet this need, the built backend code runs using the NodeJS executable.

Listing 5 shows the Dockerfile used for the backend image creation process. Installing extra software is the first step in the backend build stage because it is necessary for the backend code. There are no NodeJS libraries included in this step; it is only operating

[9] https://hub.docker.com/_/nginx

```
FROM node:21-alpine as backend

ARG VERSION

WORKDIR /app

RUN apk --no-cache add --virtual builds-deps build-base python3 ffmpeg

COPY package.json .
COPY yarn.lock .
COPY nest-cli.json .
COPY tsconfig.json .
COPY tsconfig.build.json .

RUN yarn install

COPY . .

RUN yarn build

EXPOSE 3000

CMD ["sh", "-c", "yarn start:prod"]
```

Listing 5: Dockerfile for the backend image build.

system software. The backend NodeJS libraries are installed after this software is installed. These libraries include the NestJS framework and the Postgres database connection library.

Database build

Customisation of the database Docker image is also required. The official Postgres Docker image serves as the base image for the database image. The ability to execute any SQL (Structured Query Language) file in a specific directory is provided in this base image. The database should have predefined tables, procedures, and triggers written for the project. The backend source code contains SQL files that define these database entities. After being copied to the Postgres Docker image's directory, these files are executed, and the necessary database entities are created when the image is launched inside the container. The Dockerfile for the database image is listed in Listing 6.

```
FROM postgres:16-alpine as database

ADD ./docker/postgres/pg_hba.conf /var/lib/postgresql/data/
ADD ./docker/postgres/postgresql.conf /var/lib/postgresql/data/

COPY ./docker/postgres/migrations /docker-entrypoint-initdb.d
```

Listing 6: Dockerfile for the database image build.

Chapter 6

Conclusion

The aim of this thesis was to develop an open-source video annotation tool with the ability to import, modify, and export video annotations. The tool was implemented and assessed with real world data to evaluate its performance and usability. The results showed that the tool was useful in creating annotations and improving annotation accuracy for a variety of video annotation tasks.

In the beginning, the existing video annotation tools were selected and analysed in order to identify their strengths and weaknesses. Their advantages were examined from the perspective of whether they could be implemented within a specified time and whether they were required to cover the assignment requirements. Several of their strengths were considered less important and might be implemented in future versions of the application. The weaknesses of the analysed tools were taken into account in order to strengthen them and create a more effective tool.

These studies were used in order to create a user interface that included the most useful components and features found in the existing tools. The primary objective of this part was to allow users to manipulate their videos and annotations in an intuitive interface, which will meet the requirements specified in the thesis assignment. Two design iterations were needed in order to create the interface that would satisfy the previously mentioned objectives. Alongside the user interface development, appropriate libraries were selected that were employed in the implementation of the tool. The final application was tested with actual data from the real world, which was provided by the supervisor of the work. The application satisfied all the use cases and needs defined at the beginning of the work.

There are still opportunities to expand and enhance the tool. One of the improvements might be the capability to import and export data in the most widely used formats by the other existing video annotation tools. By doing this, the user's annotation process might be streamlined. Another improvement may be the option to save annotation coordinates in a relative format instead of the absolute format that is currently used for import and export. With this change, users would be able to resize videos and still utilise the same annotation data. Enabling collaboration on projects by allowing users to use workspaces for real-time user input might be considered a long-term feature.

The final application client tier source code is available at `https://github.com/NickSettler/video-annotation-tool`. The backend tier is available at `https://github.com/NickSettler/video-annotation-backend`. The final application is also deployed on a server and is available via `https://video.settler.tech/`.

Bibliography

[1] Butler, H.; Daly, M.; Doyle, A.; Gillies, S.; Schaub, T. et al. *The GeoJSON Format* RFC 7946. RFC Editor, august 2016. Available at: https://doi.org/10.17487/RFC7946. [cit. 2024-03-21].

[2] Corporation, C. *CVAT* online. Available at: https://www.cvat.ai/. [cit. 2024-04-12].

[3] Dilek, E. and Dener, M. Computer Vision Applications in Intelligent Transportation Systems: A Survey. *Sensors*, 2023, vol. 23, no. 6. ISSN 1424-8220. Available at: https://www.mdpi.com/1424-8220/23/6/2938.

[4] Guimaraes, A. A. R. Detecting Zones And Threat On 3D Body In Security Airports Using Deep Learning Machine. Zenodo, 2018. Available at: https://zenodo.org/record/1189345.

[5] Jones, M. B.; Bradley, J. and Sakimura, N. *JSON Web Signature (JWS)* RFC 7515. RFC Editor, may 2015. Available at: https://doi.org/10.17487/RFC7515.

[6] Khemasuwan, D.; Sorensen, J. S. and Colt, H. G. Artificial intelligence in pulmonary medicine: computer vision, predictive model and COVID-19. *European Respiratory Review*. European Respiratory Society, 2020, vol. 29, no. 157. ISSN 0905-9180. Available at: https://err.ersjournals.com/content/29/157/200181.

[7] O'Connor, Z. Colour, contrast and gestalt theories of perception: The impact in contemporary visual communications design. *Color Research & Application*, 2015, vol. 40, no. 1, p. 85–92. Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/col.21858.

[8] Ravilla, D. and Putta, C. S. R. Implementation of HMAC-SHA256 algorithm for hybrid routing protocols in MANETs. In: *2015 International Conference on Electronic Design, Computer Networks & Automated Verification (EDCAV)*. 2015, p. 154–159.

[9] Studio, L. *Open Source Data Labeling* online. Available at: https://labelstud.io/. [cit. 2024-04-12].

[10] V7Labs. *V7 Gen AI & Darwin | Accelerate AI Development & Automation* online. Available at: https://www.v7labs.com/. [cit. 2024-04-12].

[11] Walker, M.; Takayama, L. and Landay, J. A. High-Fidelity or Low-Fidelity, Paper or Computer? Choosing Attributes when Testing Web Prototypes. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 2002, vol. 46, no. 5, p. 661–665. Available at: https://doi.org/10.1177/154193120204600513.