Data Annotation and Management tool

Quan Dao

Haaga-Helia University of Applied Sciences
Degree Program in Business Information Technology
Project-type Thesis
2022

Abstract

Author(s) Quan Dao
Degree Bachelor of Business Administration
Report/thesis title Data Annotation and Management tool
Number of pages and appendix pages 45 + 2

The aim of this thesis is to provide the annotation tool to the commissioning company, Verso Vision Oy, with a added-in data management feature which are realiable, dynamically, and easy to configure.

Verso Vision is one of the leader in the field of using AI vision technology to prevent falls of patients in hospitals. The company's softwares and solutions have been used and trusted widely in Finland, such as HUS, Jorvi Hospital, Hyvinkää Hospital, …, and also in other countries.

As the solution depended heavily on AI, the data annotation is one of the most important part to help us success in our work. With this annotation and managenent tool, the AI team can do the annotation and organize the data easily, reducing the efforts for the models training sessions.

Because of strictly following GDPR (General Data Protection Regulation) laws, any non-public data related to the company will be redacted in this thesis.

The final products will be evaluated and used by the commissioning company, and recommendations for the future improvement will be given after that.

Keywords
React, Nodejs, Redux, Google Protobuf, Mongo, File Systems, TypeScript

Contents

1 Introduction

Artificial Intelligence, or AI, has been playing a vital role in our every day's activities in the last decades. By the time, AI has showed its incredibly powerful by enhancing the productive, efficiency, and accuracy in all the industries, and the "magic" behind that makes AI so special is just about the data.

1.1 Theoretical Background

1.1.1 Introduction to AI/ML

AI is a concept of computer or robot, with the help of software and various kind of data, can do human tasks that require human intelligence and adjustment.

Machine learning, or ML, as the name suggested, is a technique in which we train software models by using data. It is an important part of AI because generally, AI just mimic and "learn" from human intelligence. The more and better-quality data you fetch the model, the higher chance AI can have good performances and accurate results.

1.1.2 Introduction to data annotation

Computer is just machine. It cannot understand the information as the way human brains do. In fact, it needs to be told and taught in its own languages. This is where data annotation comes to play. It is the task regarding translate information and content of subjects into simple terms so AI models can learn from it and make predictions of similar patterns later. (TELUS, 2021).

Since data is the core of machine learning, there is no doubt that data annotation is one of the most important tasks. Without it, AI would never learn and do anything useful to solve our problems. Indeed, data scientists spend a significant portion of their time preparing data, such as remove nonstandard data, fixing data, etc.

There are many types of data annotation existing now, but the most popular ones are text and image annotations. Because of the scope of the project, I will focus on the image annotation type mainly.

Image annotation is the task of labeling digital images, giving users the basic information about the objects or human present in the image.

The number of image annotations can vary depends on the use cases of the project. Some of the most popular image annotations are poses annotation, class annotation, and ROI annotation.

Human Pose Estimation (HPE) is a way of identifying and classifying the joints in the human body (Nilesh, 2022). In simple terms, after you draw the essential key points and connect them as pair, you can have a human skeleton. It is necessary to note that not all points can be connected. For example, the right shoulder must connect with right wrist, not left ankle.

There are 16 key points in a basic pose annotation, starting from nose, eyes to knees and ankles. However, with the rapidly development of data model, the demands for high accuracy have raised significantly. Nowadays, poses are defined by up to 33 key points, with the surprisingly details such as eye inner and eye outer.

In contrast to pose annotation, image classification focuses on simpler tasks. It is to identify what image represents, what are the topic of the image or the “classes” of it. Since the tool is developed to prepare data for fall-detection AI models, the classes I defined will be about the humans, or patients, existing in the images. Some examples can be WALKING, SITTING_ON_BED, or FALLING.

The last type of image annotation I would like to clarify here is ROI annotation (Region of Interests). In this annotation, the goal is to find the location (established by using bounding boxes) of individual objects within the image. By using rectangles or polygons, figure that has more than 3 angles and sides, we can define the objects existing in an image, commonly known as bounding boxes

2 Theoretical Framework

This chapter is about to introduce and clarify all the technical tools which are used to build and make the tool successfully. First is going through the programming language which is used for the project. Next, the main format and structure data, Protobuf, is discussed. After that are the databases and backend tools for creating server. Finally, the frontend and the supported libraries will be presented. These help to give a general structure of the development process.

2.1 Git

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. (Git, 2022)

In the simple terms, Git is a tool which allows multiple developers working in the same projects conveniently, improving the productivity and collaborations. The following will describe the most important features of Git in the annotation tool, which are branch and merge, submodule, and tag.

2.1.1 Git branch and Git merge

Branching is one of the most essential features Git provides. Branching means you diverge from the main line of development and continue to do work without messing with that main line. (Git, 2022) Thus, any changes which are made after branching will be stayed only in that branch, while the main branch will not be touched and be the same. This is useful because it enables multiple developers, or teams, working on different features which are all started from a stable base. Even when you work alone in a project, it is recommended to apply branch technique since you can have more controls on different steps of the development process.

In practice, branching alone is not useful at all without merging because in the end, what we need is a single branch, or version, for the production. Git merge command integrate a subbranch into the main branch, updating the main branch with all the changes created by the subbranch. After merging, the subbranch becomes stale and can be deleted from the project.

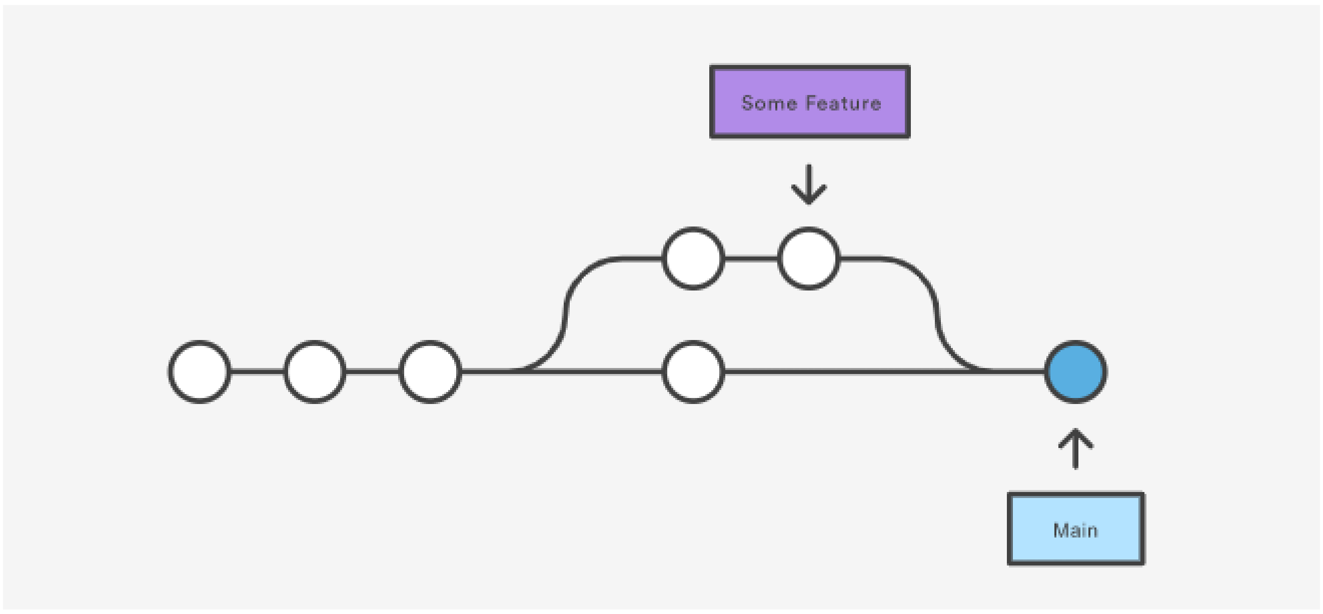


Figure 1: Git branch and merge (Atlassian, 2022)

2.1.2 Git submodule

“Submodules allow you to keep a Git repository as a subdirectory of another Git repository. This lets you clone another repository into your project and keep your commits separate.” (Git, 2022)

Submodule allows making use of multiple projects inside one project without extending the project itself. It helps maintaining process easier and more manageable. Submodule is even more powerful since it lets a project subscribes to specific version, or even commit, of other projects. Therefore, the same project can be submodule in different shape and version into others.

2.1.3 Git tag

In large-scale projects, there could be dozens of branches created and merged during the development process, letting the track becomes difficult – not mentioning to the subbranch might be deleted after merging. During the development, there normally are some phases, or versions, that we want to public to the production before developing new features, and this is where tag takes its part.

“Git has the ability to tag specific points in a repository’s history as being important. Typically, people use this functionality to mark release points (v1.0, v2.0 and so on)”. (Git, 2022)

When the product is stable and works with some features, you can release it by checking out a tag. Unlike branch which can be added new features, tag is static and always points to specific commit in the development process.

2.2 TypeScript

First, TypeScript is just JavaScript with static typing. JavaScript, often known as JS, is a language which provides both object-oriented and functional programming. It is one of the most popular programming languages used among software developers due to its lightweight, flexibility, and easy to use. JS runs on the client side, being used to create dynamic and interactive web applications on the browser. Nowadays, JS can be used to create backend server, making it the best candidate for building web-based application (About JavaScript, 2022)

There is a fact that any JS developer has faced the error of "'undefined' is not a function" more than once. it is one kind of hidden errors which are not easy to define in the development, only show in the run time.

At this point, TypeScript comes as a rescue since it supports the type checking in the compile time. Considering type checking means that any time we declare a variable, we must tell the machine which type that variable is, whether it is number or string or others. It can be more complex like declare types for objects and functions.

TypeScript is static type language which checks the type errors immediately before the code is executed, helping developers to recognize these errors immediately without running the program. Moreover, with TypeScript, it is easier to refactor code without breaking it significantly, since every component has types defined beforehand. (TypeScript for the New Programmer, 2022) (Gints D. & Olga B., 2020)

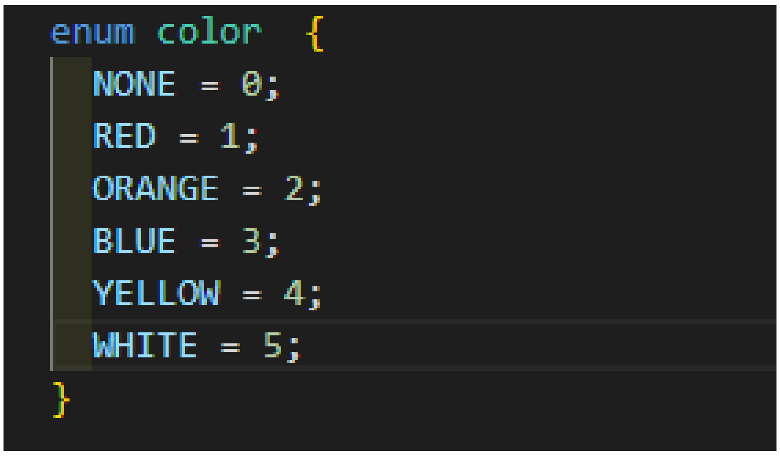
2.3 Protobuf (or protocol buffers)

"Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data". The data structure is defined as schemas first in proto files, and all the data come and go are enforced to follow these specific set of rules. It is like XML, stands for extensible markup language, but far faster and easier to implement. Protobuf can be used in various programming languages, from the old C++, Java to the new ones like Python or JS. (Protocol Buffers, 2022)

To better understand of Protobuf, it is necessary to mention about JSON, stands for JavaScript Object Notation. It is a lightweight format for storing and transporting data. (What is JSON?, 2022) JSON is widely used because of its human-readable-format and easy to understand.

However, JSON is limited to certain types of data, mostly string and number, whereas Protobuf covers a wide variety of data types when compared to JSON. Even enumerations (with the keyword *enum*), objects (keyword *message*), and methods (keyword *service*) can be serialized with Protobuf. (Protobuf vs JSON, 2022)

“An enumeration is a complete, ordered listing of all the items in a collection. The term is commonly used in mathematics and computer science to refer to a listing of all of the elements of a set”. (Enumeration, 2022). Enums are typically used when there are a set of predefined values which will not likely be changed in the future. By default, the first field of enumeration is 0 or “None”.

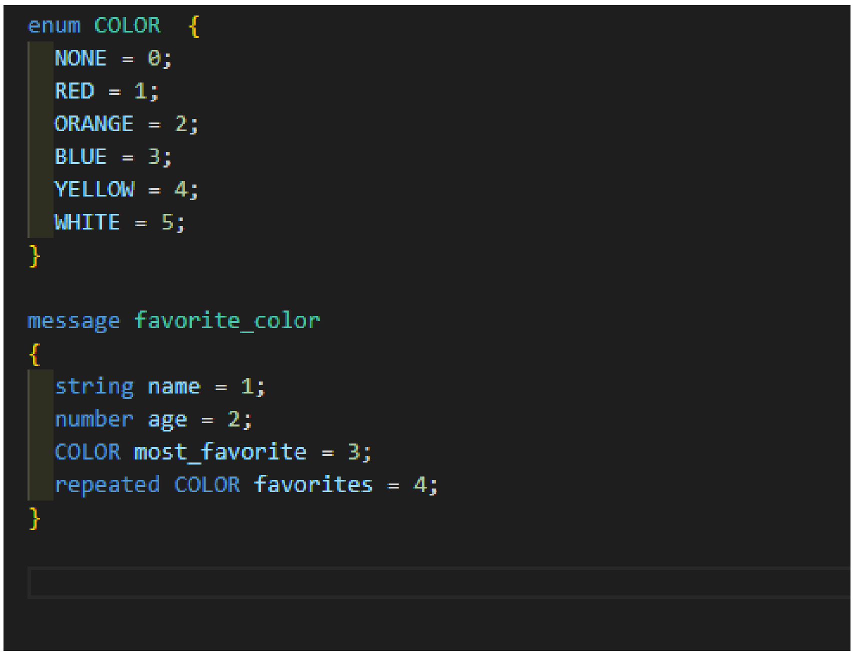


```
enum color  {
    NONE = 0;
    RED = 1;
    ORANGE = 2;
    BLUE = 3;
    YELLOW = 4;
    WHITE = 5;
}
```

Figure 2: Enumeration in Protobuf

Besides, JSON contains the only message and not schema, whereas Protobuf not only has messages but also includes a set of rules and schemas to define these messages. (Protobuf vs JSON, 2022)

Protobuf type is defined in proto files and able to be serialized and deserialized during the message transfer. Since the message is serialized into binary before sending, it is considered much faster than JSON message, and can be benefit when the size of data is large.



```
enum COLOR  {
  NONE = 0;
  RED = 1;
  ORANGE = 2;
  BLUE = 3;
  YELLOW = 4;
  WHITE = 5;
}

message favorite_color
{
  string name = 1;
  number age = 2;
  COLOR most_favorite = 3;
  repeated COLOR favorites = 4;
}
```

Figure 3: Protobuf message with enumeration

Due to the serializable, Protobuf message is used heavily in gRPC, which is a Communication Protocol whose message sizes tend to be dramatically smaller than those in REST API. Nevertheless, because of the scope of this project, I will not go into details of gRPC.

In summarize, the reasons we decide using Protobuf format are because of its schema types which support predefined enumerations, the ability to serialize and deserialize message since we need to send images in binary data between browser and server, and easy to convert to JSON format if needed. Moreover, with the help of Git Submodule, we can define all the Protobuf data structure in separate repository, implement it by adding as submodule into the needed repository, and based on the programming language, which is used in the project, compile Protobuf so that it can work along within the project.

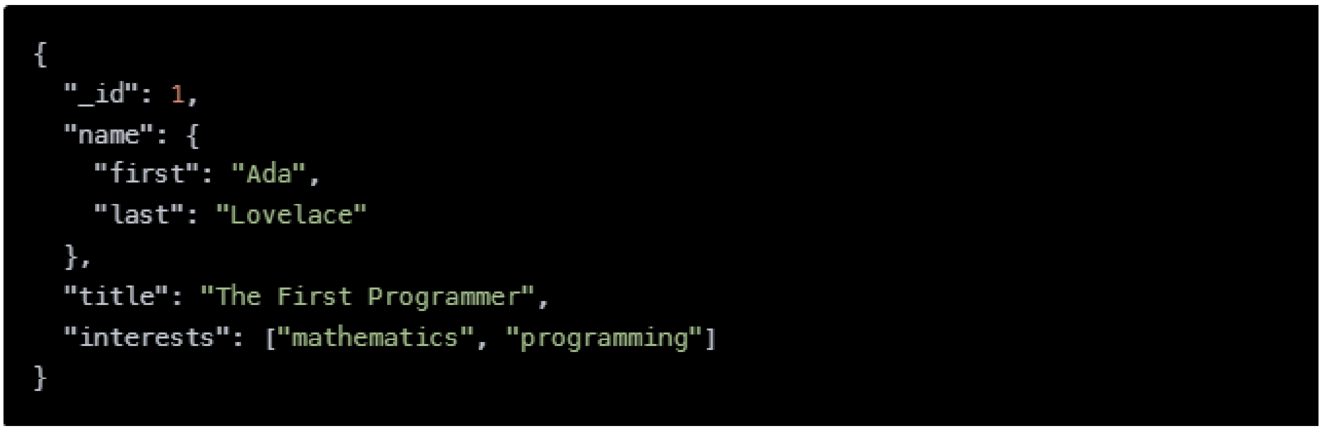
2.4 Databases

Databases are where the data are stored and can be accessed as well as modified if needed. There are two main types of databases available, which are SQL and NoSQL database. While the former is relational database where data are saved in tables with the strict structures and closely linked to each other, the latter is non-relational where data are in unstructured format and can be saved in various types. Since the data annotation are in complex type and can be updated in the future, NoSQL database was selected as the main database of the application because of its flexibility.

2.4.1 MongoDB

MongoDB is an open-source document database built on a horizontal scale-out architecture that uses a flexible schema for storing data. Instead of storing data in tables of rows or columns like SQL databases, each record in a MongoDB database is a document described in BSON, a binary representation of the data. Applications can then retrieve this information in a JSON format. (Why Use MongoDB and When to Use It?, 2022)

With the less needed SQL query, I choose mongo as the database to store the annotation data. Another reason is in the end of annotation process, what we need is some JSON files which we will be delivered to the AI team to train the models, and in mongo, everything is JSON document. We can easily export and import data to JSON files with mongodump and mongorestore.



```
{
  "_id": 1,
  "name": {
    "first": "Ada",
    "last": "Lovelace"
  },
  "title": "The First Programmer",
  "interests": ["mathematics", "programming"]
}
```

Figure 4: Mongo document with JSON format. (Why Use MongoDB and When to Use It?, 2022)

Mongo can have advantages in our use cases also when considering to models in mongo. Models are fancy constructors compiled from Schema definitions. An instance of a model is called a document. Models are responsible for creating and reading documents from the underlying MongoDB database. (Mongo models, 2022) With the current solid data structure using Protobuf, I can define the data document with the Protobuf type, creating a coherent data flow in the application.

2.4.2 Redis

Redis is an open source (BSD licensed), in-memory data structure store used as a database, cache, message broker, and streaming engine. Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster. (Introduction to Redis, 2022)

In other words, Redis is the key-value and in-memory database.

One of the strengths of Redis is that it is super-fast and considered to be one of the highest performance databases. Also, Redis is simple and easy to use. It has flexible data structures and supports almost all data structures. Besides, Redis has zero downtime or performance impact while scaling up or down. Moreover, it is open source and stable. (Redis – What and Why?, 2022)

Along with these advantages, Redis has a cool feature about keys auto expired after a specific predefined time, which is the main reason I select it to manage user sessions and dispatching images. Since we need to dispatch the free images to users anytime they start the application, there must be a database to manage that user's images' state, and we do not want to use Mongo for any other tasks rather than storing the annotations because they are valuable data. With Redis, we can easily manipulate a simple database to manage dispatched images of specific users, and if users do not have any actions after expired time, the images will be set to free automatically and dispatch to other users after that.

2.4.3 Network File Systems

Network File System (NFS) is a networking protocol for distributed file sharing. A file system defines the way data in the form of files is stored and retrieved from storage devices, such as hard disk drives, solid-state drives, and tape drives. NFS is a network file sharing protocol that defines the way files are stored and retrieved from storage devices across networks. (Network File System, 2022)

NFS enables system administrators to share all or a portion of a file system on a networked server to make it accessible to remote computer users. Clients with authorization to access the shared file system can mount NFS shares, also known as shared file systems. NFS uses Remote Procedure Calls (RPCs) to route requests between clients and servers. (Network File System, 2022)

Because of needing a place in which we store all the images for the annotation sessions, we create a common File System where all the images are saved and categorized by hospitals. All developers working in the same network can have access to the File System. The others, who work remotely, can get access by using VPN (Virtual Private Network) to connect to the company network and *mount* their local directory to the NFS directory with the help of mount command. This is also where we export finished and accepted data annotations from Mongo to deliver to the AI scientists for the models' trainings.

2.5 Nodejs

Node.js is an open-source and cross-platform JavaScript runtime environment. A Node.js app runs in is a single process, without creating a new thread for every request. This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs. (Introduction to Node.js, 2022)

Besides the above benefits, Node.js is quickly adapt and widely used because of the popular of JavaScript and TypeScript. In the first place, JavaScript was the language for the client side. With the creation of Node.js, now JavaScript developers can use their favorite programming language for both the client and server side, making the application more compatible.

Furthermore, one of the best reasons to choose Node.js is NPM, the Node.js Package Manager. NPM lets you download packages of code created by other developers and use them in your own projects. This means you don’t need to write as much code from scratch. Node.js is home to the largest software library registry in the world. (5 Reasons to Choose Node.js, 2022)

2.5.1 Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. With a myriad of HTTP utility methods and middleware, creating a robust API is easy and quick more than ever. (Express, 2022)

Because Express is built based on Node.js, we can keep all the features of JavaScript and Node to create an Express server, minimizing the learning curve. Nowadays, it normally takes only about 15 minutes to set up a server with Express, which is significantly less time-consuming compared to using Node.js itself. Express has a concise and easy-to-use syntax, helping developers focus more on building the features instead of creating server.

One of the best features Express provides is Middleware. Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. The next middleware function is commonly denoted by a variable named next. (Express, 2022).

Express has built-in request and response object to control the data transferred between client and server. The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on; while the response object represents the HTTP response that an Express app sends to its client (Express, 2022)

In simple terms, Middleware is a extra step in the middle of the client-server communication where developers can do all the side tasks after getting the request and before sending the response. Some of the most popular actions implemented here are validating user's credentials before receiving the data sent or do error handlers whenever it needed.

Rather than those, Express includes many ready-to-use features and middleware which are useful for the development process. For instance, it can host and serve the static files such as images with the built-in middleware "*express.static"*. By using plainly one line of code, we can serve and give client access to the static files, which is important feature for the annotation tool since we need to access to the images to annotate them.

2.5.2 REST API

Before getting to REST, we should be aware of what is API means. API, stands for application programming interface, is a way two or more devices "talk" or communicate to each other. Whether you are a developer or an accountant, API is closely related to your everyday activities. When you buy a train ticket, your mobile acted as client sends the API request to the server and gets the response resources, in this case is whether success or not. Another example is when you use Twitter and click to see the profile of a celebrity, you send an API request to Twitter's server and it will return the information of that person, including his or her name, picture, recent poses, and more.

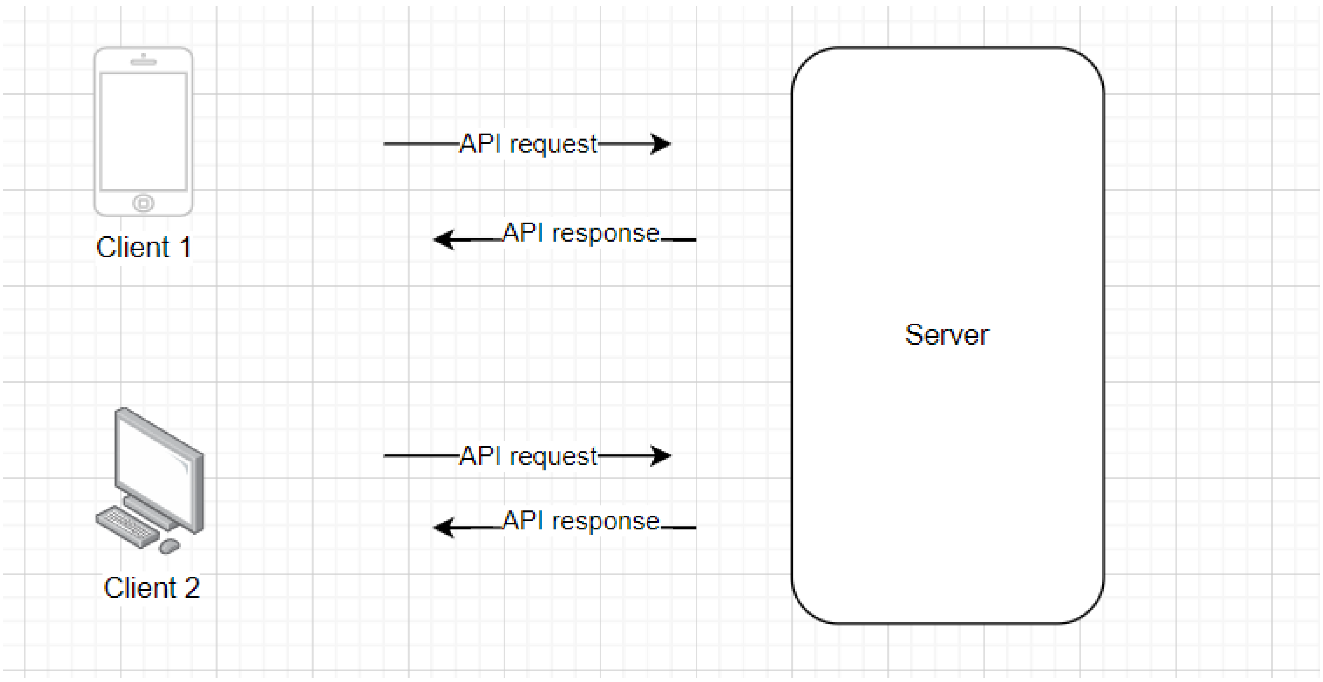


Figure 5: Client-Server communicate by using API

REST is an acronym for REpresentational State Transfer and an architectural style for distributed hypermedia systems. Like other architectural styles, REST has its guiding principles and constraints. These principles must be satisfied if a service interface needs to be referred to as RESTful. (What is REST, 2022)

A Web API (or Web Service) conforming to the REST architectural style is a REST API. (What is REST, 2022). There are 6 principles for a RESTful architecture: Uniform Interface (request include identifier, response include enough information and resource can be modified), client-server (client and server act independently on its own), stateless (each request works on its own without knowing of the previous context), cacheable (response should be defined clearly as cacheable or non, so client can reuse the data for later similar request), layered system (there might be some servers between client and server, doing security, caching, and more), and code on demand (client can request code from server). The last constraint is optional, which means API can still be defined as RESTful even without it. (What is REST, 2022)

2.5.3 File Systems (fs)

Nodejs has a built-in *node:fs* module which enables interacting with the file systems, such as reading directory, files, etc. (File system, 2022). In the tool, the server needs to read the file systems to serve images for the clients, and with *node:fs* methods, the tasks were handled easily and conveniently, either asynchronous or synchronous.

2.5.4 Passport-JWT

Passport is middleware for Node.js that makes it easy to implement authentication and authorization for the application, while JWT stands for JSON Web Token which is a technique of encoding user information using the secret key, which, as the name describes, should be keep confidential within the organization. (Passport-JWT, 2022)

Since there are multiple developers participating in the annotation, we need to differentiate each other and save the information as metadata for the annotation data, which is important for the data management part. Another reason is that it helps to manage sessions and dispatched with unique user's credentials. Nevertheless, the tool is used internally, the security and encoding mechanism are not concentrated at this point.

2.5.5 Ts-proto

Due to our annotation data are in protobuf format, there is a must to “translate” them into the programming language, TypeScript in this case. Ts-proto is excellent candidate to take care of the conversion.

Ts-proto transforms the proto files into strongly typed, idiomatic TypeScript files. (Ts-proto, 2022) It is created based on protobuf-compiler, a tool for compiling proto files to other programming languages.

With other extra tags, we can manage the TypeScript compiled files to follow specific rules. For instances, "*--ts_proto_opt=stringEnums=true"* tells the compiler to generate enumeration with types string instead of integer.

2.6 React

As first introduced in May 2013, React has rapidly developed and become one of the most popular web application development libraries at present. ReactJS is a simple, feature rich, component-based JavaScript UI library, providing a smooth user experience, responsive and can be built and test fast. React community compliments React library by providing large set of ready-made components to develop web application in a record time. React community also provides advanced concept like state management, routing, etc., on top of the React library. (ReactJS – Quick Guide, 2022)

There are many popular websites are created using React library, including Facebook, Instagram, Netflix, etc.

React is just JavaScript UI library, meaning finally, all the React codes will be transformed into JavaScript codes so the browser can read and understand it.

In the web application, the page is represented in the tree structure, or object, and is commonly known as DOM (Document Object Model). By interacting with the DOM, programming languages can change and update the page structure, style, and content (Introduction to the DOM, 2022). React uses virtual DOM to optimize the speed and performance of the application. It compares the trees and updates only the changed place instead of the whole document like usual.

React uses states and props to control the changes and flows of the web interface. While states can be updated, or in more technical term, mutable, and used to store the data and determine the behavior of the component, props are objects which are passed from one component to others. Props are immutable so we cannot modify the props from inside the component. (

React is all about components and their life cycles management, each having its own events. There are 3 main phases of React life cycle: mounting, updating, and unmounting. In the mounting phase, the element is added and put into the DOM. Next, the updating phase is called when component has changes and updates. The final phase is unmounting in which the component is removed from the DOM.

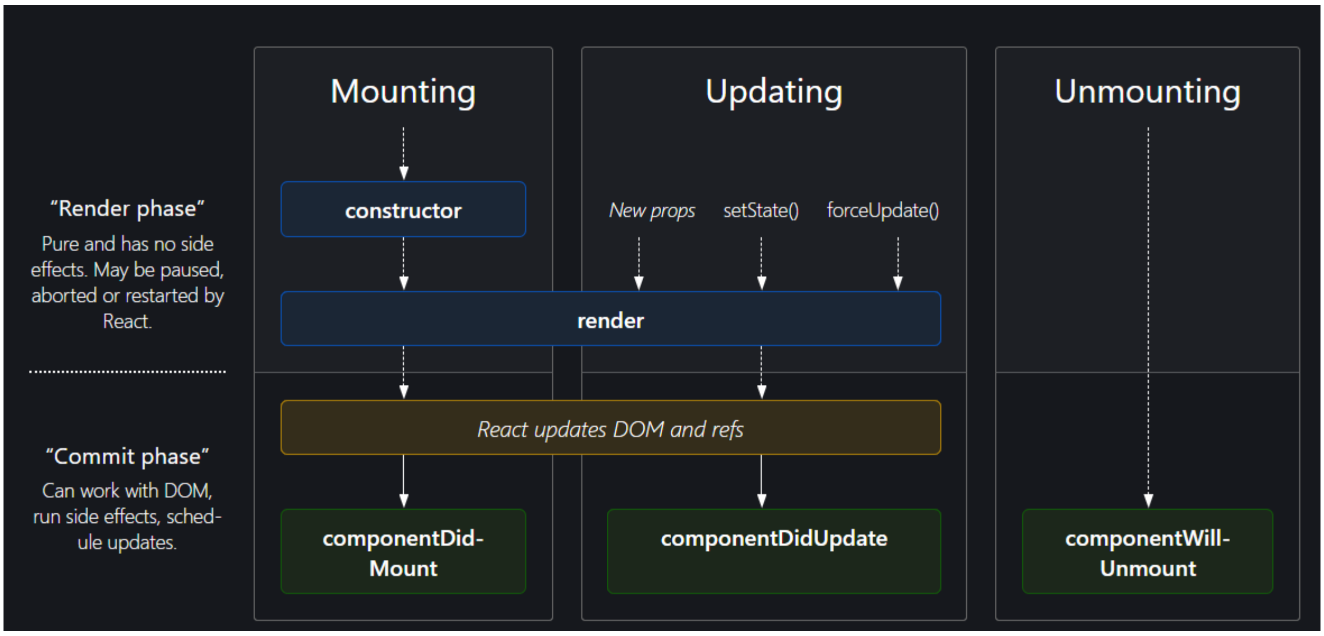


Figure 6: React lifecycle diagram (React-life-cycle-diagram, 2022)

2.6.1 React Hooks

Before version 16.8, React use Class components to manage the lifecycle of its components. Some of the most popular methods are componentDidMount (mounting phase), componentDidUpdate (updating phase), shouldComponentUpdate (updating phase), and componentWillUnmount (unmounting phase). Even though they have the clear structure of the lifecycle, Class components are considered confusing, made the code unreadable and harder to test and maintain. Moreover, the "this" keyword in Class components, in most case, is nightmare for developers.

With the introduction of Hooks in React 16.8, the Class components are no longer used and gradually becomes obsoletes for React developers, though the previous React projects created by Class components are yet remained. However, with the concise syntax and written in functional programming, which became a trend in the last decade over object-oriented programming, Hooks are widely used and adapted into schools and companies.

Hooks are started with the *use* keyword and linked with other word which help it self-descriptive. The following are the descriptions of Hooks that will be used heavily in the annotation tool.

2.6.1.1 useState

useState is a hook that lets you add React state to the component. (Introducing Hooks
, 2022) It accepts one argument as the initial value of the state and return the current state and the function for updating that state.

2.6.1.2 useEffect

Considered of React life cycle and events above, useEffect is the combination of componentDidMount, componentDidUpdate, and componentWillUnmount. (Introducing Hooks, 2022) The hook manages all the side-effects of the components, which makes it one of the most powerful hooks of React.

useEffect accepts 2 arguments. The first is a callback function containing the logic and is executed right after changes were being pushed to the DOM, while the second is dependencies which tell the hook when and how often the logic should be run. There are 3 situations for the dependencies: not provided (run after every rendering), empty (executes only one when the component is first rendered) and has values (being called every time the values are changed, otherwise not). (Dmitri P., 2022)

The last feature of useEffect hook is cleanup, which is similar to componentWillUnmount if we consider about Class component lifecycle. Sometimes we might use some external data by calling the API, and because of the huge data loads or latency, it might take few seconds to finish the fetching process. The cleanup comes to help when we want to change the action or as React official document says, “unsubscribe” to the event. The “unsubscribe” logic is described inside the “return” statement of useEffect hook.

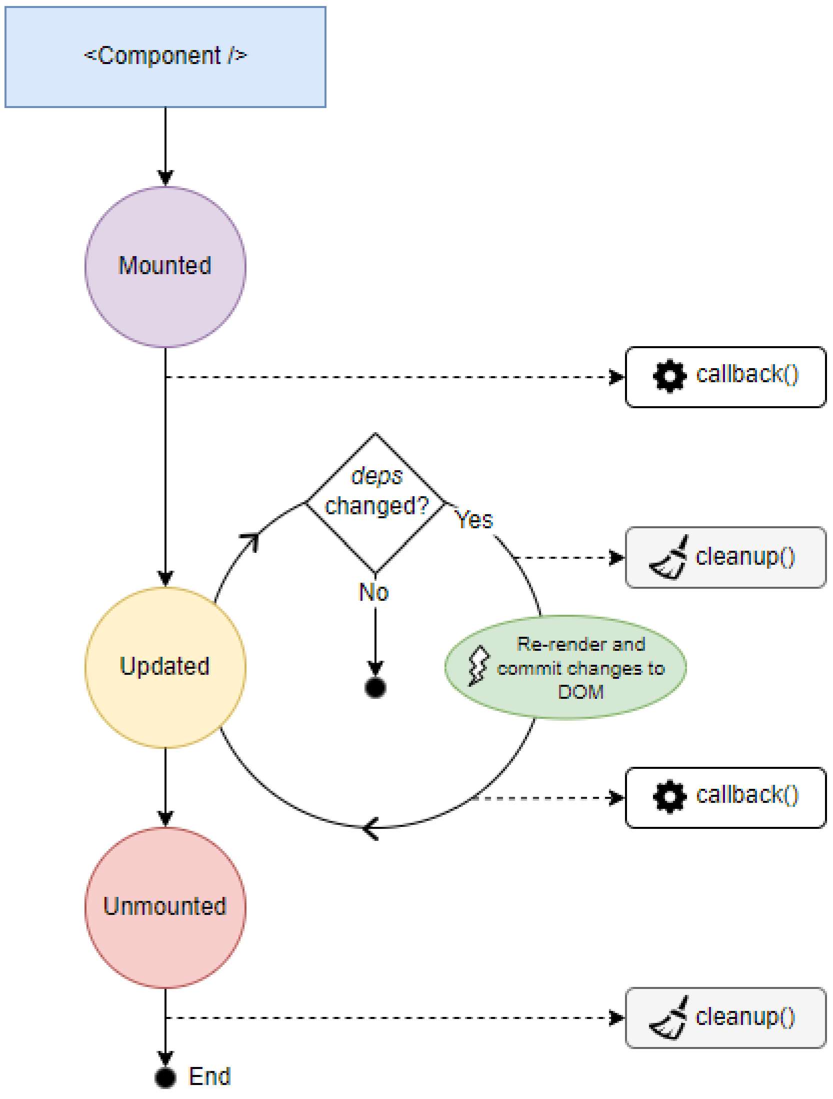


Figure 7: useEffect() Hook (Dmitri P., 2022)

2.6.1.3 useRef

useRef returns a mutable ref object whose current property is initialized to the passed argument (initialValue). The returned object will persist for the full lifetime of the component. Essentially, useRef is like a “box” that can hold a mutable value in its current property. (Introducing Hooks, 2022) The hook accepts one argument as initial value and return a reference to it.

One popular use case of useRef hook is to access the DOM element. By passing the value to *ref* attribute of the element, we can keep track of the element easily (for example implementing scrollbars). Another case is to store the value during the component lifecycle yet unlike state, ref value does not make the component re-render.

2.6.1.4 useMemo and useCallback

The reason I put these hooks together and in last is that they share many commons and as React official documents, should only be used for the performance optimization, not for logic, which means the code should work fine whether these hooks are removed or not. In the future, React might even change its logic and remove these hooks completely (Introducing Hooks, 2022)

useMemo and useCallback structures are pretty the same with useEffect hooks. Both accept callback function as the logic and depend on the dependencies that tells the function whether it should be recreated or not. While useEffect controls the side-effect and run the logic without return anything, useMemo and useCallback have the return after its being called and memorized that return for later rendering.

The main difference between these two hooks is return type. While useMemo returns the memorized value itself, useCallback only returns the callback function. They are useful when the component includes some expensive values which require times to process and React is smart enough to memorize these values for later calls.

2.6.2 Redux

Redux is the state management library that helps React in details and any other frontend libraries in general control and organize its state values. We can think of it as a store where all the states of the application are put in order and can be taken and changed values in the most convenient way. In React applications, if we do not have the proper state management method, it is guaranteed that we will face trouble at some points. The best example is when we want to pass data, or props, from parent component to its grand grandchildren, we must pass manually every single level down until it comes to the desired destination. If the situation comes to ten level down, our code will be much longer, and most of them is repeated. Consequently, bugs will come up and it is difficult to keep track due to the hierarchy is deep, not mentioning to the performance affects (all the components on the path need to re-render).

In fact, with some small to medium applications in which the interface does not require many complex components, Redux brings more harm than good due to its complicated set-up steps. Implementing Redux will most probably extend the application code base twice. In these cases, it is recommended to utilize another state management methods – one of the most popular is useContext hook. Since the hook is not used in the annotation tool and out of the scope, we will not go into details of this hook.

However, when the application grows big and requires advanced state management, Redux is the best solution for developers. It contains the well-established structure which keep state storing and updating easily. Redux also offers certain optimization and improvement for performance compared to context API. Furthermore, it has amazing plugin which help the developing and debugging process easier by showing states in tree structure.

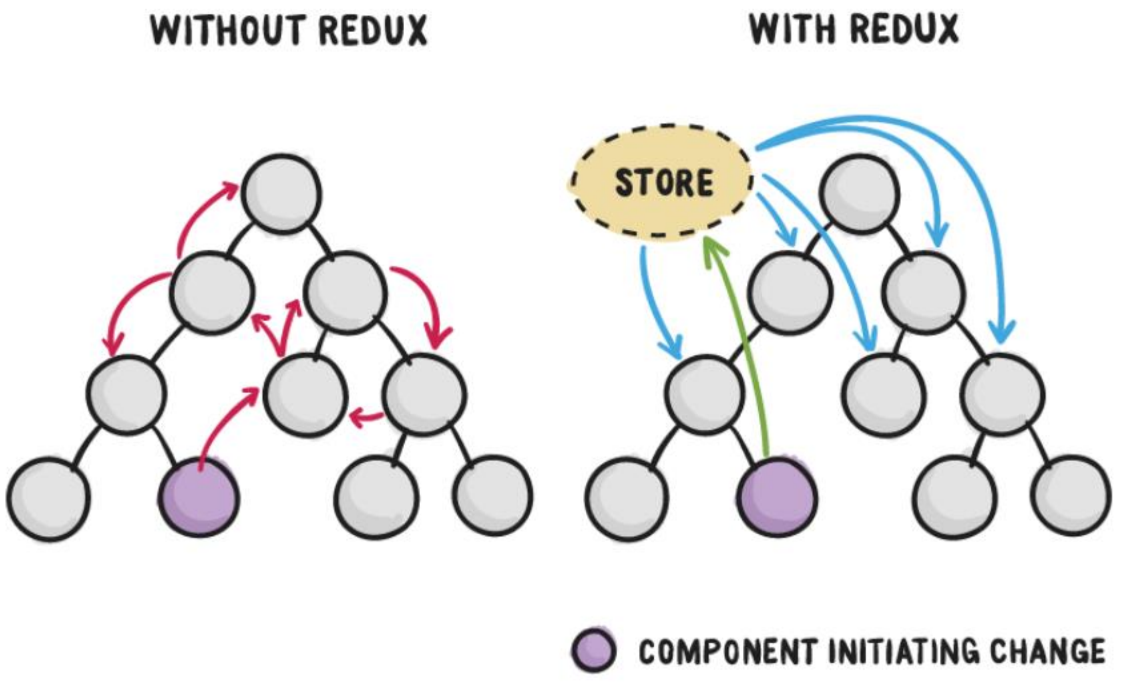


Figure 8: Inner workings of Redux (Federico K., 2017)

There are three core components in Redux – store, reducer, and action. Firstly, store is a place where the application states are kept and distributed to components whenever needed. Components can get state by accessing directly to the store, but they cannot update or modify it, but instead calling the action from the action creator functions. Action is normal object including data that are used to change the state and type of the action, so that Redux knows which part of the store should be updated and which part is not. All the “states updating process” are handled in reducers. The reducers will update the old state based on the actions received and return the new state.

In some cases, there is another concept called middleware in Redux event cycle. If we recall the middleware from Express section, it is “a guy” staying and doing extra actions in the middle – in Redux case, it is after action is called and before passing to reducer. This is the place where Redux do side effects actions, such as sending API requests, or log important information. Redux has supported library called redux-thunk to support these middleware actions.

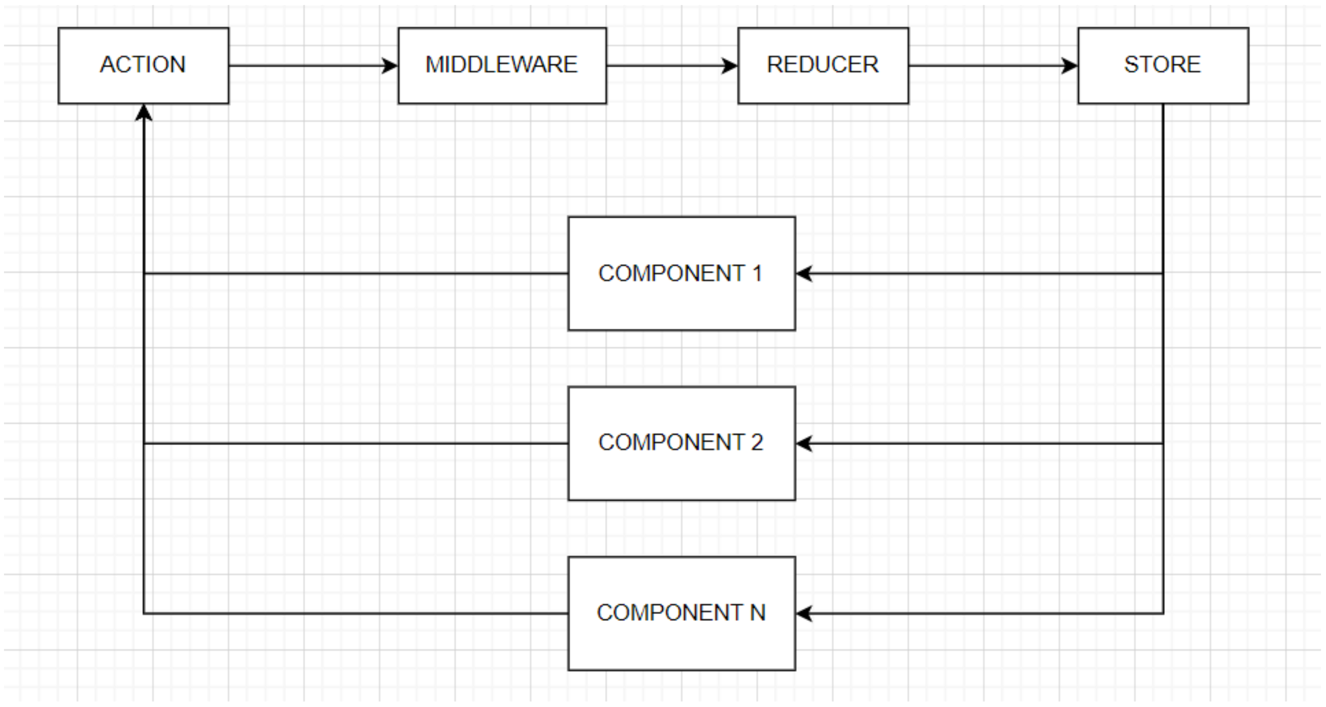


Figure 9: React-Redux operations

2.6.3 Canvas

If consider annotation process in the simplest terms, it is no more than drawing. We draw the polygons for objects, or the key points for human poses detection. In web development world, canvas is the main technique to implement the drawing graphics. Canvas is an HTML element which can be used to draw graphics via scripting (usually JavaScript). This can, for instance, be used to draw graphs, combine photos, or create simple animations. (Canvas API, 2022)

With canvas JavaScript library, we can easily draw two-dimensions graphics using script. For instance, calling “fillRect” function of Canvas with the starting point coordinators as well as the width and height of the desired output, we can have the rectangle graphic nicely, or fill its color with the “fillStyle” method.

Canvas is so powerful since it is not only support drawing basic two-dimensions graphics but also three-dimensions or animations. Canvas can even render images from binary data – data whose unit can take only two values, 0 or 1. The explanation of binary images are out of the scope and will not be included in this report.

3 The empirical part

Being aware of the full potential of AI, many companies nowadays have concentrated on implementing AI solutions to improve life of people around the world, and Verso Vision is one of the leaders of the field in Finland.

Verso Vision is a software company which provides the solution to prevent patients' falls in the hospitals and healthcare wards. First introduced in 2017, the company's product has been used widely not only in Finland but also in other countries, including Estonia, Australia, and more. Verso's team has developed a software solution that give nurses another "eyes" to take care of the patients, especially the sensitive ones who can get hurt easily by falling like elders. All the streams from cameras are handled and analyzed by AI model, and after going through lots of algorithms, it decides if the patients are going to fall soon so that can give the responsible nurse the right notification. In this way, nurse can prevent the fall even before it happens, and since the whole process is taken by AI without any people watching, the patients' privacy is kept safe and respected.

Since the solutions depend heavily on AI, it is important that we have a good amount of data, both quantity and quality, to fetch the models for the trainings (the concept will be described in detail in the next chapter). Moreover, the data are valuable that we might need to use them again in the future whenever a new model is introduced, they need to be organized and managed carefully.

With the high demand, the annotation tool is developed to help all these processes become easier and more convenient for the team. The tool will have two main features: annotation (data drawing and labelling), and data management (statistics, access controls, and data flows). It is developed based on Make-Sense, a free-to-use online tool for labeling photos but is added a lot of new features to do the annotation easily and closely related to the company's needs. (makesense.ai, 2022) It is worth to note that Make-Sense is one hundred percent frontend application written by React, while the developed annotation tool has the proper backend and databases for data management handling. Alongside with the essential features, the tool needs to work as much dynamically as possible. It means that by defining in the configuration files, the tool can display and work with the 2 new features without changing the code significantly.

The final products will be evaluated and used as production by the Verso's team, mostly within the AI development team. Feedbacks and recommendations for the future improvements will be given once the team has enough experiment with the software. However, the first version of the software is expected to do the annotation easily with the data output in Protobuf format. It should perform well with multiple users using at the same time – dispatching data from file server, access control flow, data management, etc.

3.1 Design and Starting Point

The application is built with 3 main data databases: Network File Systems (for frames that are annotated), Mongo (for saving data annotation), and Redis (for controlling users sessions and dispatched images). Mongo and Redis are set up as containers so the team can easily install them if needed for local tests. Besides, the tests pipelines can be run properly every time the code are updated since they know how to build the databases from scratch. The containers and pipelines scripts are out of scope, so they are not described in detail in this report. NFS, in fact, is not a database but more like a data storage, yet its usage is to store the data which are essential for the tool, it will be described as part of the databases.

First, NFS stores the directories in which list of images are included. They can be stored in another databases like Mongo in the binary format, but it never be a good practice to save them due to the huge amount of data. Storing files are usually handled by storages like Amazon S3. However, since they are people images and might violate GDPR regulations, NFS is the best place for this information.

Secondly is MongoDB which is also the main database for the application. There are two collections for the tool: Data Annotation and User. As the name described, the former stores the data annotation which include metadata and data itself, and the later saves the user information of the application. The following describe the Mongo and data models which are used in the tool, and because Mongo is NoSQL database, the relationship becomes more flexible.

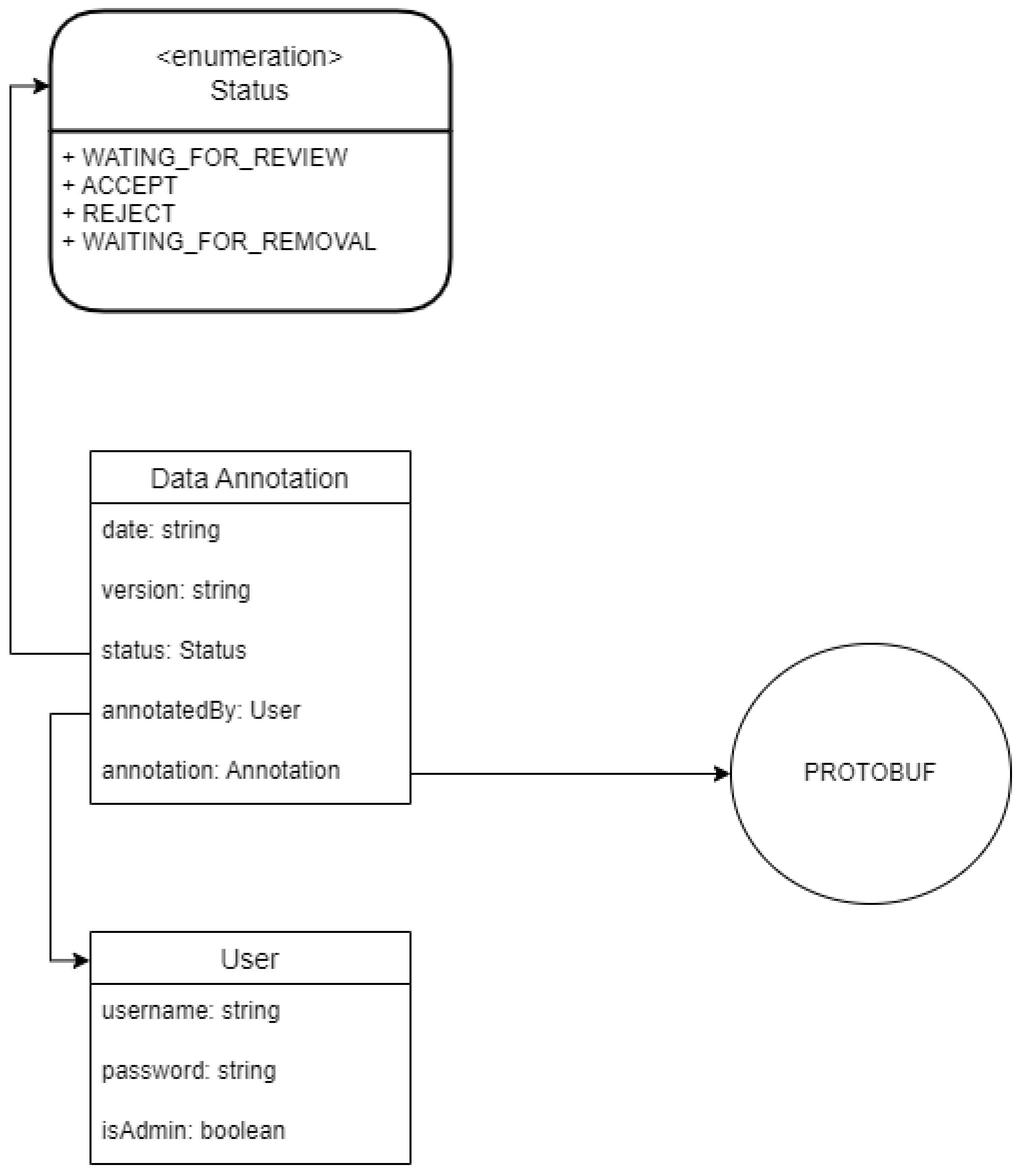
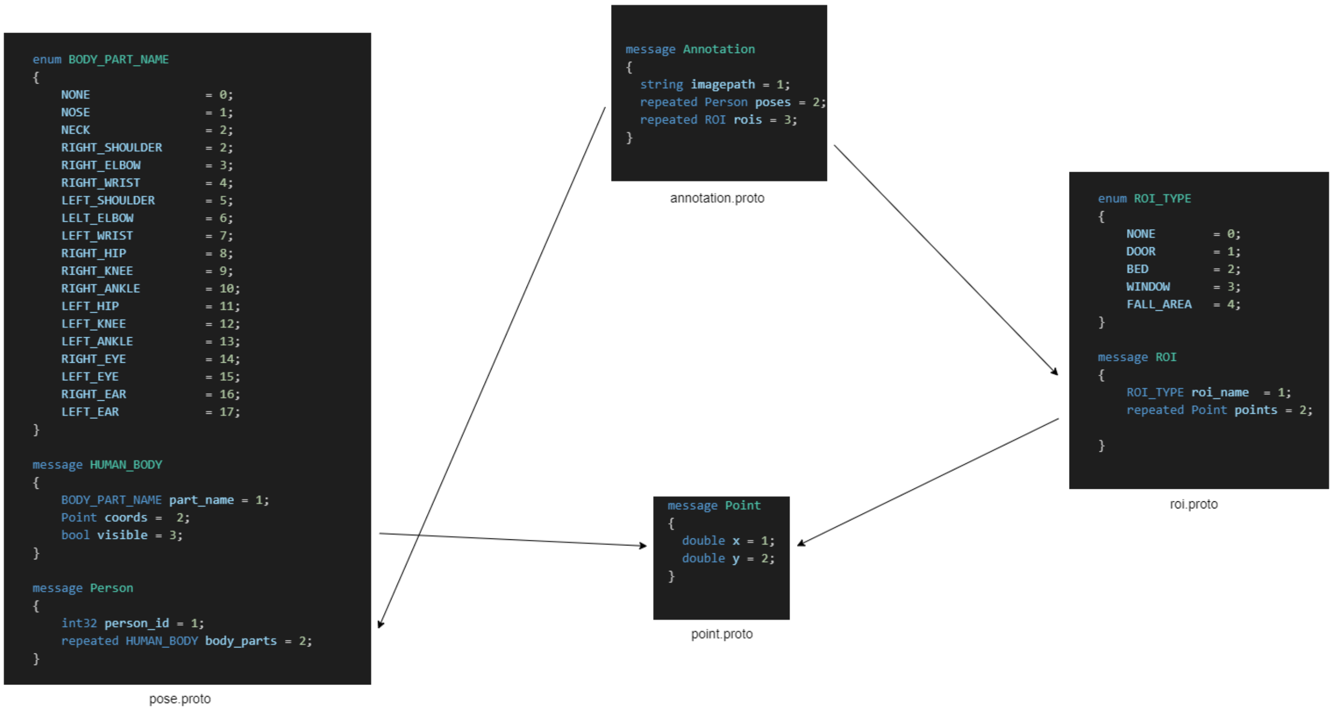


Figure 10: Mongo models relationship

For the Data Annotation collection, the metadata contains the information that describes the data: the time annotation happens, user who annotated, status of the annotation (since the data are extremely important and requires a high accuracy, they are going to be review, accept, or reject by admin account before exported for the models training sessions), and the version (we probably update the Protobuf depends on the use case and the trained model, so it is compulsory to keep track it).

The data part comes from the annotation process, which are drawing and labelling images. It contains the image path of the frame in the disk – relative path from the predefined root working path in the configuration – which combining with the version in metadata, created the unique key of the document in MongoDB (unique key is used to identify one and only one document in MongoDB). The root path in configuration file will be explained in depth in the implementation chapter. The remaining parts of the data are described in the Protobuf files.



```
enum BODY_PART_NAME
{
    NONE                = 0;
    NOSE                = 1;
    NECK                = 2;
    RIGHT_SHOULDER      = 2;
    RIGHT_ELBOW         = 3;
    RIGHT_WRIST         = 4;
    LEFT_SHOULDER       = 5;
    LELT_ELBOW          = 6;
    LEFT_WRIST          = 7;
    RIGHT_HIP           = 8;
    RIGHT_KNEE          = 9;
    RIGHT_ANKLE         = 10;
    LEFT_HIP            = 11;
    LEFT_KNEE           = 12;
    LEFT_ANKLE          = 13;
    RIGHT_EYE           = 14;
    LEFT_EYE            = 15;
    RIGHT_EAR           = 16;
    LEFT_EAR            = 17;
}

message HUMAN_BODY
{
    BODY_PART_NAME part_name = 1;
    Point coords =  2;
    bool visible = 3;
}

message Person
{
    int32 person_id = 1;
    repeated HUMAN_BODY body_parts = 2;
}
```

pose.proto

```
message Annotation
{
  string imagepath = 1;
  repeated Person poses = 2;
  repeated ROI rois = 3;
}
```

annotation.proto

```
enum ROI_TYPE
{
    NONE        = 0;
    DOOR        = 1;
    BED         = 2;
    WINDOW      = 3;
    FALL_AREA   = 4;
}

message ROI
{
    ROI_TYPE roi_name  = 1;
    repeated Point points = 2;

}
```

roi.proto

```
message Point
{
  double x = 1;
  double y = 2;
}
```

point.proto

Figure 11: Data Annotation in in Protobuf format

The messages are divided into different files by category and can be imported into each other, creating a clear data structure and being reusable whenever new Protobuf message is added. Besides common primitive types like double (for decimal numbers), string (for text), bool (for Boolean values), I defined a couple of custom fields for the annotation purpose, such as *Point* type for two-dimensions coordinators, or *ROI* type for any ROI annotation. The keyword *repeated* means there can be multiple items of that field, commonly known as *Array.* On top of that, enumeration is represented for the set of predefined collections.

The annotation field in the final Mongo Document comes from the Protobuf, while the status contains 4 values which were predefined in the enumeration: WAITING_FOR_REVIEW (first time annotated images, requires some checks from another users), ACCEPT (good annotations, ready to be delivered for the trainings), REJECT (bad annotations, need to be fixed and updated), and WAITING_FOR_REMOVAL (images which are too dark and poor quality, unable to annotate accurately and need to be deleted).

For the User document, it has the simple requirements like username and password for the authentication purposes. Besides, there is another field named *isAdmin* to grant the authority for specific users to do the “dangerous” job, such as export finished data to the disk or remove the annotation from database. Since the application is used only internally within the company, the security and authentication are not the main points and implemented as simple as possible.

The final database utilized in the tool is Redis. In contrast to the mentioned database and storage, Redis does not participate in the data annotation part but more likely to control the working. Since we do not want users to handle all the files selection each time they start the tool, which is wasted of time and reduced the effectiveness, the tool itself needs to handle all the dispatching images process – give user the very first free images in the working path that were not annotated or reserved by any other users. This is extremely important because we all use the tool with the same dataset from the NFS, the first situation we want to avoid is multiple users annotate in the same image.

Since the key will be used to valid if the specific image is available for the annotation, it needs to be unique. Therefore, the combination of username and image path are selected as the key for a Redis item. By using this combination, the key can be guaranteed unique and self-descriptive: it tells user and image for which the item reserved for. Along with key, the item has the default value “active”. Each item has expired time which was predefined in the configuration and will be renew each time user interacts with the server.

For the user interface part, it is mostly inspired by Make-Sense with added pages for the login, images dispatching, and review area. The reason is that Make-Sense provides attractive user interface with the nice drawing functionalities. Moreover, it is written completely by React with TypeScript, so the implementation can continue smoothly. By utilizing Make-Sense as the base for the front-end part, I can focus more on the server logic and annotation management which are the core of our needs. Because Make-Sense has a complicated structure and logics, it is important to study the code base to understand how the application works before adding new features.

3.2 Implementation

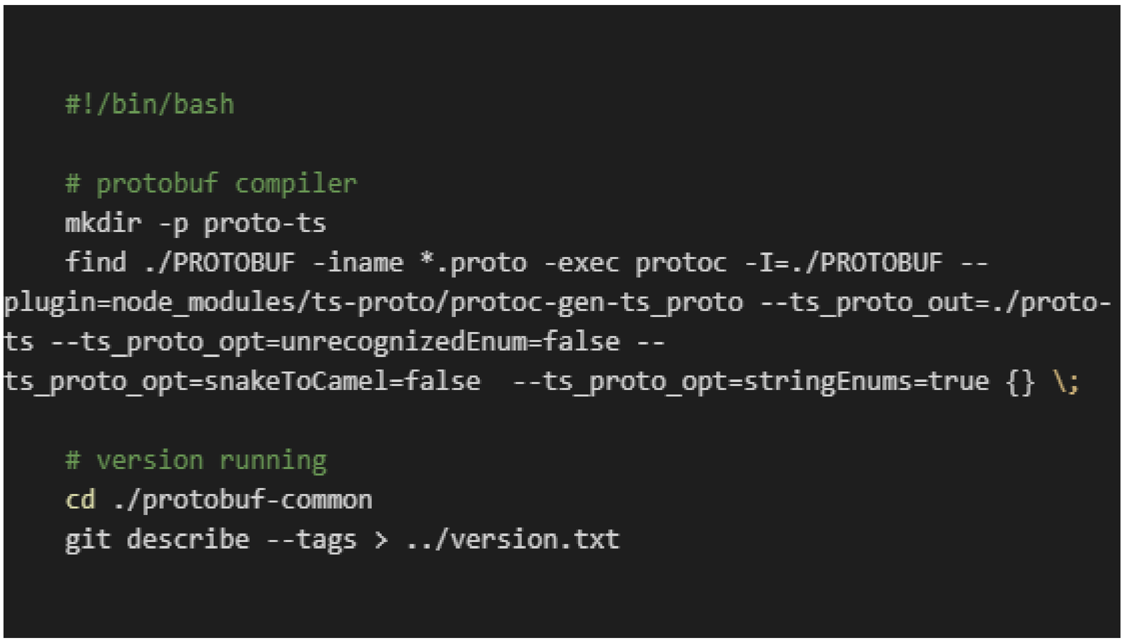
3.2.1 Setup the working environment

3.2.1.1 Protobuf

The following steps are applied in both the frontend and backend repository because they need the data annotation in Protobuf format so the data structure can be consistent between the client and server.

First, Protobuf repository needs to be added as a submodule into the frontend and backend, and after checking out in an expected version of Protobuf that works with the tool, Git Submodule recursive command is executed to clone the Protobuf repository into the project.

After figuring out the Protobuf data structures, they should be compiled to the current programming language to be utilized effectively, even though Protobuf alone can be implemented manually (Protocol Buffers, 2022). Thanks to TS-proto, the compiling process to TypeScript in quite straightforward. The script was written inside a script file so anytime there are updated in the Protobuf, I can easily get the new TypeScript types by executing the file. The script file is placed in the root of the directory.



```
    #!/bin/bash

    # protobuf compiler
    mkdir -p proto-ts
    find ./PROTOBUF -iname *.proto -exec protoc -I=./PROTOBUF --
plugin=node_modules/ts-proto/protoc-gen-ts_proto --ts_proto_out=./proto-
ts --ts_proto_opt=unrecognizedEnum=false --
ts_proto_opt=snakeToCamel=false  --ts_proto_opt=stringEnums=true {} \;

    # version running
    cd ./protobuf-common
    git describe --tags > ../version.txt
```

Figure 12: Protobuf compiler script

The file script runs 3 jobs whenever it is executed. First, it creates a directory named proto-ts which will be used to store TypeScript files – the *p* flag enables the command to create only if the directory is not existed. Then, the second job takes care of the compiling process. It checks all proto files in the PROTOBUF folder and then executes the Protobuf compiler, which is protoc, to convert the protos files into TypeScript files. The extra flags format the TypeScript to better work with the tool. The last part keeps track of the Protobuf version by using Git tag command. The output is saved into text file so every time new annotation is added, the version will be updated based on the version defined in the text file.

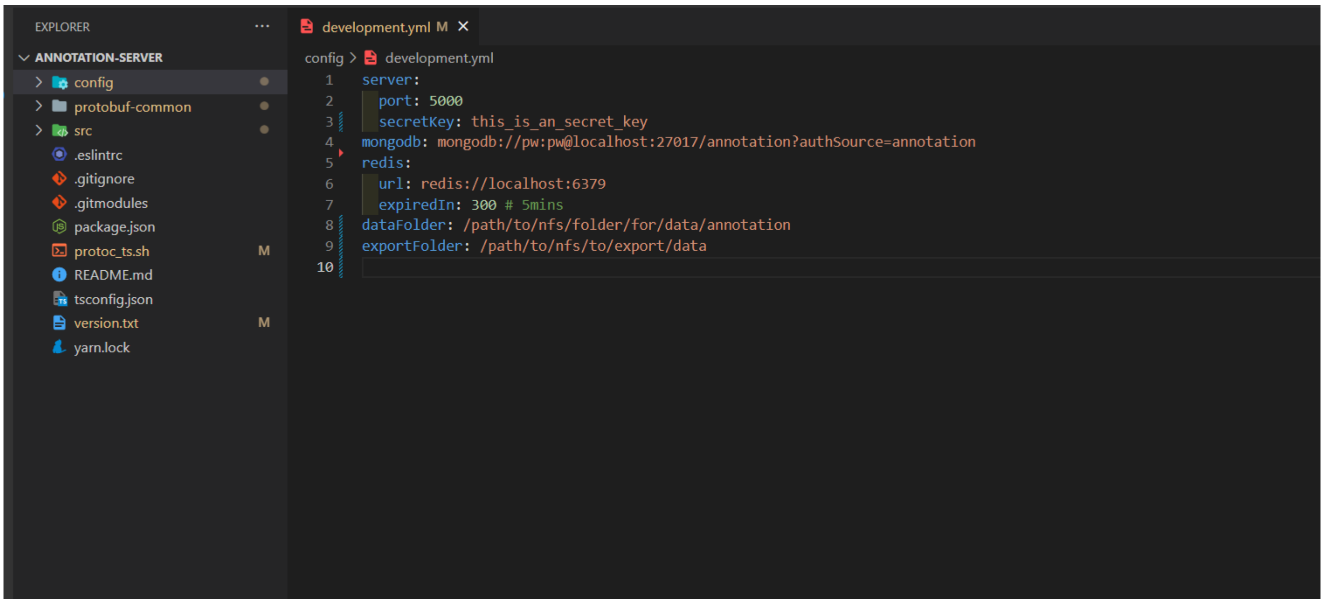
3.2.1.2 Backend Server

The server was built following the common steps of setting up a Nodejs server with Express: creating "package.json" files then installing essential dependencies, adding eslint for code format, and since the project is written in TypeScript, defining "tsconfig.json" file needed to be done properly. (Express, 2022)

The next step was to define the configuration files. These files are significant since it not only defines the necessary environment variables but also “tell” front-end what features should be included and how. Commonly “.env” files are used for these purposes, but I decided to use YML file since it is more human readable and easier to be configured.

In the configuration file, it is important to define the working path, which determine in which directory the tool will read the files. For instance, if the predefined path is *root/path/set-1/child-1*, the tool will only work with the dataset inside the *child-1* directory. On the other hand, if the predefined path is *root/path/set-1*, the tool will work with all the sub directories inside the *set-1,* which includes *child-1, child-2, etc.* And of course, if the path is left empty, the tool will loop through all the directories in the local, which is not recommended due to the huge amount of data. By defining the root working path in advance, we have more controls of how the dataset is prepared and organized for the team just by changing the configuration.

As a standard Express server, the “routes” and “controllers” folders were created to handle the REST API logic of the application. Besides, because of using Mongo, the data models needed to be defined beforehand. All the logic codes are written in the *src* folder at the root of the directory.



```yaml
server:
  port: 5000
  secretKey: this_is_an_secret_key
mongodb: mongodb://pw:pw@localhost:27017/annotation?authSource=annotation
redis:
  url: redis://localhost:6379
  expiredIn: 300 # 5mins
dataFolder: /path/to/nfs/folder/for/data/annotation
exportFolder: /path/to/nfs/to/export/data
```

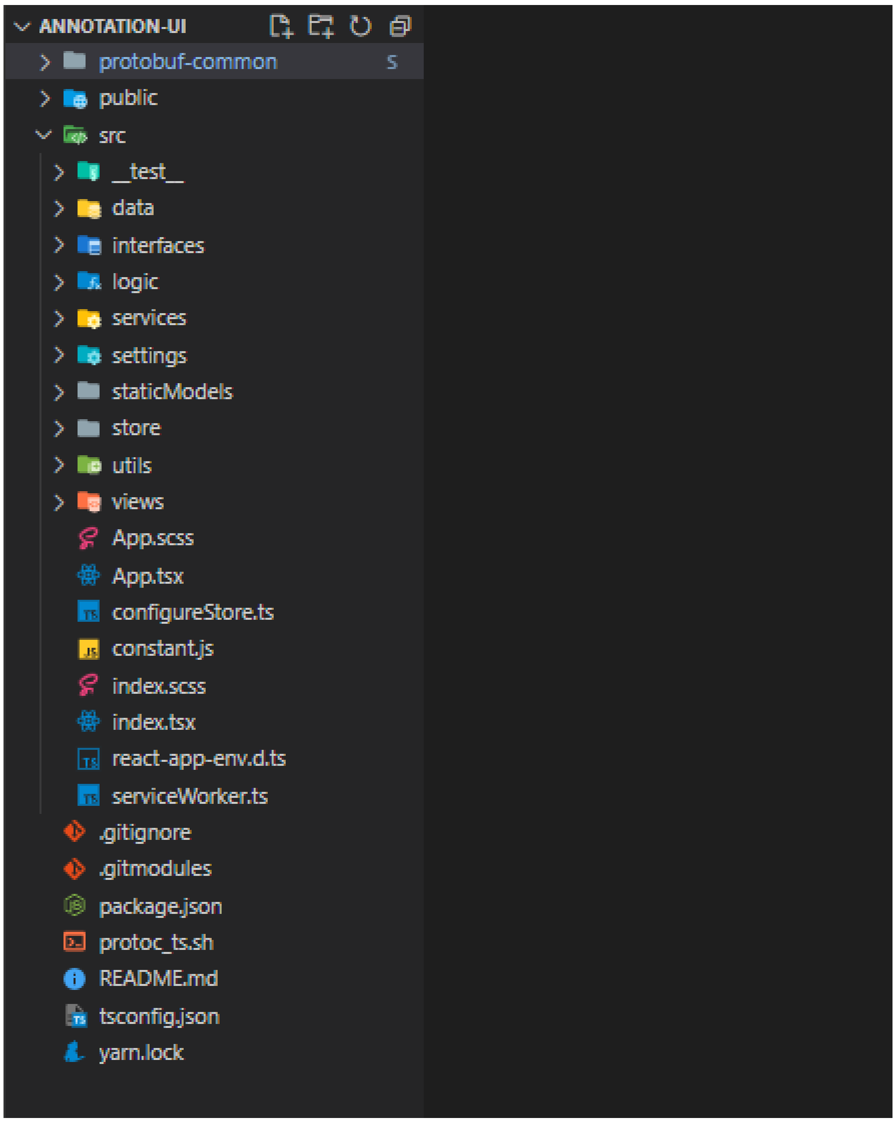
Figure 13: Backend structure with the configuration example

3.2.1.3 Database

The databases were installed and run based on Docker images. Containers are powerful, being used widely nowadays because they are simple, easy to configure and install in few minutes, in contrast to local installations. Moreover, it avoids the situation “it works on my machine but not yours” which causes headache for developers before the containers were invented. However, since container and Docker are out of the scope, they will not be included in the report.

3.2.14 Frontend

The front-end part is developed based on Make-Sense, so the setup phase is quite straightforward. The repository was forked first from the URL https://github.com/SkalskiP/make-sense.git so that the development can separate from the origin project. Then, it was cloned and cleaned by removing unnecessary features which are not needed in the annotation and management tool.



```
ANNOTATION-UI
  > protobuf-common                 5
  > public
  v src
    > __test__
    > data
    > interfaces
    > logic
    > services
    > settings
    > staticModels
    > store
    > utils
    > views
      App.scss
      App.tsx
      configureStore.ts
      constant.js
      index.scss
      index.tsx
      react-app-env.d.ts
      serviceWorker.ts
    .gitignore
    .gitmodules
    package.json
    protoc_ts.sh
    README.md
    tsconfig.json
    yarn.lock
```

Figure 14: Frontend structure

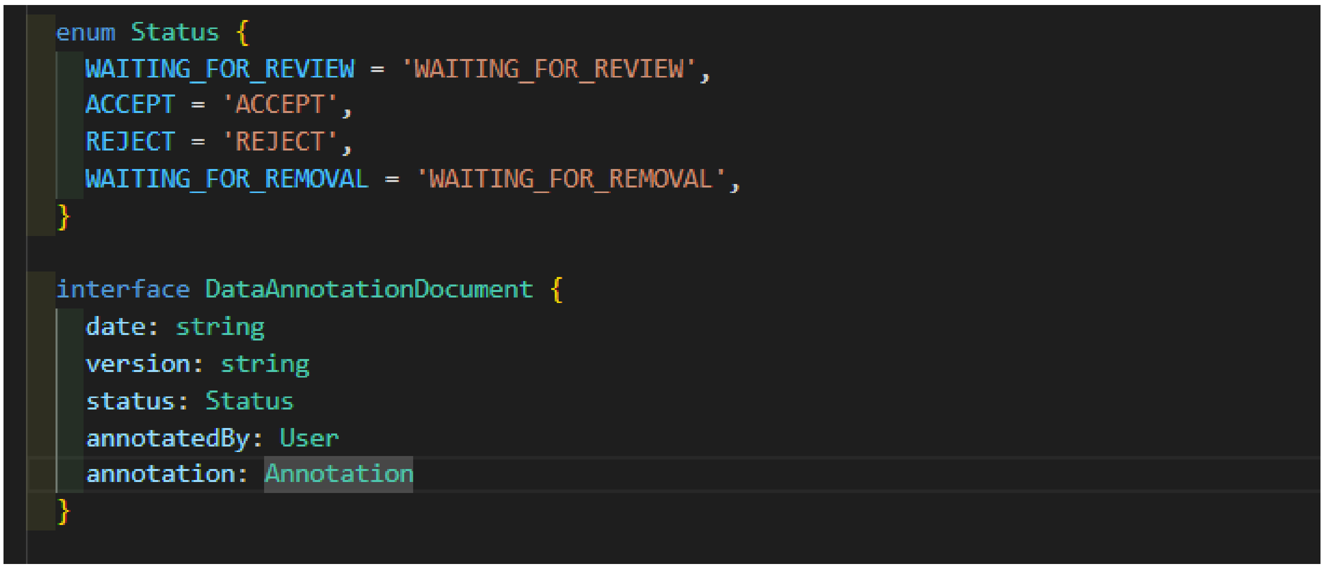
The front-end is structured based on the standard React application, dividing into different folders based on its purpose: “views” for UI components, “store” for Redux state management, services for communicating with backend, and more.

3.2.2 Backend

As any other Nodejs application, the first step was to write *index.ts* file as an entry of the application. Because this is the very first running steps of the server, all the connections to databases and routes must be defined in this file. The database connections, which are Mongo and Redis, were added in the beginning, followed by the built-in middleware like *CORS* and *static.* Besides, the custom middleware was used to improve the usability of the application, and they will be clarified in the next chapter.

3.2.2.1 Define Mongo Document for Data Model

After finishing the base *index* file, next step was to generate the Mongo models. There are two models need to define in the annotation and management tool: Data Annotation and User. While the User model is straightforward with *username, password, and isAdmin* in primitive types, the Data Annotation document requires custom type in some fields, which are *status and annotation.* Thanks to Protobuf compiler, the later becomes simple. With the compiling TypeScript files which were created in the setup environment phase, all I had to do was importing the needed interface, which is *Annotation,* and the complicated data structure was available to be used. The remaining task was to define the enumeration Status for *status* field.



```
enum Status {
  WAITING_FOR_REVIEW = 'WAITING_FOR_REVIEW',
  ACCEPT = 'ACCEPT',
  REJECT = 'REJECT',
  WAITING_FOR_REMOVAL = 'WAITING_FOR_REMOVAL',
}

interface DataAnnotationDocument {
  date: string
  version: string
  status: Status
  annotatedBy: User
  annotation: Annotation
}
```

Figure 15: Data Annotation interface

Next, I needed to convert them into Mongo objects. These steps were completed with mongoose library. By initializing mongo schema and modeling it based on the types above, I could use all the

Mongo queries inside Nodejs environment conveniently.

3.2.2.2 Middleware

The two middleware I implemented in the tool are *passport* for authentication and *errors* for error handling.

The passport middleware was put before each route, except the login because simply, user needs to login to get the credentials, not the other way around. The *passport-JWT* library creates a strategy which checks user credentials every time he or she wants to use services in the server - when logging in the system, user is granted a token which will be used for authentication. With the secret key defined in the configuration, the username is extracted from the token and checked to database. If existed, the request is passed to next route; otherwise, it returns error message back to client regarding authentication issue.

Because every request is sent through this middleware, it is probably the best place to review the user's session for the reserved images.

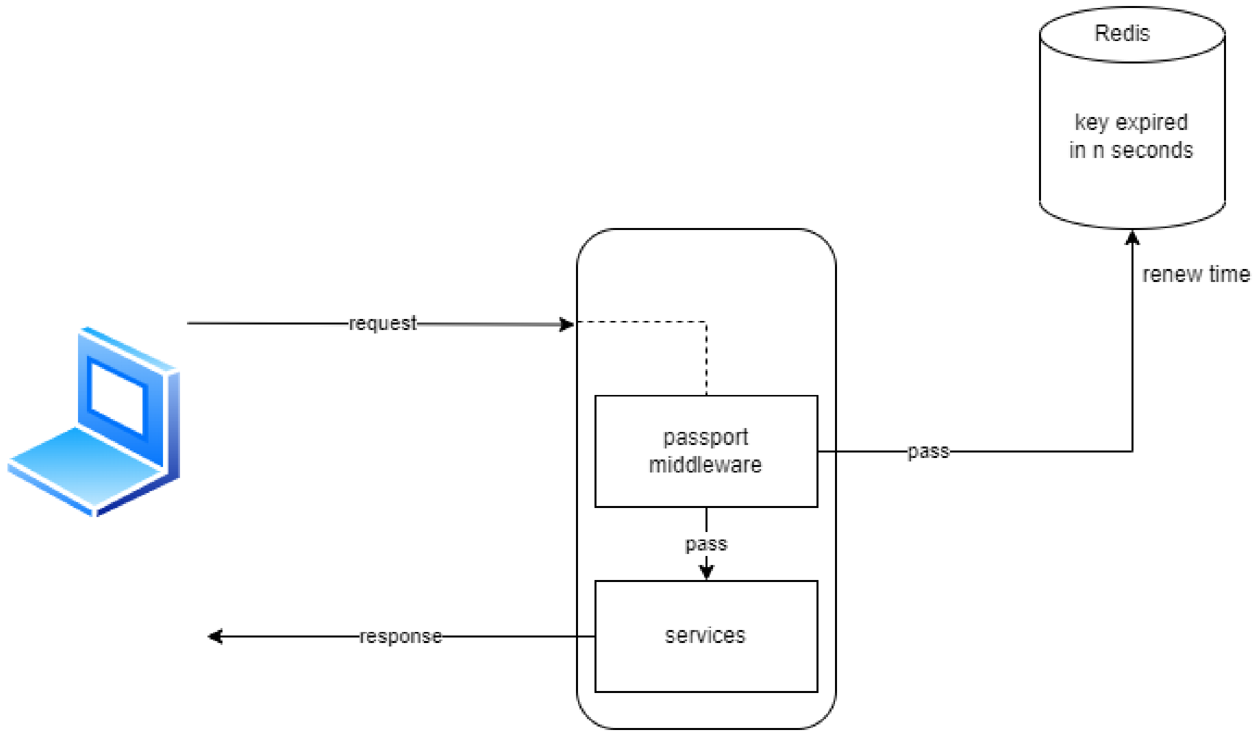


Figure 16: Renew session by passport middleware

For the error's middleware, I create a route to handle the situation when an API is called to the server with the incorrect endpoints. By putting it to the end of the routes' calls (the orders are matter because Express executes route from the first to the bottom and the first route that fit the endpoint will be used), it can be certain that whenever a call has wrong endpoint, the middleware is executed, and proper error message will be sent back to the client.

3.2.2.3 Building REST API

The routes for the APIs were implemented inside the routes folder and all the services logic (with databases and file systems) were split into the controllers folders, so it can be maintained later easier.

The first logic needed to be done was to dispatch several images to different users without knowing the actual path to the folder(s) – user might or might not give the full path to the specific folder, and if he or she leaves it empty, the tool needs to loop through all the folders in the local machine. It is not problematic since the maximum dispatched images for a user is set static, so the moment the tool finds the number of required images, it will stop reading further. Besides, the given images must not be duplicated – an image sent to user A cannot be sent to user B.

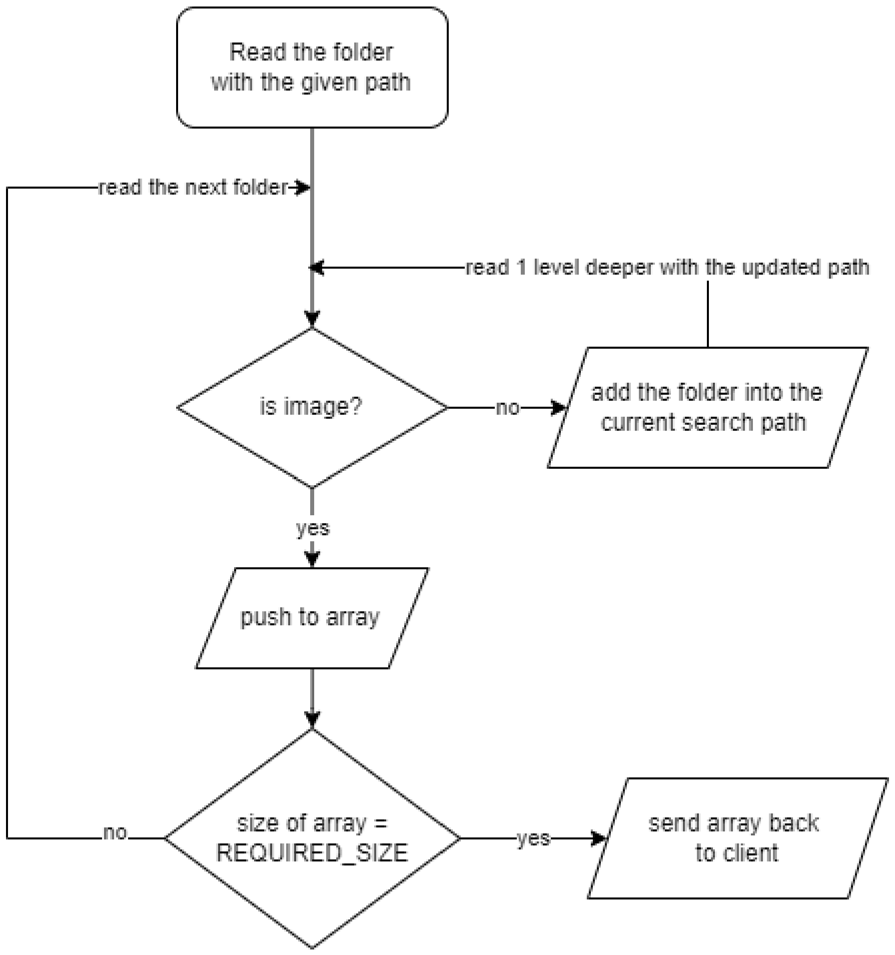

Read the folder with the given path
read the next folder
read 1 level deeper with the updated path
is image?
no
add the folder into the current search path
yes
push to array
size of array = REQUIRED_SIZE
no
yes
send array back to client

Figure 17: The process of reading images

When the server receives request with the sub path, it reads all the items in the absolute path, which is combination of the root path from the configuration and the given sub path. If the item is another folder, it adds the folder name into the current path and reads again until it finds image. The next step is to make sure the given images are unique for every user.

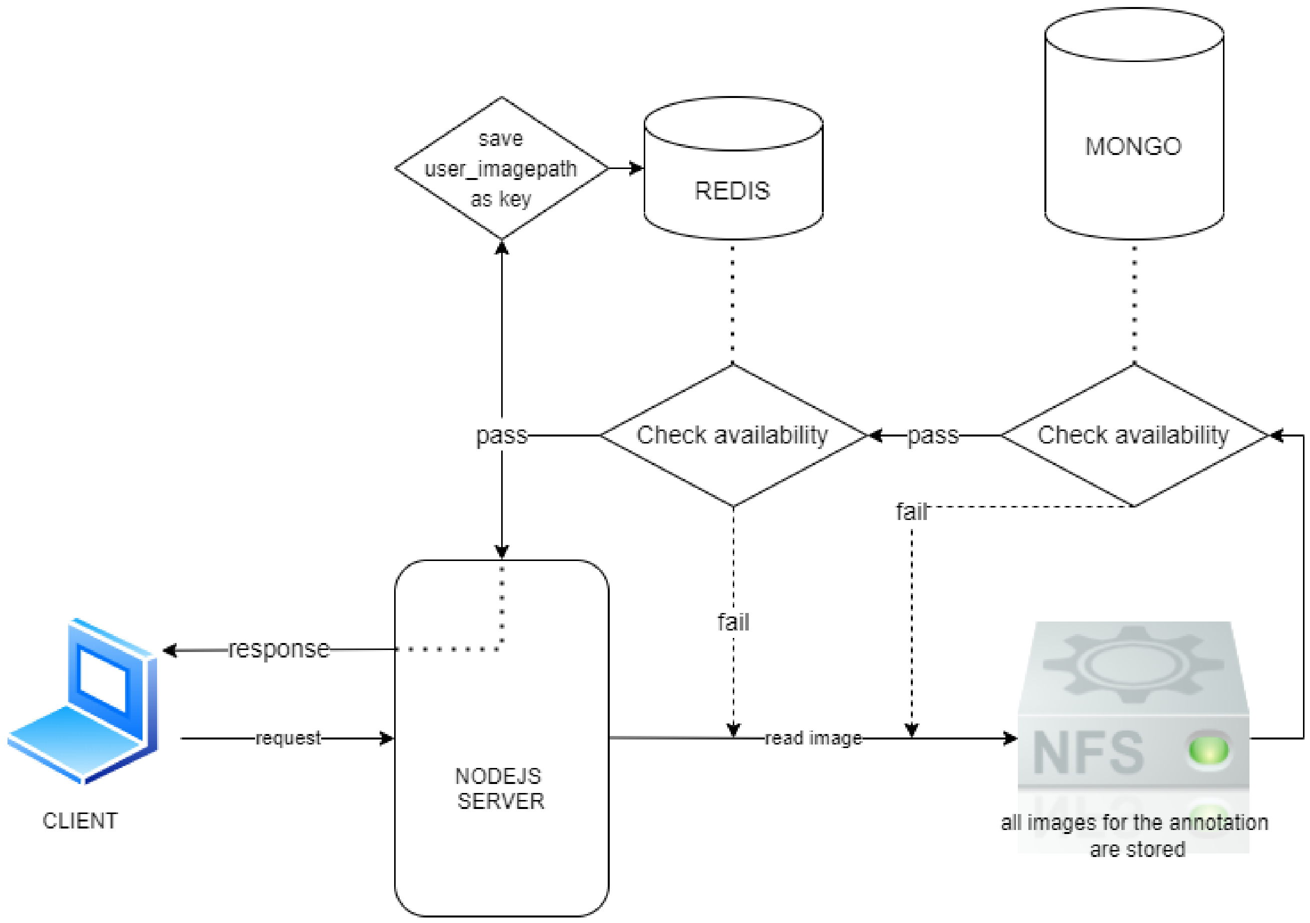


Figure 18: Full dispatching process in the server

Server first reads an image in NFS with the step described before, and after having an image and its path, server runs two checks to decide if the image is available for annotating or not: checks with MongoDB if the image has been annotated before, and then with Redis if it was reserved by other users. If either of the checks fails, Node server loops back to the read image step and repeats the checks. Until both checks pass, the image is ready to be sent to the browser. The process is repeated until it reaches the number of requested images from the user.

It is important to note that the image is not sent in the binary format, or the image itself, but in the shape of its relative path from the root working directory. Express has built-in *static* middleware which allows Node server host and serves the images, so that the browser can display the images in the runtime

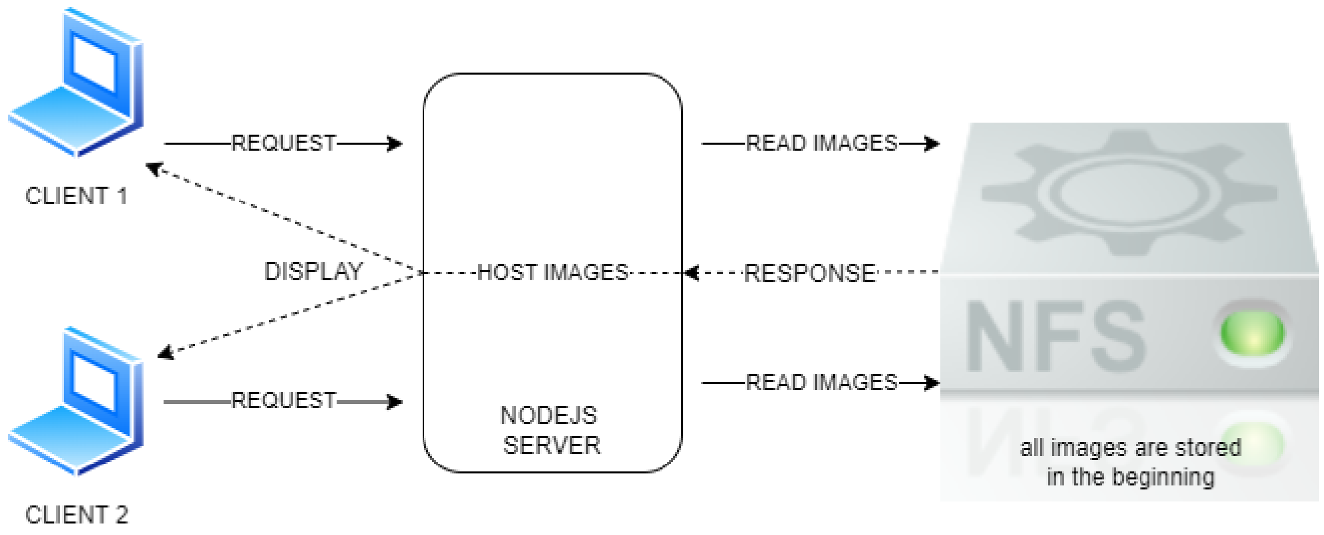


Figure 19: Nodejs Server working with NFS

After finishing the dispatch process and giving users the correct dataset, the next steps were implementing the database services. Users annotate data and do common interactions with the database through the REST API which was built by the server. The common tasks include Create, Read, Update, and Delete, commonly known as CRUD as the four essential operations with database.

The queries were written with the help of mongoose library and the Mongo models created in the previous steps. Particularly, for saving data to Mongo, there is an extra step to validate the data with Protobuf built-in method – it check all the fields of the data whether they follow the types defined in proto files. If the field obeys correctly, the data is kept as before; otherwise, the wrong field is updated to empty value. By doing the validation it can be certain that the saved data in Mongo is in correct format.

When the image is annotated in the first time, its annotation data is saved into Mongo in WAITING_FOR_REVIEW state. Then, user starts reviewing the annotated data, updating its status based on the quality and accuracy of the annotation. Finally, the image's annotation data is processed according to its status. The operations are described in detail in the following UML diagram.

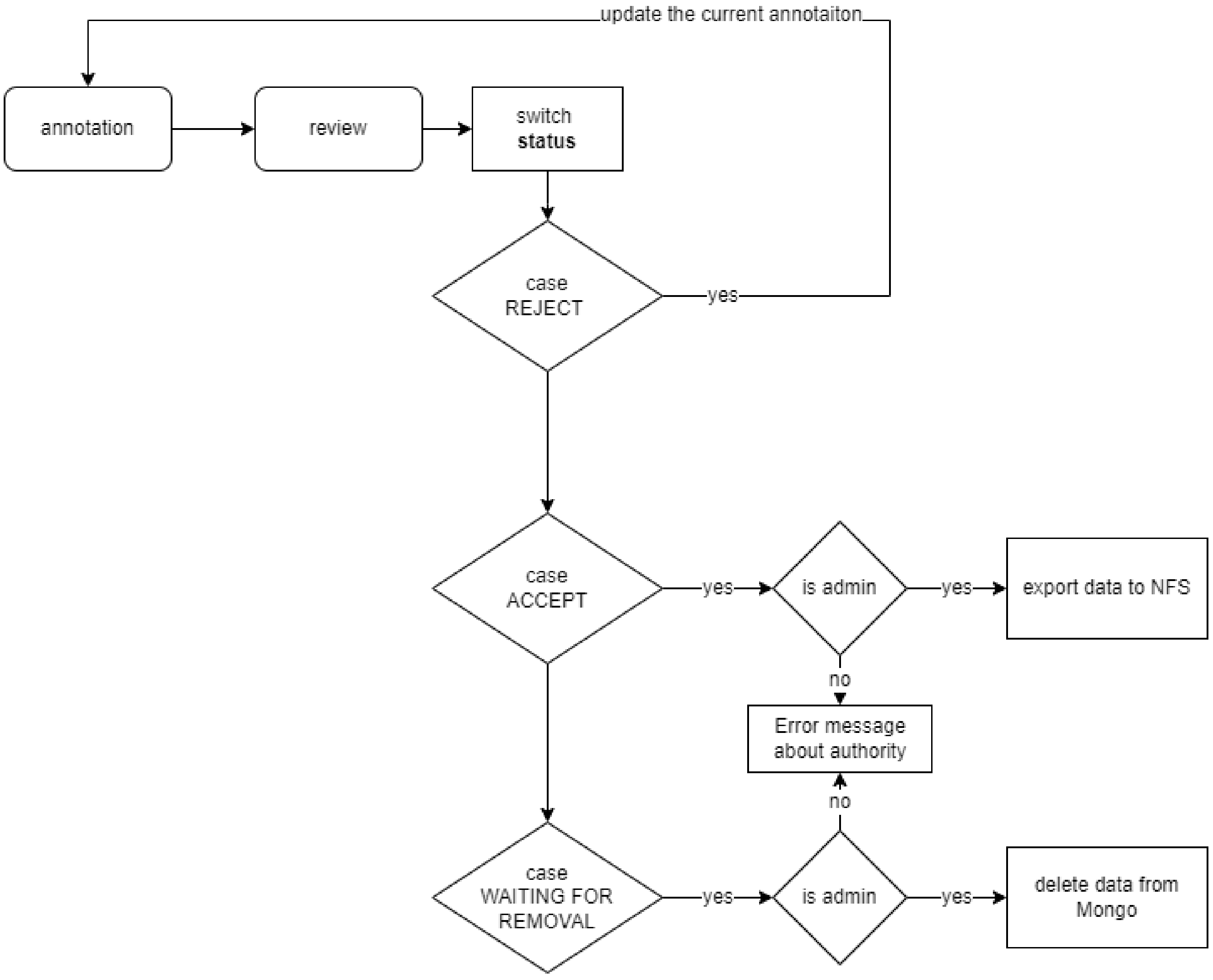


Figure 20: Review and Data Management processes

3.2.3 Frontend

Since the project uses TypeScript, the types must be defined carefully before any actual implementation started. All the data and functions type were declared into separate files, and the data annotation type, just as Mongo models preparation step, can be achieved by importing the compiled Protobuf types.

3.2.3.1 Redux State Management

After having types, the development was started by rebuilding Redux store to fit our needs. The application state is divided and categorized into different folders based on its purposes. Each one has its own reducer and action creator implementations, so the store has the nice structure, helping the maintenance later.

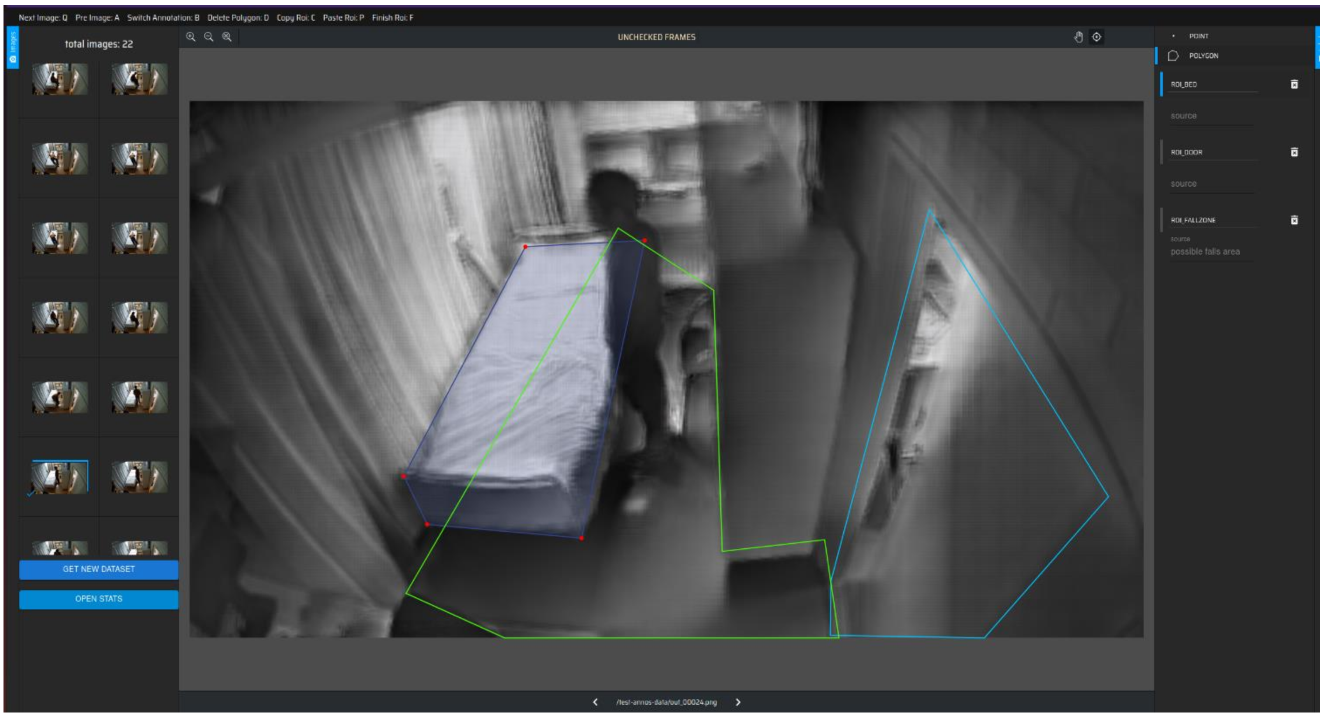
In the action creators, I added few redux-thunk functions for supporting asynchronous actions like fetching APIs so the state logic can be centralized within Redux store. With the *dispatch* parameter in redux-thunk return function, the state can be updated directly inside the actions file.

By providing the store from the root of the application, which is *index.tsx,* all the components in the application can access the state by calling the useSelector hook and update it by useDispatch.

All the API calls were added in the services folder using axios library – it supports making http requests in the most convenient way. Especially, by calling *interceptors* method, I can attach the token as an authorization header before each request are sent.

3.2.3.2 Drawing and Labeling Images

Basically, the tool has two main engines for labeling images, which are point and polygon – user can change the engine by clicking the corresponding icon or using the keyboard button predefined backend configuration. In the polygon engine, user can do the ROI annotation by drawing and labeling objects with the Protobuf enumeration labels. ROI can be updated, as well as deleted, and the values are saved into Redux state so the data can be available in all components.



ROI_BED

ROI_DOOR

ROI_FALLZONE

source: possible falls area

Figure 21: ROI annotation

Like polygon engine, the point engine runs with the set of labels called from Protobuf enumeration, and the value is stored in Redux. This engine used for POSE annotation, and because it defines human body parts, the rule is more complicated than the ROI annotation. Each point needs to be drew at the exact position of its part in the human body, if existed one in the frame. Human POSE is completed only if it has enough key points identified in the frame, which are 16. There might be multiple persons in one frame, so it is important to finish one person POSE before moving to the next. Besides, some of the key points are invisible in the frame, and they need to mark correctly so the models can get the most accurate data.

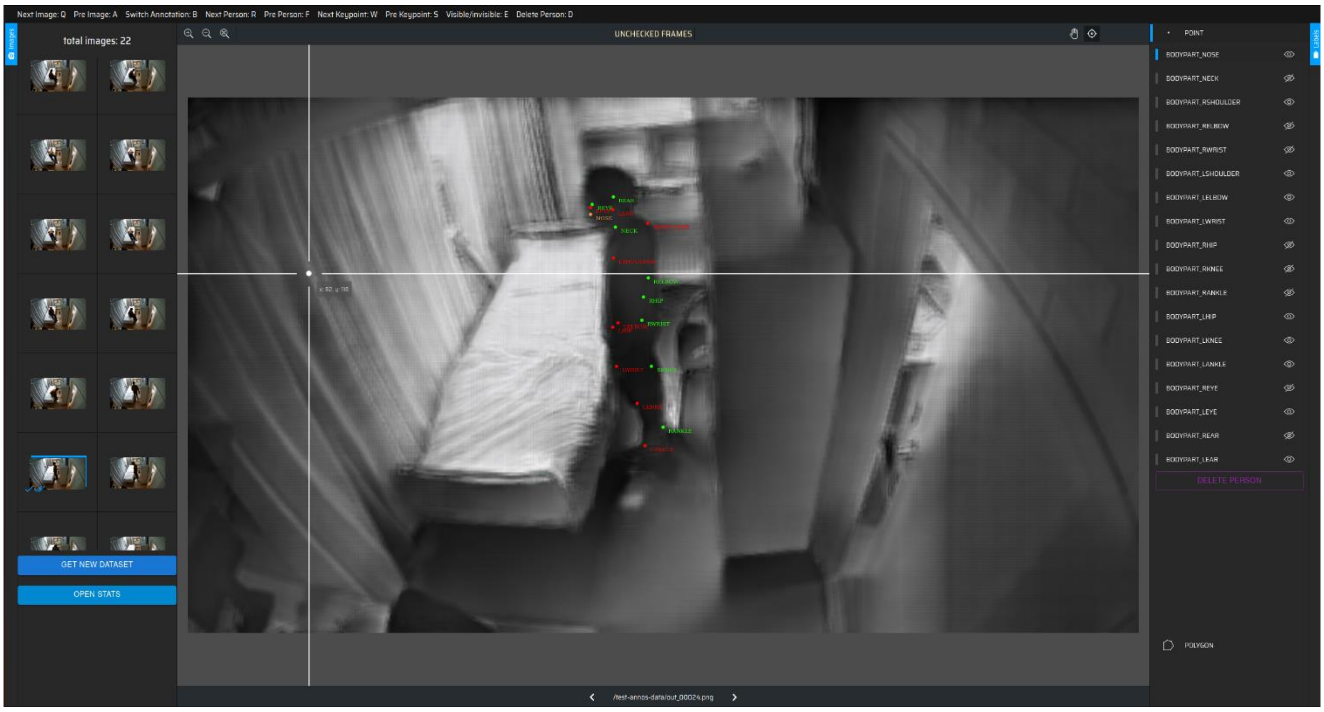


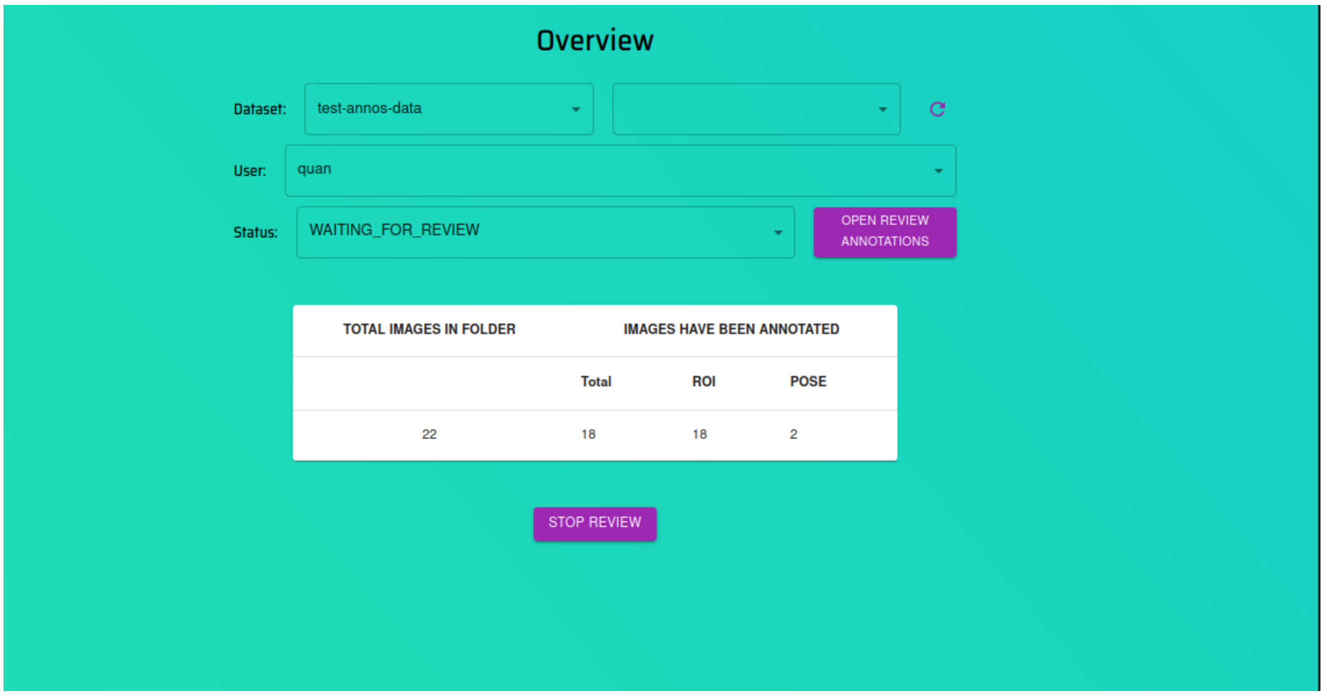
Figure 22: POSE annotation

The ticks in the gallery in the left navigation tells users whether the frame has been annotated or not, giving an overview of how many works have left before user getting the new dataset.

These works were done with the heavily used of Canvas library as well as the most popular event listeners in the browser, which are mouseDown, mouseUp, keyDown, etc. The coordinators were calculated carefully so the data from drawing can be corrected and trusted – the coordinators were figured as ratio from the image's dimensions instead of the proper pixel, so I can make sure the same result is kept even though the frame is stretched or shrink due to the resolution.

3.2.3.3 Review Data Annotation

After annotating the images, user needs to go through the data again to make sure they are checked carefully before using to train models. User can review his or her own annotations or the others, by accessing the statistics area. The view displays the data annotation information by number, how many images total in the selected working path, how many has been annotated, how many annotations of user A, and more.



Overview

Dataset: test-annos-data

User: quan

Status: WAITING_FOR_REVIEW

OPEN REVIEW ANNOTATIONS

TOTAL IMAGES IN FOLDER	IMAGES HAVE BEEN ANNOTATED		
	Total	ROI	POSE
22	18	18	2

STOP REVIEW

Figure 23: Statistics of current annotation

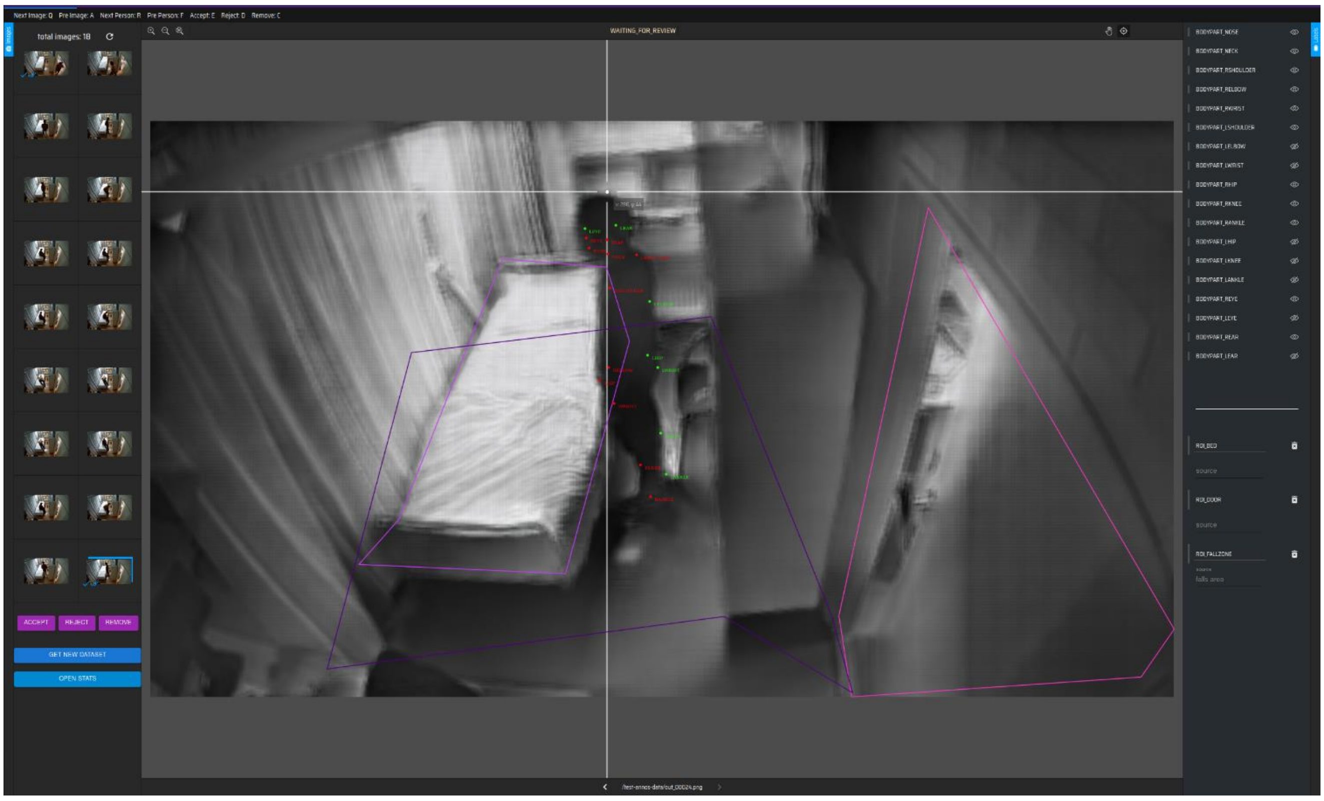


Figure 24: Review data annotation

All the annotations are displayed simultaneously, helping users process faster and spot incorrect annotation more accurately. Just like dispatching free images, the review images were implemented by the same logic so one WAITING_FOR_REVIEW image can be handled by one user at a time. He or she can accept, reject, or remove the annotation so the data is handled properly – recheck the figure 13 for more information.

3.2.3.3 View Components

Most of the components were created with the support of MUI library, one of the best libraries providing ready-to-use React components for the user interface. By passing the needed props into the component, I can achieve a nice looking interface without taking the of the styling. Some of the MUI components which were used heavily in the tool are Input and Button for common tasks, Select for form selection, Dialog for popup notifications, and more.

3.2.3.4 Built-in keyboard event

The tool was equipped with plenty of keyboard shortcuts to improve the user experience and productivity. By saving the pressed key and adding the listener into useEffect hook, I can catch the key and perform specific task depend on what was assigned to that key in the configuration file. By doing this, the selection of keys are not restricted to specific set but dynamically based on user's preference, which will be defined in the configuration. Some of the most frequent key event tasks are changing images, switching the engine, delete the annotation, etc.

There are many minor implementations to fulfill the application features. For instance, there is a *check-status* API sending to the server every few seconds by setInterval method to inspect if the user session is still valid. The server checks the time remaining of reserved images for user and response the proper message. If the time is less than specific number of seconds, the user will receive a warning message with the remaining time. If that user does not renew the session by clicking the option in popup window, he or she will be kick out and all the unsaved reserved images are free to use for other users.

4 Discussion

4.1.1 Result of the outcome

After having several personal tests and getting supports from my supervisors and colleges, the tool was ready to use in production within the company. I documented, configured, and deployed the application to the internal server with by Docker, so my colleges can access the tool from their own computers with the given credentials which were predefined when I set up the databases.

The product was evaluated by four main factors: necessity, usability, strength and weakness, and further processing possibilities. Luckily, I had a chance to have a short interview with Lauri Laaksonen, a lead scientist at Verso Vision. Lauri was my first user and after the first few days experiment with the tool, he had the following comments regarding the annotation and data management tool:

- Necessity: The tool offers an integration of two annotation types that were previously marked using two specialized tools, producing two separates annotation files without any combined view. The tool is expected greatly improve the annotation, data management and annotation parsing processes.
- Usability: Excluding some bugs related to session expiration and occasional data retrieval failures, the tool is very usable. Both the high and low-level workflows are logical, and most relevant and commonly used features have keyboard shortcuts to improve the productivity.
- Strength and weakness: The whole annotation process can be done in the tool, from image pre-filtering to annotation and review, and the tool is relatively dynamic with respect to the changes in the Protobuf messages. The main drawback is a bug that can cause data loss if the user is not careful.
- Further processing possibilities: The tool could be extended and integrated to data versioning tools, making it capable of fully handling the process of both annotation and data management.

Lauri was able to go through about 11270 of annotations in a couple of days with the new tool, and the accepted data was exported and delivered to the AI team for training models. We faced few bugs regarding the session expiration and data fetching failures sometimes when user accesses review mode. The session problem was fixed, yet it probably needs more trials among multiple users to say if it works perfectly.

The retrieval error, unfortunately, is not totally figured out yet. My first guess was because of calling asynchronous action inside a loop of hundreds of items, the fetching process is not handled well. Occasionally, it loses the data of few images in the list and if user is not careful, he or she might lose the data annotation of that image as well. My solution was instead of resolving promises inside the loop, I added all promises directly into an array and resolved all of them at once. After few tests, I no longer faced the error and the performance improved significantly – even though I had to loop two times, it is still faster than loop one time but need to await every single asynchronous action. More tests need to be done to validate the solution and if the retrieval failures still occur, I need to do more research about the phenomenon to understand what really causes the error.

4.1.2 Personal development

After processing thesis, I have learnt and developed considerably myself to become a full stack developer. Because of building an application to work with valuable data and real users, I had to consider seriously the usability, performance, and precision of the tool. Moreover, the frontend part was developed based on Make-Sense, so I had an opportunity to learn how to structure a large-scale React application by various experience developers, allowing the app runs smoothly even though there are lots of rendering happens.

TypeScript and Protobuf give me a lesson of how important of typing is when developing a software application. Especially Protobuf, it not only manages data typing but also opens a new door for another type of protocol messages – gRPC, which I believe might get over traditional http in the near future.

4.1.3 Further development

The first version of the data annotation and management tool is expected to handle most of the current annotations within the company. Based on more feedbacks given from my colleges, some minor updates or possibly new features need to be added to help the annotation process more convenient.

Furthermore, since we probably introduce more AI models in the future which requires different type of annotation, the tool needs to work as dynamically as possible, which means by updating the Protobuf messages and the configuration files, the tool should perform well with the new features without major changes in the code. The current tool runs dynamically when there are some minor changes in the Protobuf, but when a big update like adding new custom message which has complex structure comes in, the code needs to update significantly.

Another room for development is to build tests for the whole application, from unit tests to integration tests and finally, automation tests. Moreover, the whole pipeline for the application – test, build, deploy – is needed so that more automated processes are processed instead of human manual ones.

5 Source

Artificial Intelligence for Dummies - Julianna D., SME, IBM Analytics, Data Science/Machine Learning, 2021
URL: https://ebookcentral-proquest-com.ezproxy.haaga-helia.fi/lib/haaga/reader.action?docID=5325074
Accessed 22 August 2022.

What is data annotation and why does it matter? – TELUS International, 2021
URL: https://www.telusinternational.com/articles/what-is-data-annotation
Accessed 22 August 2022.

How do you train artificial intelligence?
URL: https://www.telusinternational.com/articles/how-to-train-ai
Accessed 22 August 2022.

Makesense.ai
URL: https://github.com/SkalskiP/make-sense
Accessed 15 September 2022.

Supervised vs. Unsupervised Learning: What's the Difference?
URL: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning
Accessed 8 October 2022.

A Comprehensive Guide to Human Pose Estimation – Nilesh B., 2022
URL: https://www.v7labs.com/blog/human-pose-estimation-guide
Accessed 10 October 2022

Git
URL: https://git-scm.com/
Accessed 2022

Atlassian
URL: https://www.atlassian.com/git/tutorials/using-branches/git-merge
Accessed 2022

About JavaScript

URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript#what_is_javascript

Accessed 2022

TypeScript for the New Programmer

URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

Accessed 2022

Why You Should Choose TypeScript Over JavaScript - Gints D. & Olga B., 2020

URL: https://serokell.io/blog/why-typescript

Accessed 2022

Protocol Buffers

URL: https://developers.google.com/protocol-buffers

Accessed 2022

What is JSON?

URL: https://www.w3schools.com/whatis/whatis_json.asp

Accessed 2022

Protobuf vs JSON

URL: https://www.educba.com/protobuf-vs-json/

Accessed 2022

Enumeration

URL: https://en.wikipedia.org/wiki/Enumeration

Accessed 2022

Why Use MongoDB and When to Use It?

URL: https://www.mongodb.com/why-use-mongodb

Accessed 2022

Mongo models

URL: https://mongoosejs.com/docs/models.html

Accessed 2022

Introduction to Redis

URL: https://redis.io/docs/about/
Accessed 2022

Redis — What and Why?
URL: https://medium.com/weekly-webtips/redis-what-and-why-pros-cons-ae2f5bc750fd
Accessed 2022

Network File System (NFS)
URL: https://www.techtarget.com/searchenterprisedesktop/definition/Network-File-System
Accessed 2022

Introduction to Node.js
URL: https://nodejs.dev/en/learn/
Accessed 2022

5 Reasons to Choose Node.js
URL: https://www.bitovi.com/blog/5-reasons-to-choose-nodejs
Accessed 2022

File system
URL: https://nodejs.org/api/fs.html#file-system
Accessed 2022

Express
URL: https://expressjs.com/
Accessed 2022

What is REST
URL: https://restfulapi.net/
Accessed 2022

Passport-JWT
URL: http://www.passportjs.org/packages/passport-jwt/
Accessed 2022

Ts-proto

URL: https://github.com/stephenh/ts-proto#overview

Accessed 2022

Introduction to the DOM

URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Accessed 2022

ReactJS – Quick Guide

URL: https://www.tutorialspoint.com/reactjs/reactjs_quick_guide.htm

Accessed 2022

React Component

URL: https://reactjs.org/docs/react-component.html

Accessed 2022

Introducing Hooks

URL: https://reactjs.org/docs/hooks-intro.html

Accessed 2022

React-life-cycle-diagram

URL: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

Accessed 2022

State Vs. Props

URL: https://www.javatpoint.com/react-state-vs-props

Accessed 2022

A Simple Explanation of React.useEffect() – Dmitri P., 2022

URL: https://dmitripavlutin.com/react-useeffect-explanation/

Accessed 2022

Getting Started with Redux

URL: https://redux.js.org/introduction/getting-started

Accessed 2022

A look at the inner workings of Redux – Federico K., 2017

URL: https://medium.com/@fknussel/redux-3cb5aac94a66

Accessed 2022

Canvas API

URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

Accessed 2022