1 Analysis

1.1 Introduction to problem

Currently, my family uses a whiteboard in the kitchen to track upcoming events. However, this can be inconvenient as this can only be seen and edited in person.

This project aims to solve this by creating a digital emulation of a physical whiteboard, along with adding other utility functions that are made possible by the digital medium. This will enable coordination and planning to be done remotely at any time, as opposed to requiring everybody to be in the same room.

For this project, I plan to only create a web client as this will be compatible with the most devices, but in the future I may go on to create native apps for different devices.

1.2 Interview with Client

What sort of functionality would you like the whiteboard to have?

> I think the following features are most important:
>
> - Being able to use it on a phone and a laptop
> - Adding to the whiteboard with both text boxes and some sort of pen tool
> - Adding in text boxes
> - Being able to move things around on the board and delete them
> - Having multiple people edit the same board in parallel so that everybody sees the same thing

What other functionality would be useful?

> - I would like to be able to cross off items somehow
> - It would be useful to be able to switch between multiple whiteboards
>   - Sending items between boards would be useful

What other tools or canvas items could the whiteboard have?

> Some sort of way to categorise or order information would be good
>
> > What about a system of "tags" which can be placed anywhere on a board, and a search tool that lets you list tags and jump to their location on the board?
> >
> > > Yes, please. That sounds really helpful.
>
> Inserting web links would be useful, including a thumbnail of the page the link points to.
>
> > I can do web links fine, but fetching a thumbnail would not be feasible within the project timescale
>
> I would like to be able to upload/drag+drop a file (such as a spreadsheet/word doc) and display it on the board
>
> > Parsing Office documents is not feasible, but images/text files could work
> >
> > > No problem. Would I be able to cut and paste information from a text document into a text box?
> > >
> > > > That wouldn't be a problem to implement I would like to be able to upload an image such as a screenshot That should be fine

What other features would you like to include?

> I think some sort of private board would be a good system
>
> > How about passcode protected boards
> >
> > > I think that would work well

1.3 Investigation

I decided to build a web app for my project as:

- The system can be used on a variety of devices without needing to build it separately for each one
- Synchronisation between devices already requires some form of networking so I would be using a web server anyway

For the frontend, I have therefore decided to use HTML5+CSS+TypeScript, as HTML+CSS are standard, and TS provides many advantages over plain JS.

For the backend, I will be using Rust with the Warp web server and Tokio runtime as I found these to work well together for other projects.

1.3.1 Communication

In previous projects I have used either a custom system or the `serde_json` library for serialisation. However, both of these have the issue that parsing and generating messages client-side is error-prone and/or requires repeating the structure of types used in messages. To avoid this, I found `ts-rs`, a Rust library that enables generation of TS type declarations from Rust types, which combined with `serde_json` should enable a seamless programming experience.

1.3.2 Rendering

In an earlier prototype I used the Canvas API to render the board, however this required manual rendering from basic shapes, and so I plan to use SVG to render the board.

1.4 Existing Solutions

1.4.1 Microsoft OneNote

I had been forced to use this previously in lessons and so I have some familiarity with it, though its many frustrations drove me away from it somewhat. When trialling it more thoroughly, I found that it had several issues that made it unsuitable for the problem.

Firstly, the software is divided into 5 different versions (Windows 10, iOS, Mac, Android, and Web), which despite accessing the same notebooks have varied feature support and reliability. This can make a task which is trivial on one device impossible on another.

Secondly, editing ability is limited at best. There is currently no way to rotate any object except for in increments of 90 degrees, and furthermore this functionality does not seem to be available on all platforms. Furthermore, this help page recommends using a separate application to edit photos beforehand. This is not only slow and inconvenient but cannot be applied to other objects such as text or "ink" drawings.

1.4.2 Miro

I had not used Miro before, but had heard of it from various friends and family. Upon trying it, the first thing I noticed was that the web version was somewhat slow to load, and occupied a significant amount of my laptop's resources. While my laptop is not particularly powerful, the system I aim to create should work on phones where this level of resource usage would be unacceptable. Miro does also provide a mobile app, though in this case the app is not feature-complete as it appears to be developed separately from the web version.

While Miro is a significant improvement over OneNote, it is not without some flaws, such as limited ability to work with custom shapes, along with selection of objects being clunky and confusing. For example, the rotation tool appears as a not immediately obvious icon in the bottom left corner of the selection. When releasing the mouse the selection box, and therefore rotation handle, is recomputed to fit the grid. This interrupts the flow of editing.

Another disadvantage is the lack of features in the free tier. While the most import parts are available, several utilities are limited or unavailable.

One notable feature that sets Miro apart is the existence of a separate "Interactive Display" mode targeted at static screens such as on a wall in an office. This makes it a much more powerful tool for group organisation as discussions and collaboration can be simultaneously done in person and remotely.

2 Objectives

0 Definitions

The following words (written in **bold**) are defined here:

0.1 Visual components

- **board** - the area of the program that shows an infinite scrollable and editable virtual whiteboard
- **panel** - an interactive floating window that provides a set of actions
  - **visibility button** - a button next to a **panel** that can show/hide it
  - **disabled** - a state in which a **panel**'s contents are not relevant and should not be displayed
- **item** - all canvas objects

0.2 Interaction and Editing

- **tool** - a configuration of **click** and **drag** actions
- **property** - an attribute of an **item** or **tool** which determines its behaviour
- **selection** - the **item(s)** currently being interacted with
- **transform box** - the border shown around currently **selected item(s)**
- **click** - An immediate pointer event in the **board** area
- **drag** - A sustained pointer event in the **board** area
- **multi-press action** - an interaction procedure that consists of multiple sequential **clicks**

0.3 Units/Types

- **distance** - unless specified, centimetres of screen space
- **point** - a 2-dimensional coordinate encoding **distance** to the right and up respectively from the **board** origin.

- **angle** - rotation in degrees clockwise from vertical, stored from -180 to 180
- **color** - RGBA hex color value (`#RRGGBBAA`)
- **list[T]** - An arbitrary length ordered collection of **T**
- **string** - a single line of characters, represented in UTF-8
- **text** - a multiline block of characters, represented in UTF-8

1 Layout and Interface

The program interface should consist of the following components:

1.1 Board

The full program area should be covered by the **board**

1.2 Panels

Panels should have the following properties:

1. A **visibility button** should be located adjacent to the **panel** which toggles its visibility
2. In certain contexts, a **panel**'s contents are not relevant and the **panel** should be **disabled**.
3. While a **panel** is **disabled**, it should be hidden regardless of its visibility, and its **visibility button** should be greyed out or hidden unless specified otherwise.
4. When the **panel** is no longer **disabled**, it should restore its previous visibility status

1.3 Toolbox

The program must display a "Toolbox" **panel** which consists of a grid of **tool** icons:

1. When clicked, the corresponding **tool** will be activated
2. The icon corresponding to the active **tool** must be indicated or otherwise highlighted
3. During an **action**, the toolbox should be **disabled** and its **visibility button** replaced with a button to cancel the **action**.

1.4 Properties

The program must display a "Properties" **panel** which contains a list of **properties** relevant to the currently **selected item**(s):

1. When a single **item** is **selected**, each of the **properties** of the **item** are listed and can be edited
2. The following **properties** should be omitted as they can be edited interactively with the **selection box**:
    1. Transform
    2. Start/End points for Line
    3. Points for Polygon
3. When multiple **item**s are **selected**, each of the following **properties** is listed if it applies to at least one of the **selected items**:
    - Stroke
    - Fill
4. When no **item**s are **selected**, the **properties** of the current **tool** are displayed, if any.
5. If there are no **properties** that can be displayed, the **panel** should be **disabled**

1.5 View controls

The program must display a "View" **panel** with the following buttons:

1. Zoom in
2. Reset zoom
3. Zoom out
4. Pan
5. Select Items

2 Items

A **board** may contain any number of **item**s, which are the basic visual pieces of the board

2.1 Common properties of items

The following **properties** apply to many items:

1. Transform, consisting of the following sub-**properties**:
    1. Position: the global position of the centre of the **item**'s bounding box as a **point**
    2. Rotation: the **angle** of an **item** relative to its centre
    3. Scale: the unit-less X and Y stretch of an **item**, *applied before rotations*
2. Stroke:
    1. Weight: a **distance** representing the width of the **item**'s border

    2. Color: the **color** of an item's border
3. Fill: the **color** which a shape is filled in with

2.2 Item types

The following types of **item** exist:

1. Basic shapes:
    - A simple **item** which draws one of the following shapes:
        1. Rectangle - A 1cm by 1cm filled rectangle
        2. Ellipse - A circle of radius 0.5cm
    - **Properties**: Transform, Stroke, Fill
2. Multi-point shapes:
    1. Line: a segment connecting two points
        - **Properties**: Stroke
            - Start: **point**
            - End: **point**
    2. Polygon: a closed loop of points:
        - **Properties**: Stroke, Fill
            - Vertices: **list[point]**
3. Path: a hand-drawn curve
    - Once completed, a path cannot be modified, but may be transformed
    - Unlike other **item**s, a path should be shown to other users while it is still being drawn
    - **Properties**: Transform, Stroke
4. Image: an image file loaded from a URL
    - The program should also provide clients with the means to upload images themselves
    - **Properties**: Transform
        - URL: **string**
            - The location from which to fetch the image
            - If the image is uploaded by the program, only a path should be included (e.g. `/upload/12345/image.png`)
        - Description: **string**
            - An optional description of the image, for screen readers
5. Text: a text box
    - The text should be able to have some basic formatting such as bold/italics
    - The text should be centred on its origin and each line should be centre-justified
    - **Properties**: Transform
        - Text: **text**
            - If the **item** is **deselected** while the property consists only of whitespace it should be deleted
6. Link: a hyperlink
    - Displays a clickable link which opens in a new tab
    - If no text is entered the URL should be displayed as a fallback
    - **Properties**: Transform
        - Text: **string**
        - URL: **string**

3 Tools

The following **tools** should be available:

3.1 Selection

1. Select
    - On **click**, **selects** the topmost **item** under the cursor if one is present
    - On **drag**, **selects** the **item** under the cursor and begins moving it
    - **Properties**: None
2. Multi-select
    - On **click**, **selects** all **items** under the cursor
    - On **drag**, **selects** all **items** fully contained within a rectangle between the beginning and end of the **drag**
    - **Properties**: None

3.2 Creation

1. Rectangle
    - On **click**, creates a rectangle with the default size centred at the cursor

- On **drag**, creates a rectangle with corners at the beginning and end of the **drag**
- **Properties**:
  - Scale: defaults to 1x1, determines the size of rectangles created by clicking
  - Stroke, Fill: Default values for all rectangles created

2. Ellipse
   - Creates an ellipse, functions identically to Rectangle except on **drag**, creates a circle centred on the beginning point and passing through the end point
   - **Properties**: Scale, Stroke (defaults)
3. Line
   - On **drag**, creates a line from the beginning to end of the drag
   - **Properties**:
     - Stroke (defaults)
4. Polygon
   - On **click**, creates a polygon **item** with one vertex at the cursor, and begins a **multi-press action** for adding points to the polygon
     - Successive **clicks** add further points
     - **Clicking** on the start point should close the polygon
     - Cancelling the action should **deselect** the **item** and not remove it
   - **Properties**: Stroke, Fill (defaults)
5. Path
   - On **drag**, draws a path **item**
   - **Properties**: Stroke (defaults)
6. Image
   - On **click**, inserts an Image **item**.
   - **Properties**
     - URL: **string** (upload box)
     - description: **string**
7. Text
   - On **click**, creates an empty Text **item** at the cursor
8. Link
   - On **click**, places a Link **item** at the cursor

4 Editing

The **board** should display an **transform box** around the currently **selected item**(s)

1. The box should consist of non-filled rectangle around the bounding box of the **item**s
   - If a single **item** is **selected**, and that **item** has a Transform **property**, the bounding box should be rotated and scaled to match the transform
   - If a single **item** is **selected**, and that **item** does not have a Transform **property**, the bounding box should default to the smallest aligned rectangle which completely contains the **item**
   - If multiple **item**s are **selected**, the bounding box should be the smallest aligned rectangle which fully contains the bounding box of every **selected item**
2. The box should be computed once at the beginning of the **selection** and not change relative to the **item**s while the **selection** is active
   - The exception to this is Text and Link **items**, which may change size when edited. As such, no modification to any **item** which may change its size is permitted while multiple **item**s are selected
3. At the corners and midpoints of the edges of the **transform box**, there should be square stretch handles:
   - Both types of handle stretch the selection relative to its centre
   - Midpoint handles stretch the **item**(s) only in the direction normal to the edge
   - Corner handles stretch the **item**(s) in both axes evenly
4. At the "top" of the **transform box** there should be a circular rotation handle
   - Dragging the handle rotates the **selection** around its centre
5. A drag action anywhere else in the **transform box** should translate the **item**(s) along the mouse movement
6. When editing multiple **item**s, or **item**(s) with no Transform **property**, changes may be shared with other clients in a qualitative form, but the final state *as computed by the editing client* must be shared when the **selection** is dropped

5 Synchronisation

The program should enable multiple clients to edit and view a **board** simultaneously

5.1 Synchronicity

Any edits made to the **board** should be displayed on other clients with minimal delay:

1. The creation of simple **item**s should be immediately synchronised

2. The beginning of path **item**s should be immediately synchronised, and any incremental stages should be shared within one second of the individual stage being sent.
3. Any edits made to existing **item**s should be shared within one second of the edit

5.2 Consistency

The board state must be the same across all clients:

1. At any point at which no information remains to be transmitted to or from any client, the complete list of **item**s and their **properties** held by each client must be exactly identical
2. When a client **selects** an **item**, this must be transmitted to other clients, which must not attempt to **select** the same **item**
    1. If two or more clients simultaneously attempt to **select** an **item**, the first to be processed will be successful and the others must be notified that the **item** is unavailable
    2. A client should not attempt to edit an **item** which it has not **selected**, and any attempt to do so must be rejected.
        - The client may permit the user to attempt to edit an **item** before confirmation of **selection** is received, but must not assume that such an edit will be successful
    3. When a client creates an **item**, it will immediately be **selected** and the client may assume this

5.3 Reliability

The program must function as expected whenever possible

1. In the case of client-side interruption, a session should be able to be resumed where possible
    1. If a client loses connection while an **item** is **selected**, the **item** will stay **selected**
    2. Extension - **items** can be reclaimed by the server and **selected** by other clients
2. In the case of a power failure or unexpected termination of the host, the board should *always* be restorable to a state less than 10 seconds (or a configurable duration) old

3 Documented Design

3.1 Key algorithms

3.1.1 Selection

A user's selection is modelled as a separate coordinate space from the global/canvas space, which has its origin at the centre of the user's transform and the same basis at the moment when the selection is initialised.

The selection is then processed in terms of two transformation matrices:

- The Selection Root Transform (abbreviated to SRT)
    - This is the transformation mapping selection space back to canvas space
    - This is updated when the whole transform is edited by dragging it around or rotating it, and regenerated when adding or removing items from the selection
- The Selection Item Transforms (abbreviated to SITs)
    - These transform each item from its original location in canvas space to its current location in selection space
    - These are updated when new items are added to the selection

3.1.1.1 Implementation in SVG

The selection hierarchy is implemented by the following layout

- Root selection container
    - Item container - root for all items in the selection
        - SRT container - applies the SRT to its children
            - SIT containers - Each item is wrapped in a container applying its SIT
        - Staging container - new items being added are moved here so the new bounding box can be computed by the item container

3.1.1.2 Adding items for the selection

The algorithm can be derived based on the following constraints:

- For every item, its final transform must be unchanged by the whole procedure
    - The final transform of an item is equal to the product of the SRT and the item's SIT
    - For new items, this must be the identity
- The new selection space should be axis-aligned with canvas space and have no scaling, and should be centred on the bounding box of the complete set of items - This forces the new SRT to consist only of a translation This leads to the following steps:
- First, compute the bounding box of the new selection
    - Move all new items to the staging container
    - Acquire the bounding box through the SVG `getBBox` API

- The new SRT can be calculated as the translation from the origin to the centre of the bounding box
- For existing items, their new SITs are derived as follows:

$$
\begin{aligned}
R_1 I_1 &= R_0 I_0 \\
R_1^{-1} R_1 I_1 &= R_1^{-1} R_0 I_0 \\
I_1 &= (R_1^{-1} R_0) I_0
\end{aligned}
$$

- For new items, their SITs should be the inverse of the new SRT such that the net transform cancels out

3.1.1.3 Moving the selection

- Dragging is simply translating the SRT
- Rotation:
  - At the beginning of the gesture, store the current SRT and the direction (in canvas space) from the selection origin to the cursor
  - When the cursor moves, find the new direction from the selection origin to the cursor
  - Update the SRT to the initial value rotated by the difference between the initial angle and the current angle
- Stretching:
  - At the beginning of the gesture, store the current SRT and the vector from the selection origin to the cursor
  - When the cursor moves, find the new vector from the selection origin to the cursor
  - The scale factor is computed as the component of the new vector in the direction of the initial one
  - Update the SRT by scaling in the x and/or y directions as appropriate

3.2 Server Overview

The server-side code operates on a coroutine-based event driven system, largely powered by the following constructs:

- Concurrent Hash Map, from `scc`
  - Implements a hash table able to insert and fetch items without locking the table as a whole, allowing the system to scale better on multi-threaded systems
- Multi-producer channels
  - Used in two forms: `mpsc` (Multi-producer single-consumer, from `tokio`) and `mpmc` (Multi-producer multi-consumer, from `async_channel`)
  - `mpsc` channels are used for relaying messages to clients to be stored
  - `mpmc` channels are used for sending messages to the board so that multiple tasks can process messages in parallel
- Read-Write Locks, from `tokio`
  - These are used when a structure needs to be completely owned to be mutated
  - Any number of read-only references to the wrapped value can be held, but writing requires exclusive access
  - Used in this project to wrap sets of IDs since `scc`'s maps don't have a good mechanism for iterating over them

Serving pages and communicating with clients is done through Warp, a lightweight implementation of an HTTP server

- Warp operates though "filters", patterns that match requests and can generate a reply
- Filters can be chained and composed to produce the final application, enabling different subsystems to each produce filters which add together to produce an application

3.2.1 Modules

- `lib` - top-level code including composition of filters and shared object declarations
- `upload` - filters for storing and serving media files
- `client` - filters for establishing a connection between client and server
- `message` - declaration of structures which are sent to and from clients
  - Split across several sub-modules for different sections of the protocol
- `canvas` - declaration of structures relating to the canvas itself
  - `items` - Item type declarations
  - `active` - wrapper around the tables used to store a board
- `board` - implementation of whiteboards
  - `file` - disk format and saving/loading boards
  - `manager` - keeping track of active boards and loading/saving as necessary
  - `active` - processing messages and actually maintaining a board
    - Several sub-modules for implementation of different components of this

3.2.2 Compilation targets

The project has two different compilation outputs:

- `codegen` - building TypeScript declaration files
  - Collects types and outputs them to a configurable location

- `main` - running a server
  - Handles command-line arguments and initialising the server

3.3 Data Structures

3.3.1 Canvas items

3.3.1.1 Basic structures

Aliases for specific uses of types:

```
type Color = string;
```

Point - a general 2D vector:

```
interface Point {
        x: number;
        y: number;
}
```

Stroke - a combination of stroke width and colour:

```
interface Stroke {
        width: number;
        color: Color;
}
```

Transform - a 2D transformation, corresponds to a 3x2 matrix:

```
interface Transform {
        origin: Point;
        basisX: Point;
        basisY: Point;
}
```

Spline Node - a part of a hand-drawn path:

```
interface SplineNode {
        position: Point;
        velocity: Point;
}
```

3.3.1.2 Item types

Common properties of items:

```
interface TransformItem {
        transform: Transform;
}

interface StrokeItem {
        stroke: Stroke;
}

interface FillItem {
        fill: Color;
}
```

Item types:

```
interface RectangleItem extends TransformItem, StrokeItem, FillItem {
        type: "Rectangle";
}

interface EllipseItem extends TransformItem, StrokeItem, FillItem {
        type: "Ellipse";
}

interface LineItem extends StrokeItem {
        type: "Line";
        start: Point;
```

```
        end: Point;
}

interface PolygonItem extends StrokeItem, FillItem {
        type: "Polygon";
        points: Point[];
}

interface PathItem extends StrokeItem, TransformItem {
        type: "Path";
        nodes: SplineNode[];
}

interface ImageItem extends TransformItem {
        type: "Image";
        url: string;
        description: string;
}

interface TextItem extends TransformItem {
        type: "Text";
        text: string;
}

interface LinkItem extends TransformItem {
        type: "Link";
        text: string;
        url: string;
}
```

3.3.1.3 Other supporting types

Location update - change in the position of an item:

```
type LocationUpdate = { "Transform": Transform } | { "Points": Point[]};
```

Batch changes - changes that can be made to multiple items at once:

```
interface BatchChanges {
        fill?: Color;
        stroke?: Stroke;
}
```

3.4 Client-side code

3.4.1 The property system

In several places the application needs to display and store attributes of things, and so to avoid repetition this is described in terms of properties, which are a representation of a data structure along with how it should be displayed.

The first component of this is a `PropKey`, which is a unique identifier for a piece of data stored by a property, along with other information such as a default value and/or data validators.

As a convenience feature, a `CompositeKey` allows bundling the components of a composite data structure into a function which composes the values returned by sub-keys into a single output

The second component is `PropertySchema`, which describes a property with its type, key, and other attributes such as display name and constraints

Finally, the abstract type `PropertyStore` defines the interface for getting and setting properties by key

Alongside this is a basic implementation of a property store, `SingletonPropertyStore`, which simply stores properties in a key-value map

To make properties easier to work with, two systems are implemented:

- Property templates, which build the keys and schema for common properties such as stroke or points
- property builders, which accumulate multiple sets of keys and schemas in a single object to enable properties to be more concisely defined

3.4.1.1 Class tables

(note that in this section "value" refers to the value type corresponding to a key) PropKey:

Field	Type
validator	(value) => bool

Field	Type
defaultVal?	(value)
type	string

CompositeKey<T>

Field	Type
extractor	(property getter) => T

PropertyStore

Method	Type
read(PropKey)	value
read<T>(CompositeKey<T>)	T
store(PropKey, value)	bool
abstract get(PropKey)	value | NoValue
abstract set(PropKey, value)	void

3.4.2 State

The state mechanism prevents manual updates to parts of the application by creating a wrapper with two fundamental operations:

- Read the current value
- Attach a callback to be notified when the value changes

This is then used to implement other operations such as:

- Create a derived state from a function mapping the current value to a new one
- Combining multiple states together in order to create a state derived from multiple inputs

The core of this is the abstract `State<T>` class, which implements the basic operations and has operations for creating derived instances

The other key component is `MutableState<T>`, which is used to provide a source of state and includes several means of modifying the contained value

Most of the other subclasses of `State` wrap one or more source states and produce a new value from them.

3.4.2.1 Class tables

The following shorthand applies to this section:

- action(T) is a function taking a readonly value of T and returning nothing
- map(T, U) is a function taking a readonly value of T and returning a value of U

State<T>

Field	Type
private watchers	Map<number, action(T)>
private weakWatchers	Map<number, WeakRef<action(T)>>

Method	Parameters	Return Type	Notes
get		readonly T	
getSnapshot		T	returns a clone of the held value
protected update	T	void	Updates the held value and runs update callbacks
watch	action(T)	WatchHandle	
watchWeak	action(T)	WatchHandle	
watchOn	object, action(T)	void	Will run the callback for as long as the object is alive
private removeWatcher, private removeWeak	number	void	
derived<U>	map(T, U)	State<U>	
derivedI	string, ...	State	Creates a derived state from calling an instance method of the held value with the specified parameters
derivedT		State	For tuple states, like derived but the callback receives each element of the tuple as a separate parameter
flatten		State	Flattens a state of a state into a single state
inspect<U>	map(T, U)	U	calls the function on the contained value and returns its result
with	...state	State	Collects a tuple of states together and returns a state of a tuple
debug	Logger, string?	this	Watches the state with the provided logger for debugging

MutableState<T>

Method	Parameters	Return	Notes
set	T	void	
updateBy	T => T	void	Shorthand for setting a new value based on the old one
mutate	T => void	void	Allows the held value to be mutated in-place
derivedM<U>	MutableTransformer<T, U>	MutableState<U>	Creates a new mutable state from a two-way mapping
extract<U>	(field name) | MutableExtractor<T, U>	MutableState<U>	Creates a new mutable state extracting a component of the wrapped value, with a shorthand for a field of the value

3.4.3 Channels

A channel consists of a sender and a receiver and is used for asynchronous data streams

3.4.3.1 Class table

Channel<T>

Field	Type
private queue	T[]
private handles	PromiseHandle<T>[]
private closed	bool

Method	Parameters	Type
private handlePromises		void
public push	T	void
private pop		Promise<T>
public close		void
internal getIter		AsyncIterator<T>

3.4.4 Communication with the server

Communication is split into three components:

RawClient, which handles the WebSocket communication and protocol

HttpApi, which is a collection of static methods for the other server APIs

SessionClient, which handles the connection logic and provides an abstract interface to the server protocol

In addition, the IterateReceiver helper type handles decoding multi-part responses from the server

3.4.4.1 Class tables

IterateReceiver

Field	Type
private futureParts	Record<number, item[]>
private nextPart	number
private lastPart	number
finished	bool

HttpApi

Name	Type	Description
openSession(string, ClientInfo)	Promise<Result<ConnectionInfo>>	Calls the HTTP api to open a new session on a board
uploadFile(File)	Promise<URL>	Uploads a file to the server
startTime	Promise<number>	The UNIX timestamp when the server was launched

RawClient

Field (private)	Type
callId	number
socket	WebSocket
calls	Record<number, CallRecord>
notifyCHandlers	callback for each type of Notify-C
iterateReceivers	Record<number, IterateReceiver>
messageQueue	Channel<string>
messageStream	AsyncIter<string>
url	URL

Method	Parameters	Return	Notes
private bindSocket		void	

Method	Parameters	Return	Notes
private sendPayload	string	void	
callMethod	name: string, args	Promise<return type>	
callIterate	name: string, args	AsyncIter<item type[]>	
setNotifyHandler	name: string, callback: (notify args) => void	void	
private handleMessageObject	MsgRecv	void	
private onSocketOpen	Event	void	
private onSocketError	Event	void	
private onSocketMessage	MessageEvent	void	
private onSocketClose	CloseEvent	void	

SessionClient

Field	Type
private rawClient	RawClient
readonly socketUrl	URL
readonly boardName	string
private sessionCode	number
readonly clientID	number
readonly info	ClientInfo

3.4.5 Tools

Tools can be divided into three categories:

- Instantaneous tools, which perform a single action when activated
  - The interface for this is a single `execute` method
- Mode tools, which stay active until deselected
  - This is represented by `bind` and `unbind` methods
- Action tools, which stay active until the user completes a single action
  - This is represented by a `bind` method which takes a callback, which in turn receives an `Action` object when the action starts
  - An `Action` contains two fields, a Promise for the completion of the action and a method to cancel it

Additionally, each tool can have a set of properties connected to it

To support this a set of abstract base classes are defined

- `ToolBase`, which adds a few helper properties for accessing parts of the board
- `InteractiveToolBase`, which builds on `ToolBase` to automatically build a gesture filter from optional virtual methods
- `ActionToolBase`, which builds on `InteractiveToolBase` to implement the action tool interface
- `ModeToolBase`, which similarly builds on `InteractiveToolBase`
- `InstantaneousToolBase`, which directly extends `ToolBase`

3.4.5.1 Class Tables

Aliases

Name	Type
OnBegin	(Action) => void
Action	{ cancel(): void, completion: Promise<void> }

ToolBase

Field (protected)	Type
board	Board
canvas	CanvasController
ctx	CanvasContext

InteractiveToolBase

Field	Type
gestureFilter	FilterHandle

Method	Parameters	Type
private makeFilter		FilterHandle
protected onDragGesture?	DragGestureState	void
protected onPressGesture?	PressGesture	void
protected onLongPressGesture?	LongPressGesture	void

ActionToolBase

Field	Type
private onBegin?	OnBegin
private completionResolve?	() => void

Method	Parameters	Type
bind	OnBegin	void
protected start		void
protected end		void
protected abstract cancel		void

ModeToolBase

Method	Parameters	Type
bind		void
unbind		void

InstantaneousToolBase

	Method
	abstract execute

3.4.6 UI

The structure of the UI is based primarily around icons and panels:

- A `SimpleIcon` loads an SVG icon file and renders it as an image
- A `ToolIcon` wraps a `SimpleIcon` and connects it to a tool as well as showing whether it is selected
- A `VisibilityButton` also wraps a `SimpleIcon` and updates its display to reflect a panel's state
- A `PanelController` wraps an element and adds a `VisibilityButton` along with the logic to show/hide the contents

Finally, the `UIManager` wraps all of these and ties them together

3.4.6.1 Class tables

Aliases

Name	Type
ToolIconCallback	(Tool) => void
PanelEvents	{ cancel(): void }

SimpleIcon

Field	Type
readonly element	HTMLImageElement

ToolIcon

Field	Type
private icon	SimpleIcon
readonly element	HTMLElement
private toolState	DeferredState<ToolState>
readonly active	State<boolean>
onselect?	ToolIconCallback
ondeselect?	ToolIconCallback

VisibilityButton

Field	Type
private container	HTMLDivElement
private icon	SimpleIcon
readonly openState	MutableState<boolean>
readonly events	EventProvider<PanelEvents>

PanelController

Field	Type
private visibility	VisibilityButton
readonly contents	HTMLElement
readonly openState	State<boolean>

Field	Type
readonly events	MultiTargetDispatcher<PanelEvents>
private containerElement	HTMLElement

Method	Type
private getContents	HTMLElement

UIManager

Field	Type
readonly containerElement	HTMLDivElement
readonly viewPanel	PanelController
readonly toolPanel	PanelController
readonly propertiesPanel	PanelController
readonly properties	PropertyEditor
private toolState	MutableState<ToolState>
readonly toolState	State<ToolState>

Method	Parameters	Type
addToolIcon	ToolIcon, "edit" | "view"	void
private cancelTool		void
private onIconSelect	Tool	void
private onIconDeselect	Tool	void
private createPanel	string, State<EnabledState>, ...string[]	PanelController

3.4.7 Basic canvas structure and utilities

The canvas is laid out in SVG as follows:

```
<svg>
  <g class="scaled-root" transform="...">
    <!-- ... -->
  </g>
  <g class="unscaled-root">
    <!-- ... -->
  </g>
</svg>
```

The hierarchy is split into two parts:

- Scaled content, such as items, which should zoom uniformly with the canvas
- Unscaled content, such as UI, which should move to follow the board but not change in size

This structure is implemented by `CanvasContext`, which is the foundation of most SVG manipulation and provides a wide collection of helper methods

To group unscaled elements, `CanvasContext` has an inner class, `UnscaledHandle` that stores a group of elements and enables them to be removed together

Alongside this is a set of small helper types:

- `MatrixHelper` - maintains an SVG transformation from a state of a matrix
- `TranslateHelper` - similar but only does translation
- `TransformHelper` - works with Transform instances along with extra helper methods
- `StrokeHelper` and `FillHelper` - applies properties to an item, including static methods
- `CenterHelper` - wraps an element in a container that translates it to its centre
- `PathHelper` - builds a list of `SplineNodes` and concatenates them into an SVG path string

3.4.7.1 Class tables

CoordinateMapping

Field	Type
screenOrigin	Point
stretch	number
targetOffset	Point

CanvasContext

Field	Type
private svgroot	SVGSVGElement

Field	Type
private scaledRoot	SVGGElement
private unscaledRoot	SVGGElement
readonly coordMapping	State<CoordinateMapping>
readonly cursorPosition	State<Point>
private gestures	GestureHandler

Method	Parameters	Type
createGestureFilter	GestureLayer	FilterHandle
createElement	tag name string	corresponding SVGElement
createTransform	DOMMatrixReadOnly?	SVGTransform
createPoint	Point?	SVGPoint
createPointBy	State<Point>	SVGPoint
createRect	Point, Point	SVGRect
createRootElement	tag name string	corresponding SVGElement
insertScaled	SVGElement	SVGElement
getUnscaledHandle		CanvasContext.UnscaledHandle
getUnscaledPos	State<Point>	State<Point>
createUnscaledElement	tag name string, State<Point>	corresponding SVGElement
insertUnscaled	SVGElement, State<Point>	SVGElement
translate	Point, CoordinateMapping?	Point

CanvasContext.UnscaledHandle

Field	Name
private elements	Set<SVGElement>

Method	Parameters	Type
insert	SVGGraphicsElement, State<Point>	SVGGraphicsElement
insertStatic	SVGGraphicsElement	SVGGraphicsElement
create	tag name string, State<Point>	SVGGraphicsElement
getPoint	State<Point>	SVGPoint
clear		void

3.4.8 Canvas Items

Rendering canvas items to the board is done through the `CanvasItem` abstract class, which provides an API for drawing and manipulating items, including editing and moving.

Since some items have many details in common, some additional abstract classes are defined:

- `StrokeItem` and `FillItem`, which automatically handle stroke and fill respectively
- `TransformMixin`, which wraps around any class deriving from `CanvasItem` to produce a new class which additionally handles transformations

The other key component is properties, so `ItemPropertyStore` is an implementation of `PropertyStore` which holds functions to extract and store keys in items of a specific type, and also references a set of currently selected items.

3.4.8.1 Class Tables

CanvasItem

Field	Type
readonly element	SVGGElement
protected abstract innerElement	SVGGraphicsElement
protected abstract item	Item

Method	Parameters	Type
update	Item	void
protected abstract updateItem	Item	void
static schemaFor	CanvasItem, ItemPropertyStore	PropertySchema[]
protected getBounds		Bounds
testIntersection	Point	bool
abstract getLocationUpdate	DOMMatrix	LocationUpdate
abstract applyLocationUpdate	LocationUpdate	void
protected checkType	Item, string	void

ItemPropertyStore

Field	Type
private currentItems	ItemEntry[]
private accessorTable	Mapping from item types to key-accessor table

Method	Parameters	Type
private getAccessor	ItemType, PropKey	ItemAcc
bindEntries	Iterable<ItemEntry>	void
getter	ItemType, PropKey, (item) => value	void
setter	ItemType, PropKey, (item, value) => void	void

3.4.9 Gesture handling

To enable easy handling of user input, screen interaction is modelled as gestures such as a press or drag.

These are then processed by attempting to match them to filters, which include a test function to check if a gesture is in range, and handlers for one or more gesture types.

To enable more fine-grained control, filters are also divided into priority layers, where the highest-priority layers are checked first.

Filters can also be enabled and disabled freely, so something like a tool can build its filter once and pause it when not active

This system is implemented in terms of three types

- `GestureHandler`, which handles decoding pointer events and dispatching events
- The `FilterHandle` interface, which is the public API for manipulating filters
- The private `FilterImpl` class, which implements `FilterHandle` and communicates information to the `GestureHandler`

3.4.9.1 Class tables

FilterHandle (all methods return the same instance they are called from)

Method	Parameters
pause	
resume	
setTest	(Point) => bool
setMode	FilterMode
addHandler	GestureType, (Gesture) => void
removeHandler	GestureType

FilterImpl (excluding FilterHandle methods)

Field	Type
active	boolean
types	GestureType bitflags
mode	FilterMode
check	(Point) => bool
handlers	Map from GestureType to handler function

FilterLayer

Field	Type
active	Set<FilterImpl>
inactive	WeakSet<FilterImpl>

GestureHandler

Field	Type
filterLayers	Record<GestureLayer, FilterLayer>

Method	Parameters	Type
processEvents	PointerEventStream	void
private handleGesture	Gesture	void
private updateActive	FilterImpl	void
makeFilter	GestureLayer	FilterHandle

3.4.10 Selection

The selection algorithms are described elsewhere, so this section will just list the classes used:

- `ItemHolder` wraps an item and applies a SIT

- `SelectionBorder` draws an outline around a selection
- `HandleBase` and derived classes implement stretch and rotation handles
- `SelectionBox` has code common between both types of selection box
- `RemoteSelection` implements other client's selection
- `LocalSelection` implements the current user selection

3.4.10.1 Class Tables

ItemHolder

Field	Type
readonly element	SVGGElement
readonly sit	SVGMatrix
private transform	SVGTransform

Method	Parameters	Type
updateSit	DOMMatrixReadOnly	void

SelectionBorder

Field	Type
readonly element	SVGPolygonElement
points	SVGPointList (only stored to work around a GC bug causing a crash in firefox)

HandleBase

Field	Type
protected handle	FilterHandle
readonly element	SVGGraphicsElement

Method	Parameters	Type
protected abstract getElement	CanvasContext	SVGGraphicsElement
protected abstract handleGesture	DragGestureState, srt: DOMMatrixReadOnly	void

StretchHandle and RotateHandle add no new fields/methods

StretchHandleSet

Field	Type
handles	StretchHandle[]

SelectionBox

Field	Type
protected rootElement	SVGGElement
protected itemContainer	SVGGElement
protected rootTransform	SVGGElement
private stagingContainer	SVGGElement
protected unscaled	UnscaledHandle
protected srt	MutableState<DOMMatrix>
protected size	State<Point>
protected items	Map<ItemID, ItemHolder>
protected ctx	CanvasContext
protected table	BoardTable

Method	Parameters	Type
addFromTransforms	TransformRecord[], Transform	void
addFromCanvas	ItemEntry[]	arguments for SelectionAddItems

RemoteSelection

Field	Type
private border	SelectionBorder

Method	Parameters	Type
addItems	TransformRecord[], Transform	void
moveItems	Transform, TransformRecord[]	void

LocalSelection

Field (private)	Type
border	SelectionBorder
rotateHandle	RotateHandle
stretchHandles	StretchHandleSet
srtUpdateSent	bool
_sendSrtUpdate	OwnedInterval (small helper type that cancels a periodic function after it has been freed)

Method	Parameters	Type
private updateSrt	DOMMatrix	void
private handleDrag	DragGestureState	void
createAddPayload	ItemEntry[]	SelectionAddItems
getFinalTransforms		Iterable<[ItemId, DOMMatrix]>
clear		void

3.4.11 Managing the canvas

The `CanvasController` class manages the displayed canvas and selection

3.4.11.1 Class tables

CanvasController

Field	Type
private gestures	GestureHandler
private coordMapping	MutableState<CoordinateMapping>
private cursorPos	MutableState<Point>
readonly ctx	CanvasContext
readonly svgElement	SVGSVGElement
private targetRect	DOMRect
readonly elementBounds	State<DOMRect>
private activeGestures	…
private cursorTimeouts	TimeoutMap<number>
private currentCursors	MutableState<Set<number>>
readonly isGesture	State<bool>
readonly propertyStore	ItemPropertyStore
readonly origin	MutableState<Point>
readonly zoom	MutableState<number>

Method	Parameters	Type
probePoint	Point	Iterable<ItemEntry>
getPropertyInstance	ReadonlySet<ItemEntry>	PropertyInstance
private pointerDown	PointerEvent	void
private pointerUp	PointerEvent	void
private pointerMove	PointerEvent	void

3.4.12 Tracking board state

The `BoardTable` class is the core of the system, keeping track of the board state and relaying events to and from other parts of the client.

The other key process is the bootstrap routine, which incrementally fetches data from the server and synchronises the local state with it.

Linked to this is the `ItemEntry`, `RemoteEntry`, and `SelfEntry` types, which track individual pieces of board state and link them to their local representation.

3.4.12.1 Class tables

ItemEntry

Field	Type
id	ItemID
item	Item
canvasItem	CanvasItem
selection	Option<ItemID>

RemoteEntry

Field	Type
id	ClientID

Field	Type
items	Set<ItemID>
info	ClientInfo
connection	ConnectionState
box	Option<RemoteSelection>

SelfEntry

Field	Type
id	ClientID
items	MutableStateSet<ItemID>
info	ClientInfo
connection	ConnectionState
box	Option<LocalSelection>

BoardTable

Field	Type
private items	Map<ItemID, ItemEntry>
private clients	Map<ClientID, RemoteEntry>
private self	SelfEntry
readonly ownID	ClientID
readonly selectedItems	StateSet<ItemEntry>

Method	Parameters	Type
private bootstrap		void
private bindClientEvents		void
private bindSelectionEvents		void
private addClient	ClientID	Promise<void>
addItem	ItemID, Item	void
get	Iterable<ItemID>	Iterable<Option<ItemEntry>>
get	Iterable<ItemID>, true	Iterable<ItemEntry>
get	ItemID	Option<ItemEntry>
entries		Iterable<ItemEntry>
private ensureLocalBox		SelfEntry which definitely has a box
addOwnSelection	ItemID[]	void
moveOwnSelection	Transform	void
cancelSelection		void
editSelectedItem	ItemEntry	void

3.4.13 Combining everything together

The `Board` class contains each of the subsystems and setup logic

3.4.13.1 Class Tables

BoardInfo : ClientInfo

Field	Type
boardName	string
clientID	ClientID

Board

Field	Type
readonly ui	UIManager
readonly client	SessionClient
readonly canvas	CanvasController
readonly table	BoardTable
readonly info	BoardInfo

Method	Parameters	Type
private init		void
private handlePath	ClientID, Stroke, PathID	void

3.5 Server-side code

3.5.1 The canvas

The canvas is tracked using two datastructures:

- A read-write locked set of ItemIDs which can be sequentially iterated over
- A concurrent map from ItemIDs to Items which enables parallel access but cannot be iterated over

Additionally, an atomic counter is used to generate new IDs for every item

Additionally, the ItemRef struct holds a locked entry and dereferences to the held item, allowing items to be retrieved and edited in-place

3.5.1.1 module crate::canvas::active

ItemRef<'a>

Field	Type
0	OccupiedEntry<'a, ItemID, Item> (locked reference to an ItemID:Item HashMap entry valid for 'a)
1	&'a CounterU64 (reference to an atomic U64 valid for 'a)

ActiveCanvas

Field	Type
next_id	AtomicU32
item_ids	RwLock<BTreeSet<ItemID>>
items	scc::HashMap<ItemID, Item>
edit_count	CounterU64

	Name	Receiver	Parameters	Return
pub	new_empty			Self
	get_id	&self		ItemID
pub async	get_ref	&self	ItemID	Option<ItemRef>
pub async	get_item	&self	ItemID	Option<Item>
pub async	add_item	&self	Item	ItemID
pub	add_item_owned	&mut self	Item	ItemID
pub async	scan_items	&self	impl FnMut(ItemID, &Item) (function taking an ItemID and a reference to an item which can be called from an exclusive reference)	()
pub async	get_item_ids	&self		Vec<ItemID>
pub	get_item_ids_sync	&self		Result<Vec<ItemID>, ()>

3.5.2 Communicating with clients

#TODO

3.5.2.1 module crate::client

pub MessagePayload

Field	Type
0	Vec<u8>

ClientMessage

Variant	Fields
Payload	(Vec<u8>)

ClientHandle

Field	Type
message_pipe	UnboundedSender<ClientMessage>

	Name	Receiver	Parameters	Return
	new			(Self, UnboundedReceiver<ClientMessage>)
	send	&self	ClientMessage	()
	send_data	&self	Vec<u8>	()

	Name	Receiver	Parameters	Return
pub	send_message	&self	MsgSend	()
pub	send_payload	&self	&MessagePayload	()

Session

Field	Type
client_id	ClientID
handle	BoardHandle

Name	Receiver	Parameters	Return
connect	&self	ClientHandle	
disconnect	&self		
message	&self	MsgRecv	

pub SessionRegistry - alias of RwLock<HashMap<SessionID, Session>>

Module-level functions

	Name	Parameters	Return
pub	create_client_filter	GlobalRes	BoxedFilter
async	handle_session	Session, WebSocket	()

3.5.3 Interface with board

3.5.3.1 module crate::board

BoardMessage

Variant	Fields
ClientMessage	ClientID, MsgRecv
SessionRequest	ClientInfo, oneshot::Sender<Result<ConnectionInfo, Error>>
ClientConnected	ClientID, ClientHandle
ClientDisconnected	ClientID

pub BoardHandle

Field	Type
message_pipe	async_channel::Sender<BoardMessage>

	Name	Receiver	Parameters	Return
	send_msg	&self	BoardMessage	()
pub async	create_session	&self	ClientInfo	Result<ConnectionInfo, Error>
pub	client_msg	&self	ClientID, MsgRecv	()
pub	client_connected	&self	ClientID, ClientHandle	()
pub	client_disconnected	&self	ClientID	()
	downgrade	&self		WeakHandle

WeakHandle

Field	Type
0	async_channel::WeakSender<BoardMessage>

Name	Receiver	Parameters	Return
upgrade	&self		Option<BoardHandle>

3.5.4 Saving and loading

3.5.4.1 module crate::board::file

BoardFileAttrs

Field	Type
readonly	bool

BoardFile

Field	Type
items	Vec<Item>
(flattened) attrs	BoardFileAttrs

pub BoardFileHandle

Field	Type
file_path	PathBuf
temp_path	PathBuf
attrs	BoardFileAttrs

	Name	Receiver	Parameters	Return
pub	from_path		PathBuf	Self
pub async	load_canvas	&mut self		io::Result<ActiveCanvas>
pub async	save_canvas	&mut self	&ActiveCanvas	io::Result<()>

Module-level methods

	Name	Parameters	Return
	try_get_board	DirEntry	Option<(String, BoardFileHandle)>
pub	get_boards	&Path	io::Result<Vec<(String, BoardFileHandle)>>

3.5.5 Managing boards

3.5.5.1 module crate::board::manager

LoadedState

Field	Type
handle	WeakHandle
canvas	Arc<ActiveCanvas>

	Name	Receiver	Parameters	Return
async	get_or_refresh	&mut self		BoardHandle

ActiveState

Variant	Fields
Loaded	LoadedState
Unloaded	

BoardRef

Field	Type
file	BoardFileHandle
state	ActiveState

pub BoardManager

Field	Type
boards	scc::HashMap<String, BoardRef>

	Name	Receiver	Parameters	Return
pub	new		&Path	Self
pub async	load_board	&self	String	Option<BoardHandle>
pub async	autosave	&self		

3.5.6 Active Boards

3.5.6.1 module crate::board::active

ActivePath

Field	Type
client	ClientID
nodes	Vec<SplineNode>
listeners	Vec<IterateHandle<GetActivePath>>
stroke	Stroke
last_flush	Instant

SelectionState

Field	Type
items	BTreeMap<ItemID, Transform>
own_transform	Transform

ClientState

Field	Type
info	ClientInfo
handle	Option<ClientHandle>
active_paths	Vec<PathID>
selection	SelectionState

Board

Field	Type
client_ids	RwLock<BTreeSet<ClientID>>
clients	scc::HashMap<ClientID, ClientState>
canvas	Arc<ActiveCanvas>
selected_items	scc::HashMap<ItemID, Option<ClientID>>
active_paths	scc::HashMap<PathID, ActivePath>

	Name	Receiver	Parameters	Return
	new_from_canvas		Arc<ActiveCanvas>	Self
	launch	self	tasks: usize	BoardHandle
async	handle_message	&self	BoardMessage	()
async	handle_client_message	&self	ClientID, MsgRecv	()

Module-level functions

	Name	Parameters	Return
pub	from_canvas	Arc<ActiveCanvas>, tasks: usize	BoardHandle

3.5.6.2 module crate::board::active::active_helpers

pub TakeResult

Variant
Successful
NonExistent
Occupied
AlreadyOwned

impl Board

	Name	Receiver	Parameters	Return
pub async	check_owned	&self	&ClientID, &Handle, ItemID	bool
pub async	take_item	&self	&ClientID, &Handle, ItemID	TakeResult
pub async	get_client	&self	&ClientID	OccupiedEntry<ClientID, ClientState>
pub async	get_handle	&self	&ClientID	Option<ClientHandle>
pub async	send_notify_c	&self	NotifyCType	()

impl ClientState

	Name	Receiver	Parameters	Return
pub	try_send	&self	MsgSend	()
pub	try_send_payload	&self	&MessagePayload	()

4 Technical Solution

4.1 Skills Used

4.1.1 Data structures

Structure	Location
Asynchronous Queue	Used in several places: (server) client.rs (server) board/active.rs (client) canvas/Canvas.ts Custom implementation in TypeScript: (client) util/Channel.ts
Object-oriented mechanism for automatically propagating change of state	Implemented in: (client) util/State.ts Used throughout most of client

4.1.2 Algorithms

Algorithm	Location
Dynamic generation of objects based on the OOP model	Dynamically choosing a class based on item type (client) canvas/items/ItemBuilder.ts Instantiating property UI classes (client) ui/PropertyEditor.ts
Server-side scripting with request and response objects to service a complex client-server model	Used throughout most of the server but particularly (server) client.rs
Calling parameterised Web service APIs and parsing JSON to service a complex client-server model	(client) client/HttpApi.ts (client) client/RawClient.ts (server) client.rs to some extent

4.1.3 Other techniques

Technique	Location	Description
Writing to a temporary file to prevent data loss	(server) board/file.rs	Writing to a temporary file and renaming it over the main file ensures that the saved data is never in an incomplete state
Automatically generating bindings for other languages	(server) codegen.rs type definitions in message/*	To ensure consistency between the client and server code, TypeScript definitions are generated directly from Rust source code

4.2 Server-side code

4.2.1 main.rs

```
use std::time::Duration;

use camino::Utf8PathBuf;
use clap::Parser;
use flexi_logger::Logger;
use log::{error, info};
use tokio::runtime;
use virtual_whiteboard::{
        board::BoardManager, create_api_filter, create_media_filter, create_script_filter,
        create_static_filter, ConfigurationBuilder, GlobalRes, GlobalResources,
};
use warp::{filters::BoxedFilter, reply::Reply, Filter};

#[derive(Parser, Debug)]
#[command(author, version, about, long_about = None)]
struct Args {
        // #[arg(short = 'r', long, default_value = ".")]
        // board_root: Utf8PathBuf,
        #[arg(short = 's', long = "static-root")]
        static_path: Utf8PathBuf,

        #[arg(short = 'j', long = "script-root")]
        script_root: Utf8PathBuf,

        #[arg(short = 'm', long = "media-root")]
        media_root: Utf8PathBuf,

        #[arg(short = 'b', long = "board-root")]
        board_root: Utf8PathBuf,

        #[arg(long, default_value_t = true)]
        serve_ts: bool,
}

fn main() -> Result<(), Box<dyn std::error::Error>> {
        let _logger = Logger::try_with_env()?
                //.write_mode(WriteMode::Async)
                .start()?;

        let args = Args::parse();

        let config = ConfigurationBuilder::default()
                .static_root(args.static_path.into())
                .script_root(args.script_root.into())
                .media_root(args.media_root.into())
                .board_root(args.board_root.clone().into())
                .serve_ts(args.serve_ts)
                .build()
```

```
                .unwrap();

        info!("Program Startup");

        info!("Building runtime");

        let runtime = runtime::Builder::new_multi_thread()
                .enable_io()
                .enable_time()
                .worker_threads(2)
                .build()
                .map_err(|e| {
                        error!("Failed to build Tokio runtime: {}", &e);
                        e
                })?;

        info!("Successfully constructed Tokio runtime");

        runtime.block_on(async move {
                info!("Loading boards");
                let boards = BoardManager::new(args.board_root.as_std_path());

                let res = GlobalResources::new(boards, config).as_static();

                tokio::task::spawn(async {
                        loop {
                                info!("Beginning autosave");
                                let current_task = tokio::task::spawn(res.boards.autosave());

                                tokio::time::sleep(Duration::from_secs(10)).await;

                                current_task.await.unwrap_or_else(|e| {
                                        error!("Autosave task panicked: {e}");
                                });
                        }
                });

                let filter = create_filter(res);

                info!("Starting server");
                warp::serve(filter).bind(([0, 0, 0, 0], 8080)).await
        });

        Ok(())
}

fn create_filter(res: GlobalRes) -> BoxedFilter<(impl Reply,)> {
        info!("Building filters");
        let index_filter =
                warp::path("index.html").and(warp::fs::file(res.config.static_root.join("index.html")));
        let api_filter = warp::path("api").and(create_api_filter(res));
        let static_filter = warp::path("static").and(create_static_filter(res));
        let script_filter = warp::path("script").and(create_script_filter(res));
        let media_filter = warp::path("media").and(create_media_filter(res));

        return api_filter
                .or(static_filter)
                .or(index_filter)
                .or(script_filter)
                .or(media_filter)
                .boxed();
}
```

4.2.2 codegen.rs

```
#![cfg(feature = "codegen")]
use camino::Utf8PathBuf;
use clap::Parser;
use itertools::Itertools;
use lazy_static::lazy_static;
use std::{format, fs, string::String};
use ts_rs::TS;
```

```rust
use virtual_whiteboard::message::{
        iterate::IterateSpec, method::MethodSpec, notify_c::NotifyCSpec,
};

#[derive(Parser, Debug)]
#[command(author, version, about, long_about = None)]

struct Args {
        #[arg(short = 'o', long = "out-dir")]
        output_root: Utf8PathBuf,

        #[arg(short = 't', long, default_value_t = false)]
        types: bool,

        #[arg(short = 'm', long, default_value_t = false)]
        methods: bool,

        #[arg(short = 'c', long = "notify-c", default_value_t = false)]
        notify_c: bool,

        #[arg(short = 'i', long = "iterate", default_value_t = false)]
        iterate: bool,

        #[arg(short = 'd', long, default_value_t = false)]
        dry_run: bool,
}

enum ExportTarget {
        Types,
        Methods,
        NotifyC,
        Iterate,
}

impl ExportTarget {
        fn to_str(&self) -> &'static str {
                match self {
                        Self::Types => "Types",
                        Self::Methods => "Methods",
                        Self::NotifyC => "NotifyC",
                        Self::Iterate => "Iterate",
                }
        }

        fn is_enabled(&self) -> bool {
                match self {
                        Self::Types => ARGS.types,
                        Self::Methods => ARGS.methods,
                        Self::NotifyC => ARGS.notify_c,
                        Self::Iterate => ARGS.iterate,
                }
        }
}

lazy_static! {
        static ref ARGS: Args = Args::parse();
}

fn export(target: ExportTarget, content: String) {
        if !target.is_enabled() {
                return;
        }

        let path = ARGS.output_root.join(format!("{}.ts", target.to_str()));

        if ARGS.dry_run {
                println!("======== BEGIN [{}] ========", path,);
                println!("{}", content);
                println!("======== END [{}] ========", path);
        } else {
                fs::write(path, content).unwrap();
        }
}
```

```
fn make_spec_export<Spec: TS>(type_imports: &str, exports: String, names: Vec<String>) -> String {
	let spec_name = Spec::name();
	let spec_export = Spec::decl();
	let names_str = names.iter().map(|s| format!("\"{s}\"")).join(", ");
	format!(
		r#"
// @ts-ignore this is generated code
{type_imports}

{exports}

export {spec_export}

export const {spec_name}Names: (keyof {spec_name})[] = {{
	{names_str}
}};
"#
	)
}

macro_rules! export_scanner {
	( [
		$($type:ty,)*
	] with $param:ident => $func:expr) => {{
		fn scan<$param: TS>() -> String {$func}

		let mut items = Vec::new();

		let str_data = [$(
			{
				items.push(<$type>::name());
				scan::<$type>()
			}
		),*].join("\n");

		(str_data, items)
	}}
}

fn main() {
	let (type_export, names) = {
		use virtual_whiteboard::{
			canvas as c, canvas::item as i, message as m, message::reject as r,
		};
		export_scanner! {
				[
				m::ErrorCode,
				m::Error,
				m::Result,
				m::ClientInfo,
				m::ClientState,
				m::ConnectionInfo,
				m::SessionID,
				m::ClientID,
				m::ItemID,
				m::PathID,
				m::LocationUpdate,
				m::BatchChanges,
				r::RejectLevel,
				r::RejectMessage,
				r::RejectReason,

				c::Point,
				c::Color,
				c::Stroke,
				c::Angle,
				c::Transform,
				c::SplineNode,
				c::Spline,

				i::RectangleItem,
				i::EllipseItem,
				i::LineItem,
				i::PolygonItem,
```

```
                        i::PathItem,
                        i::ImageItem,
                        i::TextItem,
                        i::LinkItem,
                        i::TagItem,
                        i::Item,

                        virtual_whiteboard::tags::TagID,
                ] with T =>format!("export {}", T::decl())
        }
};

export(ExportTarget::Types, type_export);

let names_import = format!(r#"import type {{ {} }} from "./Types";"#, names.join(", "));

let (method_export, method_names) = {
        use virtual_whiteboard::message::method::*;
        export_scanner! {[
                SelectionAddItems,
                SelectionRemoveItems,
                SelectionMove,
                EditBatchItems,
                EditSingleItem,
                DeleteItems,
                CreateItem,
                BeginPath,
                ContinuePath,
                EndPath,
                GetAllItemIDs,
                GetAllClientIDs,
                GetClientState,
        ] with T => T::decl()}
};

export(
        ExportTarget::Methods,
        make_spec_export::<MethodSpec>(&names_import, method_export, method_names),
);

let (notify_c_export, notify_c_names) = {
        use virtual_whiteboard::message::notify_c::*;
        export_scanner!([
                ClientJoined,
                ClientConnected,
                ClientDisconnected,
                ClientExited,
                SelectionItemsAdded,
                SelectionItemsRemoved,
                SelectionMoved,
                BatchItemsEdited,
                SingleItemEdited,
                ItemsDeleted,
                ItemCreated,
                PathStarted,
        ] with T => T::decl())
};

export(
        ExportTarget::NotifyC,
        make_spec_export::<NotifyCSpec>(&names_import, notify_c_export, notify_c_names),
);

let (iterate_export, iterate_names) = {
        use virtual_whiteboard::message::iterate::*;
        export_scanner!([
                GetPartialItems,
                GetFullItems,
                GetActivePath,
                Count,
        ] with T => T::decl())
};

export(
```

```
                ExportTarget::Iterate,
                make_spec_export::<IterateSpec>(&names_import, iterate_export, iterate_names),
        );
}
```

4.2.3 lib.rs

```
//! The virtual whiteboard

#![recursion_limit = "256"] // TT munching
#![warn(missing_docs)]

#[path = "board/board.rs"]
pub mod board;
#[path = "canvas/canvas.rs"]
pub mod canvas;
pub mod client;
#[path = "message/message.rs"]
pub mod message;
#[path = "tags/tags.rs"]
pub mod tags;
pub mod upload;
mod utils;

use std::{path::PathBuf, time::SystemTime};

use board::BoardManager;
use client::{create_client_filter, SessionRegistry};
use upload::create_upload_filter;
use warp::{filters::BoxedFilter, reply::Reply, Filter};

pub use upload::create_media_filter;

/// Global options for the application
#[derive(derive_builder::Builder)]
pub struct Configuration {
        /// Path to static files
        pub static_root: PathBuf,
        /// Path to script files
        pub script_root: PathBuf,
        /// Path to media files (upload and serving)
        pub media_root: PathBuf,
        /// Path to stored boards
        pub board_root: PathBuf,
        /// Whether or not to serve TypeScript files as well as generated JS
        pub serve_ts: bool,
}

/// A container of all resources shared across parts of the application
pub struct GlobalResources {
        /// See [`BoardManager`]
        pub boards: BoardManager,
        sessions: SessionRegistry,
        /// See [`Configuration`]
        pub config: Configuration,
}

/// A reference-counted wrapper of [`GlobalResources`]
pub type GlobalRes = &'static GlobalResources;

impl GlobalResources {
        /// Initialise the structure with the given fields
        pub fn new(boards: BoardManager, config: Configuration) -> Self {
                Self {
                        boards,
                        sessions: SessionRegistry::default(),
                        config,
                }
        }

        /// Move [`self`] into a leaked static reference
        pub fn as_static(self) -> GlobalRes {
```

```
                Box::leak(Box::new(self))
        }
}

fn create_start_time_filter() -> BoxedFilter<(impl Reply,)> {
        let time = SystemTime::now()
                .duration_since(std::time::UNIX_EPOCH)
                .unwrap()
                .as_millis()
                .to_string()
                .into_boxed_str();

        // Leak and reborrow immutably
        let time = &*Box::leak(time);

        warp::path("start_time").map(move || time).boxed()
}

/// Create a warp [`Filter`] handling all dynamic paths
pub fn create_api_filter(res: GlobalRes) -> BoxedFilter<(impl Reply,)> {
        create_start_time_filter()
                .or(create_client_filter(res))
                .or(create_upload_filter(res))
                .boxed()
}

/// Create a warp [`Filter`] serving static files
pub fn create_static_filter(res: GlobalRes) -> BoxedFilter<(impl Reply,)> {
        warp::fs::dir(res.config.static_root.to_owned()).boxed()
}

/// Create a warp [`Filter`] serving scripts, and optionally the original source files
pub fn create_script_filter(res: GlobalRes) -> BoxedFilter<(impl Reply,)> {
        let main_filter = warp::fs::dir(res.config.script_root.join("out"));
        if res.config.serve_ts {
                warp::path("source")
                        .and(warp::fs::dir(res.config.script_root.join("src")))
                        .or(main_filter)
                        .unify()
                        .boxed()
        } else {
                main_filter.boxed()
        }
}
```

4.2.4 utils.rs

```
/// Creates an atomic counter at each invocation site that evaluates to a new value each time
macro_rules! counter {
        ($tname:ident) => {{
                static COUNTER: std::sync::atomic::$tname = std::sync::atomic::$tname::new(0);
                COUNTER.fetch_add(1, std::sync::atomic::Ordering::Relaxed)
        }};
}

pub(crate) use counter;

pub struct CounterU64(std::sync::atomic::AtomicU64);

impl CounterU64 {
        pub fn new() -> Self {
                Self(0.into())
        }

        pub fn next(&self) -> u64 {
                self.0.fetch_add(1, std::sync::atomic::Ordering::Relaxed)
        }

        pub fn get(&self) -> u64 {
                self.0.load(std::sync::atomic::Ordering::Relaxed)
        }
}
```

```
pub struct ResultIter<T, E, TIter: Iterator<Item = Result<T, E>>>(TIter);

impl<T, E, TIter: Iterator<Item = Result<T, E>>> Iterator for ResultIter<T, E, TIter> {
	type Item = T;

	fn next(&mut self) -> Option<Self::Item> {
		loop {
			match self.0.next() {
				Some(Ok(val)) => return Some(val),
				Some(Err(_)) => continue,
				None => return None,
			}
		}
	}
}

pub trait IterExt: Iterator {
	fn filter_ok<T, E>(self) -> ResultIter<T, E, Self>
	where
		Self: Iterator<Item = Result<T, E>> + Sized,
	{
		ResultIter(self)
	}
}

impl<T: Iterator> IterExt for T {}
```

4.2.5 upload.rs

```
//! API routes for file uploads

use std::{
	io,
	path::Path,
	time::{SystemTime, UNIX_EPOCH},
};

use futures_util::StreamExt;
use tokio::io::AsyncWriteExt;
use warp::{
	filters::{
		multipart::{FormData, Part},
		BoxedFilter,
	},
	reply::Reply,
	Filter,
};

use crate::{utils::counter, GlobalRes};

/// Get a semi-unique ID for a file by combining the current time with an execution-unique value
///
/// The only way collisions could occur would be if multiple instances were running in parallel, which would already be a bad idea
fn get_file_id() -> String {
	let time = SystemTime::now()
		.duration_since(UNIX_EPOCH)
		.expect("This code should not be running before the UNIX epoch")
		.as_micros();
	let count = counter!(AtomicUsize);
	format!("{time}-{count}")
}

async fn try_upload_part(target: &Path, mut part: Part) -> io::Result<String> {
	// Attempt to extract just a filename from the provided input
	let name = part
		.filename()
		.and_then(|n| Path::new(n).file_name())
		.ok_or_else(|| io::Error::other("Unable to extract filename"))?;

	let id = get_file_id();
```

```
        let mut path = target.join(&id);
        tokio::fs::create_dir_all(&path).await?;
        // The path returned to the client
        let resource_path = format!(
                "{id}/{}",
                name.to_str()
                        .ok_or_else(|| io::Error::other("Unable to convert name to str"))?
        );
        path.push(name);

        let mut file = tokio::fs::File::create(&path).await?;
        while let Some(Ok(mut buf)) = part.data().await {
                file.write_all_buf(&mut buf).await?;
        }

        file.flush().await?;
        file.sync_all().await?;
        return Ok(resource_path);
}

/// Create a filter that receives files and stores them
pub fn create_upload_filter(res: GlobalRes) -> BoxedFilter<(impl Reply,)> {
        let target = &*res.config.media_root;
        let form_options = warp::multipart::form().max_length(1024 * 1024 * 64);
        warp::path("upload")
                .and(warp::post())
                .and(form_options)
                .and_then(move |mut form: FormData| async move {
                        while let Some(Ok(part)) = form.next().await {
                                if part.name() == "file" {
                                        if let Ok(name) = try_upload_part(target, part).await {
                                                return Ok(name);
                                        }
                                }
                        }
                        Err(warp::reject())
                })
                .boxed()
}

/// Create a filter for serving uploaded files
pub fn create_media_filter(res: GlobalRes) -> BoxedFilter<(impl Reply,)> {
        warp::fs::dir(res.config.media_root.to_owned()).boxed()
}
```

4.2.6 client.rs

```
//! Interfacing with clients
//! The main interface of this module is [`create_client_filter`], which builds a filter to forward WebSocket requests to a
board

use futures_util::{SinkExt, StreamExt};
use log::{error, info, warn};
use tokio::sync::mpsc;
use warp::{
        filters::{
                ws::{Message, WebSocket, Ws},
                BoxedFilter,
        },
        reject::Rejection,
        reply::Reply,
        Filter,
};

use crate::{
        board::BoardHandle,
        message::{ClientID, ClientInfo, MsgRecv, MsgSend, SessionID},
        GlobalRes,
};

/// An opaque payload that can be duplicated and sent to multiple clients
pub struct MessagePayload(Vec<u8>);
```

```rust
impl MessagePayload {
	/// Create a new stored payload from the send message
	pub fn new(msg: &MsgSend) -> Self {
		Self(serde_json::to_vec(msg).expect("Failed to serialize payload"))
	}
}

enum ClientMessage {
	Payload(Vec<u8>),
}

/// A handle used to send messages mack to a client
#[derive(Debug, Clone)]
pub struct ClientHandle {
	message_pipe: mpsc::UnboundedSender<ClientMessage>,
}

impl ClientHandle {
	fn new() -> (Self, mpsc::UnboundedReceiver<ClientMessage>) {
		let (sender, receiver) = mpsc::unbounded_channel();
		(
			Self {
				message_pipe: sender,
			},
			receiver,
		)
	}

	fn send(&self, message: ClientMessage) {
		self.message_pipe.send(message).unwrap_or_else(|e| {
			warn!("Failed to dispatch message to client: {e}");
		})
	}

	fn send_data(&self, data: Vec<u8>) {
		self.send(ClientMessage::Payload(data))
	}

	/// Dispatch a message to a client
	pub fn send_message(&self, message: MsgSend) {
		let payload = MessagePayload::new(&message);
		self.send_data(payload.0);
	}

	/// Send a copy of an existing [`MessagePayload`]
	pub fn send_payload(&self, payload: &MessagePayload) {
		self.send_data(payload.0.clone())
	}
}

/// Max request body length for session creation (1KiB but subject to change)
pub static MAX_SESSION_CREATE_LENGTH: u64 = 1024;

#[derive(Clone)]
struct Session {
	client_id: ClientID,
	handle: BoardHandle,
}

impl Session {
	fn connect(&self, handle: ClientHandle) {
		self.handle.client_connected(self.client_id, handle)
	}

	fn disconnect(&self) {
		self.handle.client_disconnected(self.client_id)
	}

	fn message(&self, msg: MsgRecv) {
		self.handle.client_msg(self.client_id, msg)
	}
}
```

```rust
type RegistryInner = tokio::sync::RwLock<std::collections::HashMap<SessionID, Session>>;

/// Lookup table of session IDs
#[derive(Default)]
pub struct SessionRegistry(RegistryInner);

fn create_session_filter(
	registry: &'static RegistryInner,
) -> impl Filter<Extract = impl Reply, Error = Rejection> {
	warp::path("session")
		.and(warp::path::param())
		.and(warp::ws())
		.and_then(move |id: SessionID, ws: Ws| async move {
			let sessions = registry.read().await;
			if let Some(session) = sessions.get(&id) {
				let session = session.clone();
				Ok(ws.on_upgrade(|ws| async { handle_session(session, ws).await }))
			} else {
				Err(warp::reject())
			}
		})
}

/// Create the board route as a [`Filter`]
pub fn create_client_filter(res: GlobalRes) -> BoxedFilter<(impl Reply,)> {
	let session = create_session_filter(&res.sessions.0);

	let session_create: _ = warp::path!("board" / String)
		.and(warp::body::content_length_limit(MAX_SESSION_CREATE_LENGTH))
		.and(warp::body::json())
		.and_then(|name, info: ClientInfo| async {
			if let Some(handle) = res.boards.load_board(name).await {
				let session = handle.create_session(info).await;
				if let Ok(info) = &session {
					if let Some(_) = res.sessions.0.write().await.insert(
						info.session_id,
						Session {
							client_id: info.client_id,
							handle,
						},
					) {
						error!("Duplicate session ID: {:?}", info.session_id);
					}
				}
				use crate::message::Result;
				Ok(
					serde_json::to_string(&Result::from(session)).unwrap_or_else(|e| {
						error!("Failed to serialize response: {e}");
						String::new()
					}),
				)
			} else {
				Err(warp::reject())
			}
		});
	session.or(session_create).boxed()
}

async fn handle_session(session: Session, ws: WebSocket) {
	let (mut tx, mut rx) = ws.split();

	let (handle, mut board_recv) = ClientHandle::new();

	session.connect(handle);

	tokio::task::spawn(async move {
		while let Some(msg) = board_recv.recv().await {
			match msg {
				ClientMessage::Payload(msg) => {
					tx.send(Message::binary(msg))
						.await
						.unwrap_or_else(|e| warn!("Failed to send WebSocket message: {e}"));
				}
			}
```

```
                }
        });

        while let Some(Ok(msg)) = rx.next().await {
                if msg.is_close() {
                        session.disconnect();
                        info!("Socket closed");
                } else {
                        match serde_json::from_slice(msg.as_bytes()) {
                                Ok(msg) => session.message(msg),
                                Err(e) => {
                                        info!(
                                                "Received malformed message from client: {e}\n{}",
                                                msg.to_str().unwrap_or(""),
                                        )
                                }
                        }
                }
        }
}
```

4.2.7 canvas/canvas.rs

```
//! Collection of types relating to board objects

pub mod active;
pub mod item;

use serde::{Deserialize, Serialize};
#[cfg(feature = "codegen")]
use ts_rs::TS;

pub use active::ActiveCanvas;
pub use item::Item;

/// A global location on the board plane
#[derive(Serialize, Deserialize, Debug, Clone, Copy, Default)]
#[cfg_attr(feature = "codegen", derive(TS))]
pub struct Point {
        x: f64,
        y: f64,
}

/// A CSS-compatible color
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
pub struct Color(String);

/// A descriptor for how to render a line
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
pub struct Stroke {
        /// The thickness of the line, from one side to the other
        pub width: f64,
        /// The color of the line
        pub color: Color,
}

/// An angle, measured in degrees clockwise
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
pub struct Angle(f64);

/// A mapping used to position objects on the board
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[serde(rename_all = "camelCase")]
pub struct Transform {
        /// The global coordinate which the object is centered on and which all other transformations are relative to
        pub origin: Point,

        /// The basis vector of the original X-direction
```

```
        pub basis_x: Point,

        /// The Y-direction basis vector
        pub basis_y: Point,
}

impl Default for Transform {
        fn default() -> Self {
                Self {
                        origin: Point::default(),
                        basis_x: Point { x: 1.0, y: 0.0 },
                        basis_y: Point { x: 0.0, y: 1.0 },
                }
        }
}

/// A point along a [`Spline`]
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
pub struct SplineNode {
        /// The position of the node
        pub position: Point,
        /// The direction of the curve at the node
        pub velocity: Point,
}

/// ### May change at a later date
/// A curved path, currently represented as a series of [`SplineNode`]s
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[non_exhaustive]
pub struct Spline {
        /// The points the path travels through
        pub points: Vec<SplineNode>,
}
```

4.2.8 canvas/item.rs

```
//! The item types themselves

use super::{Color, Point, Spline, Stroke, Transform};
use crate::{
        message::{reject::RejectReason, ItemID, LocationUpdate},
        tags::TagID,
};
use paste::paste;
use serde::{Deserialize, Serialize};
#[cfg(feature = "codegen")]
use ts_rs::TS;

macro_rules! item_enum {
        {
                $(#[$($attr:tt)*])*
                enum $enum_name:ident { $($name:ident),* }
        } => {
                $(#[$($attr)*])*
                pub enum $enum_name {
                        $(
                                $name(paste!([<$name Item>])),
                        )*
                }

                $(
                        impl paste!([<$name Item>]) {
                                /// Wraps self in the Item enum
                                pub fn to_item(self) -> $enum_name {
                                        $enum_name::$name(self)
                                }
                        }
                )*
        }
}
```

```
item_enum! {
        /// A union of all Item types, see the individual types for more information
        #[derive(Serialize, Deserialize, Debug, Clone)]
        #[cfg_attr(feature = "codegen", derive(TS))]
        #[serde(tag = "type")]
        #[non_exhaustive]
        #[allow(missing_docs)]
        enum Item {
                Rectangle,
                Ellipse,
                Line,
                Polygon,
                Path,
                Image,
                Text,
                Link,
                Tag
        }
}

impl Item {
        /// Attempt to update the position of an item, returning the update if it is successful and the original location
if not
        pub fn apply_location_update(
                &mut self,
                id: ItemID,
                update: &LocationUpdate,
        ) -> Result<(), (LocationUpdate, RejectReason)> {
                macro_rules! transform_types {
                        {
                                $($name:ident),* ($item:ident) => $te:expr,
                                $(
                                        $oname:ident($si:ident) => $e:expr
                                ),*$(,)?
                        } => {
                                match self {
                                        $(
                                                Self::$name($item) => {
                                                        $te
                                                },
                                        )*
                                        $(
                                                Self::$oname($si) => $e,
                                        )*
                                }
                        };
                }

                let t = transform_types! {
                        Rectangle, Ellipse, Path, Image, Text, Link, Tag (item) => {
                                if let LocationUpdate::Transform(t) = update {
                                        item.transform = t.clone();
                                        Ok(())
                                } else { Err((
                                        LocationUpdate::Transform(item.transform.clone()),
                                        "Transform",
                                        "Point[]"
                                ))}
                        },
                        Line(item) => {
                                if let LocationUpdate::Points(p) = update {
                                        if p.len() == 2 {
                                                item.start = p[0];
                                                item.end = p[1];
                                                Ok(())
                                        } else { Err((
                                                LocationUpdate::Points(vec![item.start, item.end]),
                                                "Point[2]",
                                                "Point[]"
                                        ))}
                                } else { Err((
                                        LocationUpdate::Points(vec![item.start, item.end]),
                                        "Point[2]",
```

```
                                        "Transform"
                                ))}
                        },
                        Polygon(item) => {
                                if let LocationUpdate::Points(p) = update {
                                        item.points = p.clone();
                                        Ok(())
                                } else { Err((
                                        LocationUpdate::Points(item.points.clone()),
                                        "Point[]",
                                        "Transform"
                                ))}
                        },
                };

                t.map_err(|(update, expected, received)| {
                        (
                                update,
                                RejectReason::IncorrectType {
                                        key: Some(id.to_string()),
                                        expected,
                                        received: received.to_string(),
                                },
                        )
                })
        }
}

/// A rectangle.
///
/// NOTE: The size is implemented through the [`Transform`], instead of a separate property
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[allow(missing_docs)]
pub struct RectangleItem {
        pub transform: Transform,
        pub stroke: Stroke,
        pub fill: Color,
}

/// An ellipse
///
/// NOTE: The size is implemented through the [`Transform`], instead of a separate property
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[allow(missing_docs)]
pub struct EllipseItem {
        pub transform: Transform,
        pub stroke: Stroke,
        pub fill: Color,
}

/// A line segment between two points
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[allow(missing_docs)]
pub struct LineItem {
        pub start: Point,
        pub end: Point,
        pub stroke: Stroke,
}

/// A closed loop of points
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[allow(missing_docs)]
pub struct PolygonItem {
        pub points: Vec<Point>,
        pub stroke: Stroke,
        pub fill: Color,
}

/// A hand-drawn path between two points
#[derive(Serialize, Deserialize, Debug, Clone)]
```

```rust
#[cfg_attr(feature = "codegen", derive(TS))]
#[allow(missing_docs)]
pub struct PathItem {
	pub transform: Transform,
	pub path: Spline,
	pub stroke: Stroke,
}

/// An image stored in a URL
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[allow(missing_docs)]
pub struct ImageItem {
	pub transform: Transform,
	pub url: String,
	pub description: String,
}

/// A text box, rendered with Markdown
///
/// NOTE: The [`Transform`] controls the text box, not the text itself
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[allow(missing_docs)]
pub struct TextItem {
	pub transform: Transform,
	pub text: String,
}

/// A hyperlink
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
#[allow(missing_docs)]
pub struct LinkItem {
	pub transform: Transform,
	pub url: String,
	pub text: String,
}

/// An indexed tag
#[derive(Serialize, Deserialize, Debug, Clone)]
#[cfg_attr(feature = "codegen", derive(TS))]
pub struct TagItem {
	#[allow(missing_docs)]
	pub transform: Transform,
	/// The ID of the tag type
	pub id: TagID,
	/// The data associated with the tag
	pub data: String,
}
```

4.2.9 canvas/active.rs

```rust
//! An implementation of a currently active canvas

use std::{
	collections::BTreeSet,
	ops::{Deref, DerefMut},
	sync::atomic::{AtomicU32, Ordering},
};

use scc::hash_map::OccupiedEntry;
use tokio::sync::RwLock;

use crate::{message::ItemID, utils::CounterU64};

use super::Item;

/// An open canvas
pub struct ActiveCanvas {
	next_id: AtomicU32,
	item_ids: RwLock<BTreeSet<ItemID>>,
```

```
        items: scc::HashMap<ItemID, Item>,
        edit_count: CounterU64,
}

/// A lock-holding reference to an item on the board
pub struct ItemRef<'a>(OccupiedEntry<'a, ItemID, Item>, &'a CounterU64);

impl<'a> Deref for ItemRef<'a> {
        type Target = Item;
        fn deref(&self) -> &Self::Target {
                self.0.get()
        }
}

impl<'a> DerefMut for ItemRef<'a> {
        fn deref_mut(&mut self) -> &mut Self::Target {
                self.1.next();
                self.0.get_mut()
        }
}

impl ActiveCanvas {
        /// Create a new empty canvas
        pub fn new_empty() -> Self {
                Self {
                        next_id: AtomicU32::new(1),
                        item_ids: Default::default(),
                        items: Default::default(),
                        edit_count: CounterU64::new(),
                }
        }

        fn get_id(&self) -> ItemID {
                let val = self.next_id.fetch_add(1, Ordering::Relaxed);
                ItemID(val)
        }

        /// Get a reference to an item on the canvas
        pub async fn get_ref(&self, id: ItemID) -> Option<ItemRef> {
                Some(ItemRef(self.items.get_async(&id).await?, &self.edit_count))
        }

        /// Retrieve the specified item if present
        pub async fn get_item(&self, id: ItemID) -> Option<Item> {
                Some(self.items.get_async(&id).await?.get().clone())
        }

        /// Insert a new item on the canvas and return an ID for it
        pub async fn add_item(&self, item: Item) -> ItemID {
                let id = self.get_id();
                self.items
                        .insert_async(id, item)
                        .await
                        .expect("Duplicate Item ID, something is wrong");
                self.item_ids.write().await.insert(id);
                self.edit_count.next();
                id
        }

        /// Insert a new item synchronously from an exclusive reference
        pub fn add_item_owned(&mut self, item: Item) -> ItemID {
                let id = self.get_id();
                self.items
                        .insert(id, item)
                        .expect("Duplicate Item ID, something is wrong");
                self.item_ids.get_mut().insert(id);
                id
        }

        /// Run the provided callback on each item in the canvas
        pub async fn scan_items(&self, mut f: impl FnMut(ItemID, &Item)) {
                self.items.scan_async(|&id, item| f(id, item)).await
        }
```

```
        /// Get a vector of all current Item IDs
        pub async fn get_item_ids(&self) -> Vec<ItemID> {
                self.item_ids.read().await.iter().cloned().collect()
        }

        /// Try to read the current ItemIDs without blocking
        pub fn get_item_ids_sync(&self) -> Result<Vec<ItemID>, ()> {
                let ids = self.item_ids.try_read().or(Err(()))?;
                Ok(ids.iter().cloned().collect())
        }
}
```

4.3 Client-side code

4.3.1 Board.ts

```
import { Logger } from "./Logger.js";
import { CanvasController } from "./canvas/Canvas.js";
import { StrokeHelper } from "./canvas/CanvasBase.js";
import { BoardTable } from "./BoardTable.js";
import { PathHelper } from "./canvas/Path.js";
import { SessionClient } from "./client/Client.js";
import { ClientID, ClientInfo, PathID, Stroke } from "./gen/Types.js";
import { ToolIcon } from "./ui/Icon.js";
import { createEditToolList, createViewToolList } from "./ui/ToolLayout.js";
import { UIManager } from "./ui/UIManager.js";
import { None } from "./util/Utils.js";

const logger = new Logger("board");

type BoardInfo = ClientInfo & {
        boardName: string,
        clientID: number,
}

export class Board {
        public static async new(name: string, info: ClientInfo): Promise<Board> {
                const client = await SessionClient.new(name, info);
                const table = new BoardTable(client);
                const canvas = new CanvasController(table);
                const ui = new UIManager(canvas, table);
                const boardInfo = { ...info, boardName: name, clientID: client.clientID };

                const board = new this(ui, client, canvas, table, boardInfo);

                queueMicrotask(() => board.init());

                return board;
        }

        private constructor(
                public readonly ui: UIManager,
                public readonly client: SessionClient,
                public readonly canvas: CanvasController,
                public readonly table: BoardTable,
                public readonly info: BoardInfo,
        ) { }

        private async init() {
                for (const [name, tool] of createEditToolList(this)) {
                        const icon = new ToolIcon(name, tool);
                        this.ui.addToolIcon(icon, "edit");
                }

                for (const [name, tool] of createViewToolList(this)) {
                        const icon = new ToolIcon(name, tool);
                        this.ui.addToolIcon(icon, "view");
                }

                this.ui.containerElement.classList.setBy("gesture-active", this.canvas.isGesture);

                this.client.bindNotify("PathStarted", ({ path, stroke, client }) => {
```

```
                        this.handlePath(client, stroke, path);
                });
        }

        private async handlePath(client: ClientID, stroke: Stroke, path: PathID) {
                if (client == this.client.clientID) return;

                const points = this.client.iterate.GetActivePath({
                        path,
                });

                const first = await points.next();

                if (first === None) return;

                const pathElem = this.canvas.ctx.createRootElement("path");
                pathElem.setAttribute("fill", "none");


                const helper = new PathHelper(pathElem, first.shift()!.position);
                new StrokeHelper(pathElem.style, stroke);

                helper.addNodes(first);

                for await (const chunk of points) {
                        helper.addNodes(chunk);
                }

                pathElem.remove();
        }
}
```

5 System Testing

5.1 Test Plan

5.1.1 Layout and interface

- Open program
- Verify that basic layout is as described in objectives
  - (1.1, 1.2.1, 1.3, 1.4, 1.5)

5.1.2 Items and Tools

5.1.2.1 Creation tools

For each of the following tools, perform the following steps:

- Select the tool from the tool panel
- If the tool has properties, change the values of them
- Attempt to use the tool with a press or drag as appropriate
- Verify that the item created reflects the property values set
- If the tool is supposed to support both drag and press inputs, deselect the item and create another with the other input mode
- Attempt to set properties of the created item and verify that they are immediately reflected on the board Tools with only drag input:
- Pen
- Line Tools with only click input:
- Image
- Text
- Link
- Polygon Tools with both click and drag input:
- Rectangle
- Ellipse

5.1.2.2 Selection tool

- Using the selection tool, select one item and verify that its specific properties can be edited
- Select more items and verify that fill and stroke can be edited for multiple items simultaneously
- When selecting and deselecting items, verify that items never move during these operations

5.1.2.3 View tools

- While using other tools, occasionally verify that the following tools may be used and behave appropriately:
  - Pan
  - Zoom In
  - Reset Zoom
  - Zoom Out

5.1.3 Synchronisation and reliability

5.1.3.1 Basic synchronisation

- Open the board on multiple different devices
- Create and edit several items simultaneously
- Verify that the board state is shared correctly

5.1.3.2 Consistency and reliability

Verify that the following do not cause any significant loss of data

- Reloading a client while editing the board
- Disconnecting a client from the network while editing the board
- Restarting the server while editing the board

6 Evaluation