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 a **multi-press action**, the toolbox should be **disabled** and its **visibility button** replaced with a button to cancel the **multi-press action**.

1.3 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, any **item**-specific actions, along with the ability to delete the **item**
2. 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
3. When no **item**s are **selected**, the **properties** of the current **tool** are displayed, if any.
4. If there are no **properties** that can be displayed, the **panel** should be **disabled**

1.4 View controls

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

1. Zoom in
2. Zoom out
3. Reset zoom

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**
        - Actions: Swap start/end
    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 rendered using Markdown to enable formatting
    - Transform is applied to the text box, not the text itself, so font sizing is unaffected by scaling
    - If the text can no longer fit in the box after editing, the box should be automatically stretched vertically to accommodate this.
    - **Properties**: Transform
        - Font style: Enumeration of a small selection of fonts
        - Font size: Number (standard font size unit)
        - Text: **text**
            - If the **item** is **deselected** while the property consists only of whitespace it should be deleted
    - Actions:
        - Bold, Italic, Strikethrough, Underline
            - Apply the formatting options (through Markdown) around the cursor (or the whole field if nothing is selected)
6. Link: a hyperlink
    - Displays a clickable link which opens in a new tab
    - **Properties**: Transform
        - Text: **string**
        - URL: **string**

- Tag: an indexed searchable identifier
    - **Properties**: Transform
        - Name: **string**
        - Data: **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 activation, prompts the user to either upload an image or input a URL, then places it at the centre of the screen
        - The image is then selected and the tool is then immediately deactivated
7. Text
    - On click, creates a Text item and moves keyboard focus to editing the text property
8. Link
    - On click, prompts the user to input a description and URL, and places a Link item at the cursor

4 Editing

4.1 Controls

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 items
    - 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 items 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 items 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 items are selected
3. At the corners and midpoints of the edges of the transform box, there should be square stretch handles:
    - Midpoint handles stretch the item(s) only in the direction normal to the edge, such that the opposite edge is not affected by the movement
    - Corner handles stretch the item(s) such that the opposite corner does not move and the aspect ratio of the selection is unchanged
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 items, 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-like **item**s should be immediately synchronised, and any incremental stages should be shared within one second of the individual stage being sent. (This enables bundling of edits in the case of limited bandwidth)
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 network failure, the program should quietly alert the user and enable changes to be made client-side where possible
    1. The program should attempt to reconnect periodically and notify the user when it is successful
    2. If a client loses connection while an **item** is **selected**, the **item** will stay **selected** for a configurable duration before other clients are able to **select** it
    3. Conflicting changes made during the interruption should be resolved by following the most recent edit
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
    - UI container - root for additional UI - border box, drag handles, etc.

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 Client Overview

Data flow throughout the application is based on a few key mechanisms:

- `State`, a value which can change throughout the lifetime of the application
  - To reduce the need to manually pass around callbacks and propagate notifications, `State` encapsulates this process into an interface 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
  -
- `Channel`, an asynchronous queue
  - A channel consists of a sender with the ability to push items, and a receiver which is an asynchronous iterable of items

3.2.1 Key Modules

- GenWrapper
- Properties
- client
  - IterateReceiver
  - RawClient
  - Client
  - HttpApi
- ui
  - Icon
  - Panel
  - PropertiesEditor
    - ResourcePicker
  - UIManager
- canvas
  - CanvasBase
  - CanvasItems

    - Gesture
    - Selection
        - SelectionUI
    - Canvas

3.3 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

3.3.1 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

3.3.2 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

3.4 Data Structures

3.4.1 Canvas items

3.4.1.1 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;
}
```

3.4.1.2 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;
}
```

3.4.1.3 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;
}
```

4 Technical Solution

4.1 Server-side code

4.1.1 main.rs

```
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.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");
                // The board manager should stay alive for the lifetime of the program
                let boards = BoardManager::new_debug();

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

                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 Client-side code

4.2.1 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 } from "./ui/ToolLayout.js";
import { UIManager } from "./ui/UIManager.js";
import { AsyncIter } from "./util/AsyncIter.js";
import { None } from "./util/Utils.js";
```

```typescript
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 items = new BoardTable(client);
		const canvas = new CanvasController(items);
		const ui = new UIManager(canvas);
		const boardInfo = { ...info, boardName: name, clientID: client.clientID };

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

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

		return board;
	}

	private constructor(
		public readonly ui: UIManager,
		public readonly client: SessionClient,
		public readonly canvas: CanvasController,
		public readonly items: 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);
		}

		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.createElement("path");
		pathElem.setAttribute("fill", "none");

		this.canvas.addRawElement(pathElem);

		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

6 Evaluation