Development of a Multi-Platform Whiteboard Application

Lucas Soares
lucas.soares@tecnico.ulisboa.pt
Instituto Superior Técnico
Lisbon, Portugal

Abstract

The goal of this project is to develop an innovative whiteboard application supported in multi-platforms that aims to facilitate the presentation and manipulation of handwritten mathematical content. The project will use as a starting point the application Xournal++ Mobile. The idea is to extend the editor with features that allow for reliable input and manipulation of mathematical content while supporting remote and self-learning. Given the recent shift to online/remote work and teaching, a whiteboard application that can be used regardless of the device choice can have a massive impact on note-taking and the overall learning experience. The idea is to make a solid whiteboard application to assist the development of the first mathematical structure editor using a cross-platform app software.

Keywords: handwritten mathematics; digital ink; structure editor; tablet PCs; pen-based input; mathematical sketching; mathematical expression recognition; gestures; STEM education; calculational method; educational technology; intelligent tools; sketch recognition

1 Introduction

We are experiencing a technological revolution in the pedagogical area, progressively improving communication means and teaching methods. The launch of electronic presentations was a big mark in this process. It allows the instructor to prepare classroom presentations and present high-quality content with the benefit of saving time in class and avoiding human errors [5, 12]. Technology has also shown to be imperative during the COVID pandemic, when multiple institutions were forced to shut down their installations.

Electronic presentations are very reliable for most areas but have significant downsides in areas that involve writing mathematical formulae and expressions such as in STEM education. Commonly used input devices like keyboards and mice do not support most mathematical symbols and involve high cognitive load. Blackboards solve this kind of drawback but imply that the presenter writes everything in presentation time, which consumes a high amount of time and can lead to errors [3, 17].

Digital ink could be used to teach students how to solve problems related to mathematical content. In addition to solving problems, digital ink lets students express themselves in innumerous ways. It lowers barriers so that more students feel comfortable participating in class and reduce the high cognitive load associated with common input methods. Digital ink also facilitates active learning, where students are directly engaged in the learning process [1, 9, 10, 16].

However, because manipulating mathematical content involves a large number of syntactic manipulations, users often find themselves overloaded with tasks that could be optimized through computer software used in tablet PCs. For this reason, there is a need for an interface able to effectively and reliably input and manipulate mathematical content to improve the overall experience of writing and make the transition to tablets as smooth as possible [13].

1.1 Work Objectives

The main goal of this thesis is to enrich the state of the art by developing the most advanced whiteboard application built with Flutter, supported in multi-platforms. The idea is to construct a solid starting point for the development of the first structure editor supported in multi-platforms.

This application will be developed to support and improve the presentation and manipulation of any written content through features that aim to make the process more interactive and intuitive, less time-consuming, and prevent human errors. The system should provide flexible and reliable tools that assist the user in writing and displaying content. By combining digital ink with the benefits of computer software, the idea is to improve the overall note-taking experience in multiple devices to build the starting point for the first structure editor using cross-platform software.

It will be discussed which cross-platform frameworks available are the best to use and some advantages and disadvantages of this kind of software compared with native development. Finally, since Flutter was the toolkit used, some other similar will be surveyed.

2 Background and Related Work

Topics introduced in the previous section will now be explored in further detail. To better understand this work's objectives and thereafter the proposed solution, software

that contributed to the state of the art will be analysed in order to build up a concrete set of ideas that will be helpful to the development of this project.

2.1 Pen-based devices and Digital Ink

Instructors are increasingly relying on digitally projected slides rather than using blackboards and whiteboards to write and display content. It allows the preparation of high-quality content in advance without requiring the instructor to write everything during the presentation. While allowing easy sharing and reuse of materials, it also facilitates distance learning.

This kind of tool lacks the flexibility to adjust the materials in lecture time so the natural response is to integrate digital ink, giving instructors the flexibility they need to adjust previously prepared materials [2].

2.1.1 Overview. Digital ink refers to the technology that represents handwritten content in a digital kind of way. The majority of these systems use some kind of pen, stylus, or even the user's finger over a digitizer laid under an LCD screen to record what is being written. Ink can change colors, be moved and resized, be transformed into standardized text, among other things.

2.1.2 Role in education. Digital Ink systems are becoming more popular over the years across a wide variety of domains. One key ability that makes digital ink beneficial for learning is its potential to support intelligent interaction and visualization features [11]. By encouraging students to engage with the content, technology may promote active learning. To better understand a concept, tutoring experiences guide students through a series of interactive activities. Because the majority of these tools are WIMP-based (Windows, Icons, Menus, Pointers), we end up with low fluidity levels and a slower learning rate, which affects the learning process and might distract the student from his task [4, 8, 15].

Digital Ink allows the interactive tutoring experience without the complications associated with these kinds of tools. Because each activity is followed by feedback, students tend to understand and retain better what is being taught. This statement is supported by the Constructivist Theory which says that to get a deep understanding of a topic, it is very important for the learner to be actively engaged in the process [6, 7, 14].

2.2 Cross-Platform Application Frameworks

We need a larger user base to generate more income. Furthermore, in today's quickly evolving technological world, we have the need for robust cross-platform app frameworks. The reason for this is simple: creating a cross-platform software allows our product to reach a wider audience at a lower cost. Based on the objectives of the future mobile application, we may choose one of two development paths: create two or more native apps, or create a single cross-platform app that works on many devices at once. When we have a large potential, limited time, and a short budget, a cross-platform software is the appropriate choice. Another reason to create a cross-platform mobile app might be if we want a simple app with no complicated animations or functionality.

The need for cross-platform app development frameworks has skyrocketed. The main reason for this surge in demand is that cross-platform apps have a far greater reach than native apps. Cross platform applications enable the access to a larger number of individuals. Developers created customized frameworks to make the cross-platform app development task more efficient. Cross-platform app frameworks enable developers to create mobile apps with a single line of code and run them on many devices with few adjustments. There are numerous good cross-platform frameworks for mobile app development available nowadays that allows us to build high-quality apps.

2.2.1 Cross-platform technology options. We can build cross-platform apps using 3 different options:

- Web-based Apps

A web-based application is software that is accessible via HTTP over a network connection rather than being stored in memory on a device. Web applications can be used on all mobile and desktop platforms, but they lack native mobile functionalities. PWA (Progressive Web App) capabilities may also be integrated with the recently announced Web APIs, allowing developers to construct web apps that operate similarly to native apps.

- Hybrid Apps

Hybrid apps are built using web technologies and operate on the device (HTML5, CSS and JavaScript). Hybrid apps run inside a native container and use the browser engine to render HTML and handle JavaScript locally. A web-to-native abstraction layer gives you access to device features like the accelerometer, camera, and local storage that aren't available in Mobile Web app. The main drawbacks are the lack of performance compared to native apps and the native feature limitations.

- Cross-Platform Native Apps

The process of developing an app that runs across many platforms is referred to as cross-platform development. This is accomplished utilizing tools such as Flutter, React Native, and Xamarin, and the resulting apps may be used on both Android and iOS. While this saves time and money, it comes at the expense of quality because the program will require an additional abstraction layer to execute.

2.2.2 Pros and Cons. Our solution can run on numerous mobile operating systems and is created in a single programming language thanks to cross-platform development. When an app's code is complete, it passes through a bridge that

converts it to iOS or Android's native APIs. It helps us to create the system quickly; it saves us money since we only need one codebase to operate the app on many operating systems; it provides for reusable code; and it allows us to reach a huge portion of the mobile app market.

On the other hand, we have limited access to some native features and can encounter issues with user experience. If we don't want to compromise the user experience and want to take advantage of all of a phone's native features, we may go with native app development.

3 Development of the Whiteboard Application

3.1 Initial changes to the interface

I decided to maintain the original application layout and overall organization (with minor adjustments) as it was already well designed and well structured. In this section, I will mention all the changes I made to the original interface and explain the reason behind them. All the changes and new features implemented were taking in consideration that the application is designed mainly for pen-based devices, although it can be used on computers and smartphones as well. The image below 1 shows the final application interface. I zoomed the application screen so it would be more visible in the document. The application has the same aspect as the original, although it seems different.

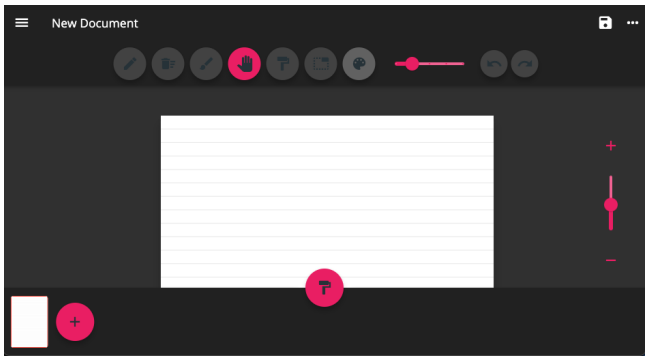

Figure 1. Final interface of the application

Staring with the toolbar. The toolbar was aligned at the left end of the bar, which seemed off from the rest of the interface. I went for a center alignment for design and functional purposes because it is easier for the user to change between tools, as they are much closer to the canvas and consequently pressed much faster. There were a lot of icons with not implemented features (Text, LaTeX, Whiteout eraser, Image, and Select) which I removed for being a waste of space and to avoid misguiding the user to click on icons without functionality. I ended up with a much more clean, precise and easy to use toolbar.

Regarding the stroke width bar, it makes no sense for the bar to have values from 0.1 to 30 because in this range of

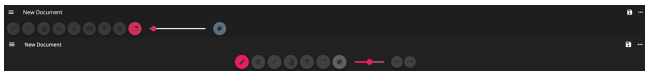

Figure 2. On top: old tool bar; on bottom: new tool bar

values, we find overly thin and overly thick strokes that are just unpractical to use. I replaced the original width bar for a quantitative stroke width range (from 1 to 5, with 3 as a starting value) and added a dynamic text box indicating the current width size. In this scenario, all the values available are plausible to use and the user has real-time knowledge regarding the exact stroke width he is choosing and is much easier to replicate a previously used stroke width.

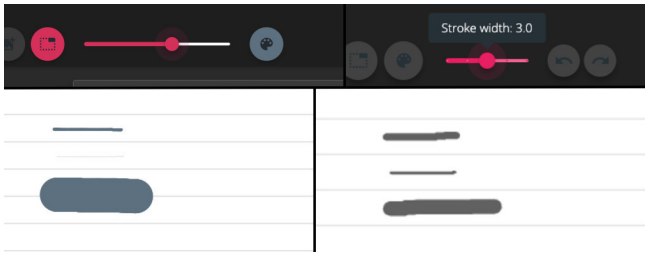

Figure 3. on the left: old stroke width bar and old stroke width; on the right: new stroke width bar and new stroke width

Zoom In/Out bar was standing out (in a negative way) from the rest of the application, as it was barely noticeable. I changed the button and background colors of the bar so it could match with the interface design. I added color to the "+" and "-" buttons to add some contrast with the background and added a tooltip for both the plus and the minus buttons (tooltips can be seen on hover, in case the input device is a mouse, and on long press, in case the input device is a stylus pen or any other similar device). With these changes the zoom bar is way more visible and included in the overall app design.

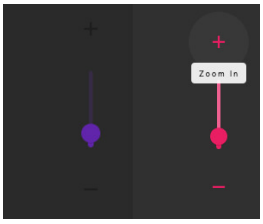

Figure 4. on the left: old Zoom In/Out bar; on the right: new Zoom In/Out bar

I have fixed the way that the stroke is dealt with outside the canvas area. This was giving a false notion of canvas area as this update was not made in real-time. I solved this problem by wrapping the Stack (in which the CustomPaint

responsible for the drawing was inserted in) with a ClipRect. This prevented the user from drawing outside the canvas. If the user comes back to the canvas, the stroke will continue and will be dealt as the same stroke.

Figure 5. on the left: stroke before finishing the action ; on the right: stroke after update

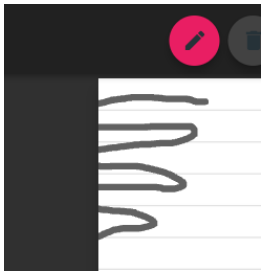

Figure 6. stroke constantly being drawn inside and outisde canvas area

3.2 New features implementation

In terms of what was already implemented in the project, there were some drawbacks regarding usability. In this section, I will be mentioning those usability flaws and how I manage to solve them. I will also be explaining the new features that were implemented and giving an explanation for my decisions.

3.2.1 Eraser. The eraser tool had some malfunctions associated. Although in the majority of times it worked properly, there were times where the stroke we were trying to erase would not get precisely cut apart (or would not get erased at all). I could not find the reason for what was happening, as I couldn't distinguish if it was a consequence of a width variation malfunction or simply just a wrong implementation.

Because we could not completely rely on this eraser, I implemented another type of stroke deletion: the erase-by-stroke tool. I decided to keep the original eraser tool as well as they both have different uses (but not as a default tool). With this implementation, we end up with two eraser tools. For more general uses, the default tool allows us to delete whole strokes/set of strokes. If we only want to partially delete a stroke, we use the second eraser. In the sequence below 7, we can see how both tools work and how we can change between them.

As soon as we hover (or long-press) on the eraser button, the tooltip "Eraser by Stroke" immediately shows so that the user knows the tool he is using (and more specifically which kind of eraser). The eraser deletes the whole stroke (or multiple ones) as soon as the mouse pointer (or pen) collides with the content coordinates. Plus sign (+) was deleted on step 3. If we want to change between eraser tools, we just need to click again on the eraser icon. As we can see in step 4, the icon and the tooltip changed to the corresponding eraser tool. Finally, in step 5, we can see the letter "a" being partially deleted. The stroke is now handled as two different strokes and the rest of the content stored in the array is shifted to the right.

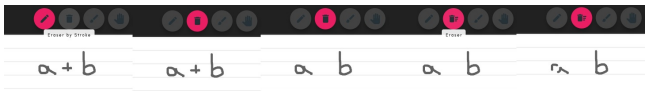

Figure 7. Example of how both delete methods work and how we can switch between them. Expression is written and Eraser button is hovered/long pressed. Eraser button is pressed; User clicks on the "+" sign; Eraser button is pressed; User hovers on the canvas

3.2.2 Highlighter. After we used the highlighter, the new highlight stroke would have no opacity whatsoever, acting like a normal pen stroke with 5 times its width, which is the predefined width value for highlighter. I later discovered that the stroke would acquire highlighter properties (0.5 opacity) as we were deleting it. Anyhow, this tool wasn't usable, so I re implemented it as we can see in the figure bellow. The highlighter can be useful when the user wants to emphasize some of his written content as the new stroke is see through. (Highlighter stroke is dealt as a normal stroke when performing the actions described above and bellow).

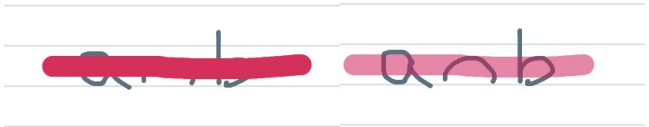

Figure 8. On the left: previously implemented highlighter; on the right: my implementation

3.2.3 Whiteout eraser. I implemented the whiteout eraser functionality, which is kind of self-explanatory. Similar to the previews tools, has a tooltip associated with the button to help the user always be sure of what kind of tool he is using. After the user selects the "Whiteout eraser" button, as soon as he clicks on the canvas area all the content is immediately erased. For preventing miss clicks, this functionality was purposefully not executed on button press but on canvas press, with immediate visual effect. If by any means the user still miss-clicks and erases the whole page, it is possible to revert the action by clicking again in the Whiteout eraser

Icon. All the content that has been deleted is stored until the user uses another tool. For example, if the user whiteouts everything and then selects the pen tool and writes new content, it is no longer possible for the old content to be recovered. In the original Xournal++ Mobile, the user could also delete everything that was written on the canvas by creating a new file. The problem with creating a new file is that it also restarts the color we were using, the stroke width, the canvas position on the screen, the zoom, the page background color, and, in the case we were using multiple pages, all the remaining content.

The whiteout eraser tool allows us to keep all the previously mentioned settings and start on an empty canvas, with the possibility to revert the action (which is not possible if we create a new file). By being in the center of the screen in the toolbar, also facilitates performing this action. This tool can be handy when the user wants to delete everything that is written on the canvas and wants to keep the previously set up settings.

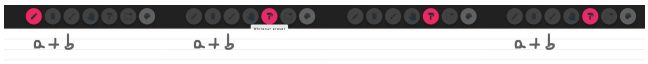

Figure 9. Sequence of the whiteout tool being used: Expression is written; Whiteout Eraser button is pressed; Canvas is pressed; Whiteout Eraser button is pressed again.

3.2.4 Undo and Redo. I implemented the Undo and the Redo buttons that, just like the whiteout, are self-explanatory. These two buttons (alongside the Color button) are the only buttons that do not permanently change color when pressed, as the change of color only occurs when a tool is selected. The Color button changes color after a new color is selected from the color pallet and only changes again when we select a different color. When Undo/Redo buttons are pressed, they suffer from a fade in/fade out kind of animation, changing from their color to a more light one and changing back to the original color in a small matter of time, so the user knows the button as been used.

The Undo allows us to revert the last added stroke all the way back to an empty canvas and the Redo allows us to advance all the way forward to our starting point (before we used Redo). As soon as we introduce a new stroke to the canvas, all the previously deleted strokes are permanently deleted and are no longer available for recovery through the Redo. These tools work together with both eraser tools, sharing a similar behavior. If a stroke has been deleted by Eraser-by-Stroke, it is recoverable through the Redo button. If a stroke has been partially erased or cut apart leaving us with two strokes (through the other eraser), is possible to reintegrate the deleted part back into the original stroke. I opted not to include what had been deleted by the Whiteout Eraser for recovery because I already included a feature to undo that action in the Whiteout Eraser itself.

This feature can be useful if the user mistakenly erased some content he wants to recover or wants to remove what has just been drawn, without having to select it. The figure bellow shows 3 different scenarios of this functionality 10.

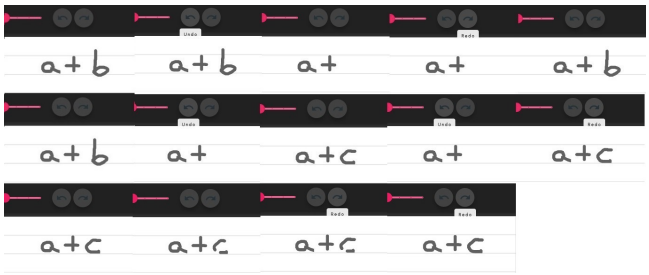

Figure 10. Sequence of the Undo/Redo tools being used.
First scenario: Expression is written; Undo button is hovered/long pressed; Undo button is pressed; Redo button is hovered/long pressed; Redo button is pressed.
Second scenario: Expression is written; Undo button is pressed; New stroke is drawn; Undo button is pressed; Redo button is pressed.
Third scenario: Expression is written; Eraser is used; Redo button is pressed; Redo button is pressed.

3.2.5 Gestures. I implemented some features that are not as evident as the previously mentioned, as they are not triggered by buttons but by gestures instead. For example, if the user is using the pen tool and wants to change to eraser mode, it is possible to switch between these two without actually having to press the eraser button. By simply double tapping the canvas on pen mode the eraser tool is immediately triggered. The inverse is also possible. If the user has the eraser mode activated and wants to switch back to drawing mode, a simple double tap on the canvas will do it. If any other tool is selected and a double tap action on the canvas is executed, I chose the pen tool to be the default switch back.

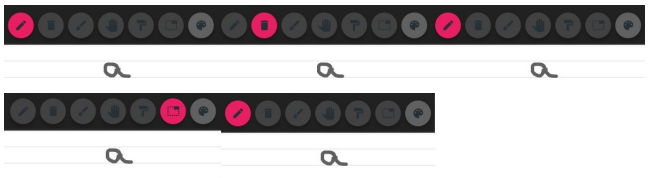

Figure 11. Sequence of actions triggered by double tap on canvas: Expression is written; User double taps on the canvas; User double taps on the canvas again; User switches to select mode; User double taps on the canvas

Similar to what happens on double-tap actions, long-press also has functionality. If the user is using the pen and wants to switch to highlighter mode, a long press on the canvas area will change between the tools and the inverse is also

possible. In the case another tool is selected, the default switchback tool is also the pen. By adding these gestures to switch between tools, it takes from the user the additional effort to press on buttons, making the overall experience a bit more interesting and fluid.

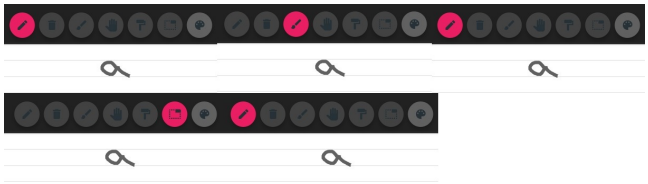

Figure 12. Sequence of actions triggered by long press on canvas: Expression is written; User long presses on the canvas; User long presses on the canvas again; User switches to select mode; User long presses on the canvas

3.2.6 Select. The select functionality is divided in 3 main steps:

- Detect selection

The first thing we have to do is to detect what is being selected. We need to check for collisions between the coordinates that are constantly being pressed against the collection of strokes that are already drawn and stored on the canvas.

After this step, the user has to have the knowledge of what is selected and what is not, so it is necessary to somehow be able to distinguish the selected strokes.

Initially, I changed all the selected strokes' colors to blue, but this solution would be problematic and confusing if the user opted for using another stroke color other than the default grey. As multiple strokes with different colors would be added to the canvas, we would not be able to distinguish the original color until the content was deselected. The solution for this problem was to change the strokes color opacity so we could be able to maintain the original colors while selecting and give the user a more clear perception of his content.

Figure 13. On the left: drawn expressions on the canvas; on the right: first expression is selected

Because the original strokes were already drawn and stored, we could not change their appearance or modify them in any sense. The solution was to access the array where strokes were being stored, erase those strokes and replace them for new strokes with the same characteristics but with lower opacity values. This was only possible because we were using previously drawn strokes and only changing color values and not stroke points. If we wanted to change the stroke's appearance we would have to redraw it.

- Isolate the selected content

Now that the selected strokes are already known, I decided to make a rectangle around what is selected to better isolate the content from the rest. This rectangle is updated in real-time (redrawn with bigger dimensions), as the user is selecting new content. I also added 5 key points to make the presentation more aesthetic (top left, top right, bottom left, bottom right and middle).

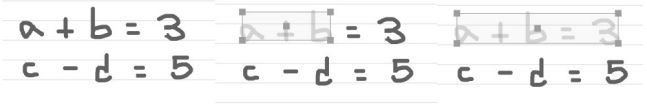

Figure 14. Sequence of an expression being selected

- Drag and drop content

The solution I came with for dragging the selected content was to include a semi-transparent container over the rectangle area so I could wrap it around a Draggable widget and be able to drag and drop it. In this case, as the strokes are already drawn in the canvas, it is possible for deleting and replacing the strokes for new ones with new offsets.

To be able to select some content, the user presses the Select button and starts to press on the screen, just like he was drawing. The difference is that no stroke is shown and wherever the user presses the canvas, if it encounters a stroke, it will be selected. If the user made a bad selection or wants to select new content, he just needs to press the canvas outside the selected area and a new selecting will begin. It is also possible to restart the selection by pressing the selection button. If the content is selected and the user wants to delete it without having to manually delete every stroke, it is possible to do that by just simply dragging the content to the outside of the canvas area. If the user presses any functionality button it deselects the content. If the user no longer wants to use the select and wants to switch back to writing, the gestures on double tap and long press for returning to the pen tool also work.

The select functionality is described below in a sequence of prints. Firstly, the user presses the select button. As he no longer wants the last expression, he selects and drags it to the outside of the canvas so he could easily delete it. The expression is immediately deleted as the user lets go of the screen. The user makes another selection and wants to keep a part of it. He drags the expression to the edge of the canvas as he knows everything that is outside the canvas is deleted. With the new selected expression, the user correctly places it where he wants and finally presses outside the selected area to deselect.

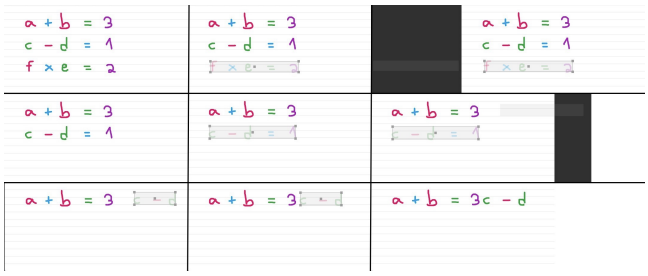

Figure 15. Sequence of the select tool being used: User presses the select button; selects expression; drags the expression to outside the canvas; makes new selection; drags the expression to the edge of the canvas; moves the selected content; clicks outisde the selected area

3.3 Problems while developing the application

After the selection was implemented, the idea was to allow for copy pasting the selected content. By simply double pressing what was selected, the expression would immediately be copied to a new location below. That old expression would be deselected and the new expression selected in its place, so the user could now move the copied content. Here is the first prototype of this functionality. 16.

Figure 16. Selected expression; User double clicks on the selected expression

In order to be able to understand why I did not went further with this functionality and the development of the structure editing part of the application, its necessary to understand a bit more about Flutter and the overall project structure. That is why, in this part, I will be a bit more technical.

In the PointerListener class, the body of our app, is where we handle all that happens on the canvas. The Canvas page is where we define what to do with what just happened.

For example, for a stroke to be saved: as soon as the user presses the canvas, PointerListener has a method that immediately starts to track every point the user has pressed and when the user releases the pointer, it saves all those points in a XppStroke structure. It sends that structure to CanvasPage to be stored in a collection of strokes and, at the same time, draws those strokes in the canvas through a CustomPaint widget.

Initially, while trying to do the copy/paste functionality, my idea was to go to the structure where strokes where being saved, duplicate the selected strokes and change their offset as described in 16. Because the strokes have to be drawn before being able to be displayed on the canvas, I ended up with what is shown in the picture below 17.

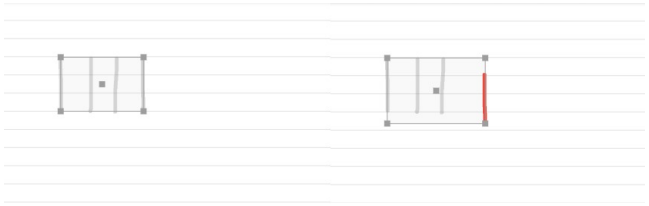

Figure 17. Problem with duplicating the strokes without drawing them

Although the strokes were being added to the collection of strokes, they were not being added to the canvas, so the old strokes would be replaced by the new ones at the canvas. In the image 17 we can see the last stroke being redrawn at the new location (in red). To solve this problem, I add to somehow be able to draw a stroke and save it with its own structure. I used the selected strokes's attributes (tool, points and color) to paint the new strokes, and that is what is shown in figure 16. The problem at this step is how the code is arranged inside the PointerListener. Because what is been drawn at the canvas is inside a Stack of widgets (so we can overlap drawings), this stack only accepts widgets inside, so we cannot call functions or any other methods. As the directly drawn strokes at the canvas were being saved before they were painted (through onPointerMove method used by PointetListener as explained before), they just add to be paited inside this stack at this step. Our new replicated strokes, on the other hand, can not use this method because they are really never manually drawn, they are built through code. This being said, I could not tell the interface to build a XppStroke structure with these newly drawn strokes neither to store them in the collection of strokes. I stopped here the development of the project because, without being able to duplicate content and classify it as an unique stroke/character, the recognition and the structure manipulation would not be possible.

4 Evaluation

The purpose of evaluating the system is to ascertain whether the implemented features are usable and if the application fulfills its overall goal, which is to support the presentation and manipulation of mathematical content. It will also reveal eventual bugs that the application might show. So I could acquire feedback about whether the project is usable or not, 10 users interacted directly with the system through 2 phases: in the first phase, I scripted a series of actions that the user add to complete; in the second phase, I wanted the user to

move freely over the interface. I offer the testers a one-page sheet explaining the interface and the action I wanted them to perform .

I decided not to do an extensive questionnaire with questions like "totally agree" or "totally disagree" after the usability test and, instead, asked users to have a small conversation with me in the end where they would give some overall feedback of the experience and some suggestions of how the interface could be improved. I asked the users to think out loud during the experience so I could understand what was going through their minds and took notes so I could better analyze the experience later. Through observation of the users' actions and thoughts, and by communication, I collected a detailed evaluation of the system. This way I did not overload the user with the experience and generated very interesting data regarding the usability of the system.

After I got the approval for starting the user test, I handed the sheet of paper mentioned in **??**. After the users read the script, I explained to them how the interface worked and how each feature should be used. I left the select and the gestures unexplained. I then started with testing the interface and got the following analysis and results.

4.1 Phase 1: Users perform a script of actions

At steps 1, 2, and 3, there is not much to say because no problems were found. I observed that most users were still getting used to the interface so they took a bit to select the tool I was asking them to select. This changed after the users finished the scripted part of the user testing and explored the app by themselves. Some users asked if they could use the eraser at step 1 because they wanted to rewrite the expression. As the important here is to evaluate the performance and usability of each tool, I asked them to follow the script because the aspect of the content was not important.

At step 4, some users had problems with the originally implemented eraser, because it was not deleting the way they wanted (this problem was already addressed in Section 3).

At step 5, some users questioned why the redo was being used for adding back the deleted strokes and suggested it should be the undo to revert the last actions. I noticed, in this step, that there were some problems restoring previously partially deleted strokes using the original eraser tool. This might have happened because the removal of stroke points was not being done instantaneously but instead through a series of small steps. In some cases, users had to redo around 10 times to restore a small portion of the removed part of the stroke.

At step 6, although all the testers manage to change the width and the color of the stroke with no problem whatsoever, some of them forgot they had the eraser tool selected and had to change it back to the pen tool. Step 7 was also done with ease by all the users as the tool was kind of self-explanatory.

At step 8, I wanted to see how the user would try to select content by them. The majority of the users were using lasso and rectangle selection and the rest was using a selection where they just had to click on the strokes they wanted to be selected. Users that tried to use the lasso selection couldn't select anything because the pointer didn't hover any content. Users that tried to press directly on the strokes they wanted to be selected found it strange why the content was not being selected but the animation was showing. People that were used to using rectangle selection immediately discovered how to select, as the initial part in this kind of selection is similar. At this step, I explained to everyone how to correctly use this functionality.

Some users commented on the fact that the content was not moving along the container while being moved. Some people tried to make a new selection while one was still on the screen and got confused why they could not select more content. Users commented that sometimes they were not able to perceive where the box was while being dragged.

At step 9, after the whiteout was executed, most users tried to use the undo button instead of using the whiteout button to undo the whiteout eraser. Maybe this was due to the name "undo" I used to describe this step. After I explained again how to revert this action properly they suggested one more time that undo should undo the last action and redo remake the last action.

4.2 Phase 2: Users move freely over the interface

And now, at step 10, I told the users to freely explore the app as they pleased and in the end to give me small feedback regarding the overall usability of the app. Before letting them explore the interface, I told them about how the "on double-tap" and "on long-press" would allow them for changing between tools and that they could erase selected content by simply throwing it to the outside of the canvas.

Because I had just explained how the gestures worked, all the users decided to test out this new feature. At this step, only one user remembered to change back to the pen tool to perform gesture events. Because I linked the rest of the buttons to these gestures, this was not a problem. All the users found the "on double-tap" and "on long-press" gestures very helpful, as they would not need to manually press buttons to change between tools. At this step, all the users were able to use the select functionality for themselves and to test how

content could be deleted while selected.

They mentioned again the fact that strokes were not moving alongside the container while being dragged. I could not solve this problem as the widget I was using for dragging content around only allowed me to access the pointer's offset after the drag was ended. The interface would also bug sometimes while trying to select the content that was previously partially deleted and added again (using the original eraser tool).

After some usages, users found interesting the way the select functionality was implemented, as they could have a more precise notion of what was being selected and make more customized and precise decisions, and rapidly adapted.

4.3 Usability testing conclusions

In the end, all the users complemented the overall experience with the interface. They also mentioned some usability issues (most of which I had already noted while they were testing the application) and made really strong suggestions on how the interface could be improved.

The fact that I introduced gestures to trigger action was highly praised. Not only for being able to change between tools faster but also the way the selected content could be deleted. Users said they would like to see this implemented in a more advanced and robust interface.

Users suggested:

- PDF and images should be possible to incorporate
- we should be able to shrink/expand selected content by pressing the edges
- be able to zoom in/out using a pinch gesture with 2 fingers
- select mode being automatically turned on when long pressing strokes
- being able to print screen by swiping up with 3 fingers
- some additional selection modes should be incorporated
- undo/redo buttons should be used to undo/redo actions and not be related with the strokes directly
- each stroke should have its own color and width
- detect different levels of pressure while drawing
- add text boxes
- allow for copy pasting selected expressions

The final conclusion with this evaluation is that, although with some flaws, the implemented features are usable and the application fulfills its overall goal. This whiteboard application revealed itself suitable for inserting and manipulating mathematical content and to be a good starting point for being able to go through with the development of the first structure editor interface supported in multiple devices.

5 Conclusions

The idea was always to be focused on pen-based devices and on how digital ink could be a valuable asset to the educational area. Because I was extremely interested in learning about mobile development, when professor João Ferreira introduced me to the Xournal++ Mobile for the first time, I did not have much to think about before deciding to use it as my starting point. Besides a well-designed interface, a pen and an eraser already implemented, it was built using Flutter. That was all I needed. Flutter lets us program in a single code base (in dart) and auto-generate applications for iOS, Android, Windows, Mac, Linux, and even for the web. This is truly interesting to think from a programmer's point of view because we do not have to change a single line of code for the interface to run on different devices and operating systems. Although Flutter itself has only 4 years of existence, I decided to learn a new language and embrace the opportunity I was being given.

As Flutter is relatively new, is deductible that there is still not much information online, at least compared to Python or Java. Because, in our case, the subject is even more specific, the amount of information we can gather and the help we can get is even more reduced. This was my main drawback while developing the project. I could not manage to find much help from online searching as there was not enough development in this subject. Making a whiteboard application on Flutter is way more specific than just building a simple user interface with buttons and menus as a common developer uses it for.

When we look at the flutter whiteboard interfaces I mentioned earlier, we can see there is not much to compare to and to be based on. These interfaces have a pen, an eraser, and that is it. If we compare those interfaces to the Xournal++ Mobile I started with, we can see how superior this application is at every level. Although the tools for on canvas actions are basically the same (a pen, an eraser, and a color change), this interface is much more appealing design and structure-wise and has already tools implemented for changing the background, for adding new pages, and for saving the document. It looked like a really promising application. Unfortunately, they stopped updating it as soon as I started developing it, so they must have come to the conclusion it was no longer worth it to keep updating.

While making changes to the application and adding new features, I realized I was not able to copy and paste previously drawn strokes because of the way the code was organized. This was already explained in . This being said, I could not go further with the structure editing part of this thesis original idea because it was simply not possible to manipulate and store strokes created via code, they had to be physically drawn in order to be saved and handled.

5.1 Achievements

By using Xournal++ Mobile as a starting point, I have developed the most advanced whiteboard application built with Flutter until today (to my knowledge). In addition to the design and functional changes I made to the interface itself, I created a new eraser, a highlighter, a whiteout eraser, undo and redo buttons, a select functionality, and added some action triggered with the usage of gestures. I can say I was a pioneer when it comes to whiteboard applications built-in Flutter and that my contributions are remarkable since there is very little (to none) exploration in the field (once more, to my knowledge). The work that I have done here brings new contributions to the state of the art regarding whiteboard applications built-in Flutter and gives us an insight into what kind of obstacles we might face when trying to explore this toolkit and more particularly, the CustomPaint widget.

5.2 Future Work

Firstly, I would like to make changes to the interface based on the user testing results in order to improve the overall usability of the system. Would be interesting to integrate pdf and image files, add text boxes and make the interface rely way more on gestures like the ones described at the end of the evaluation. If I somehow managed to overcome what made me stop the development of the application in the first place, the development of the project would go to a whole new level with being able to manipulate strokes.

I would start the implementation where I left, this being with the copy-pasting of selected expressions. With the integration of a handwritten mathematics recognizer, the points that are being pressed on the screen could automatically be subjected to the recognizer while being drawn and as soon as the user finishes the action, we would store the recognized characters instead of the strokes. Another possible approach could be to use the recognizer on selected content and then proceed to store strokes as characters in another structure. Because every individual has his own handwriting, it would be interesting to have a model for handwritten characters which was updated as the user draws new content. Just this feature alone would be a lot to think about mainly because recognition systems normally output multiple results with different percentages of being correct and we had to be able to select or to give the user the decision on which recognition was correct. We also have to consider the fact the system might give us an incorrect output and the user would have to be able to change that. We also would have to think of a way for the user to take advantage of the recognition without overloading the system.

References

[1] Richard Anderson, Ruth Anderson, Peter Davis, Natalie Linnell, Craig Prince, Valentin Razmov, and Fred Videon. 2007. Classroom presenter: Enhancing interactive education with digital ink. *Computer* 40, 9 (2007), 56–61.

[2] Richard J Anderson, Crystal Hoyer, Steven A Wolfman, and Ruth Anderson. 2004. A study of digital ink in lecture presentation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 567–574.

[3] Lisa Anthony, Jie Yang, and Kenneth R Koedinger. 2005. Evaluation of multimodal input for entering mathematical equations on the computer. In *CHI'05 Extended Abstracts on Human Factors in Computing Systems*. 1184–1187.

[4] Lisa Anthony, Jie Yang, and Kenneth R Koedinger. 2005. Evaluation of multimodal input for entering mathematical equations on the computer. In *CHI'05 Extended Abstracts on Human Factors in Computing Systems*. 1184–1187.

[5] Robert A Bartsch and Kristi M Cobern. 2003. Effectiveness of PowerPoint presentations in lectures. *Computers & education* 41, 1 (2003), 77–86.

[6] Mordechai Ben-Ari. 1998. Constructivism in computer science education. *Acm sigcse bulletin* 30, 1 (1998), 257–261.

[7] Charles C Bonwell and James A Eison. 1991. *Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports*. ERIC.

[8] Salman Cheema and Joseph J LaViola. 2010. Applying mathematical sketching to sketch-based physics tutoring software. In *International Symposium on Smart Graphics*. Springer, 13–24.

[9] Jamie Cromack. 2008. Technology and learning-centered education: Research-based support for how the tablet PC embodies the Seven Principles of Good Practice in Undergraduate Education. In *2008 38th Annual Frontiers in Education Conference*. IEEE, T2A–1.

[10] Thomas J Fitzgerald. 2004. The Tablet PC takes its place in the classroom. *The New York Times* 9 (2004).

[11] Md Athar Imtiaz, Rachel Blagojevic, Andrew Luxton-Reilly, and Beryl Plimmer. 2017. A survey of intelligent digital ink tools use in STEM education. In *Proceedings of the Australasian Computer Science Week Multiconference*. 1–8.

[12] Ferman Konukman, Erik Rabinowitz, Michael W Kernodle, and Robert N McKethan. 2010. The effective use of PowerPoint to facilitate active learning. *Journal of Physical Education, Recreation & Dance* 81, 5 (2010), 12–16.

[13] Alexandra Mendes, Roland Backhouse, and Joao F Ferreira. 2014. Structure editing of handwritten mathematics: Improving the computer support for the calculational method. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. 139–148.

[14] Jane Mills, Ann Bonner, and Karen Francis. 2006. The development of constructivist grounded theory. *International journal of qualitative methods* 5, 1 (2006), 25–35.

[15] Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. 2016. *Designing the user interface: strategies for effective human-computer interaction*. Pearson.

[16] Beth Simon, Ruth Anderson, Crystal Hoyer, and Jonathan Su. 2004. Preliminary experiences with a tablet PC based system to support active learning in computer science courses. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. 213–217.

[17] Tran Diem Trang. 2015. Using ppt in the ESL classroom: Benefits and drawbacks from high school students' perspectives. *Transforming English Language Education in the Era of Globalization* (2015), 301–309.