Keyboard maker project

Final Report

Nicolas, Antonin, Eva et Leïla

1. Introduction

> **Disclaimer**
>
> Our tool has been created for a niche community: Keyboard hobbyists.
>
> Before we dive into the user requirements and the design, let's take a trip in the world of custom keyboards.

1.1. Custom keyboards

For a specific category of people, classical keyboards suck. The keyboards that are included in your laptop are inefficient, not ergonomic, and unpleasant to use. For some of these people, changing the layout (The mapping from keys to symbols) is enough[1]. But for others, the only way to fix the problems of current keyboards is to buy a custom one.

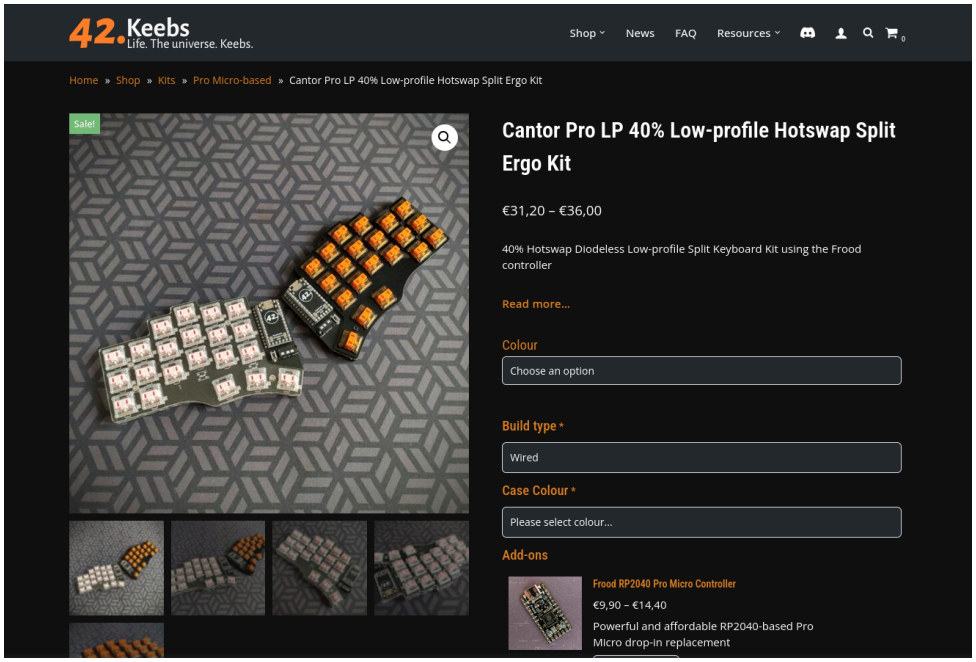


Figure 1: A keyboard kit in a specialized online shop

Buying, doing the setup and using such a keyboard is a time-consuming investment. We will refer to such people as *Keyboard hobbyists. Keyboard hobbyists* form a supportive community, and they are mainly active on discord, github[2] and Reddit[3].

[1] A great overview for french layouts: https://ergol.org/alternatives
[2] https://github.com/help-14/mechanical-keyboard?tab=readme-ov-file#tutorials
[3] https://www.reddit.com/r/MechanicalKeyboards/wiki/index/

Very common activities for keyboard hobbyists include:

- chosing a kit to buy
- soldering the components of the keyboard
- thinking about the ideal layout for the keyboard[4]
- configuring the keyboard using special firmware[5]

It is also quite common for keyboard hobbyist to try to make their own custom mechanical keyboard. It is of course a lot more time consuming, but they are resources online to get started.

For some of these activities, specialized tools exist (see Section 2.2.1).

However, we identified a lack of a unified keyboard creation app. Such an app would allow the maker to create its keyboard geometry, to propose a custom layout. It would also allow the final user to import the keyboard, change the layout, and export a firmware configuration from it.

> **A brainstorming app**
>
> The main advantage of our app is to **materialize** the idea of the keyboard maker. By forcing him to express his keyboard idea in a concrete way, it will help him design the keyboard faster. It can also be used for collaborative keyboard design, and for teaching the fundamental of how custom keyboard work.

This is why we chose to create it: a **Unified keyboard maker app**.

The first step is to have a deeper understanding of the user, before collecting our requirements.

[4] There is a lot of theory in this domain: https://workmanlayout.org/#back-to-the-drawing-board
[5] https://docs.qmk.fm/

2. The User

2.1. Identifying the user needs

2.2. Existing tools

2.2.1. Keyboard Layout Editor (KLE)

2.2.2. Reddit : Survey on desired features

To identify the user needs, we decided to directly ask people that fitted the profiles of potential users : people that want to customize their keyboards, wether it is for gaming, as a hobby, professionnally or because of an impairment. To start a discussion with them, we posted on subreddits focalised on this activity and mechanical keyboards, ie r/MechanicalKeyboards and r/olkb. We chose this subreddits because they were focused on keyboards discussions, and since keyboard customization is quite niche, we couldn't just ask the general population as they would probably not be the main users of our site. Therefore, we went to ask the people that would most likely use it, and posting on public forums can :

- gather a lot of input since there are a lot of people on these subreddits. r/MechanicalKeyboards has 1.3 Million users and r/olkb has 62k users.
- The posts stay on for a long time so we could gather more input has time passes if people fall on our post
- makes reaching out to niche communities easier

The post we made on r/olkb we got 4 users input, and the redditors gave us examples and ressources. The main points that the user desired were :

- Keyboard Layout Editor(KLE) was considered pretty much perfect by an editor as it allowed to change the style, colors, etc. One problem for this user was that it wasn't easy to use on mobile, and it doesn't provide key statistics per language, so this would set our tool appart. Here are some examples of what can be achieved using this site : 1, 2, 3, 4. Another user however pointed out that this tool doesn't handle curves or non-standard keys, as can be seen on this example when attempting to reproduce a MS Natural Original Keyboard. Another issue pointed out with this tool is that KLE uses two versions of JSON, one of which is non standard which can cause issues with 3rd party tools. Also, there is no snap to grid functionnality allowing the user to automaticalluy align the keys together. Finally, the image exports is considered not great, developpers were working on an SVG exports that was never finished.
- Integrating Swill's laser cutouts Swill is a web-based tool that generates SVG files for laser cutting keyboards plates and cases.
- Integrating KLE-render : a tool that that a KLE layout and generates a 3D rendering of the keyboard.
- Using OpenSCAD for the mechanical design : hard to use for basic tasks but there are Python extensions to make it easier to use, but not as easy as SolidWorks. The user mentionning this tools would like a computer-aided design tool for Linux that is visual.
- One user mentioned computeritis (an ailment caused by the use of computers (pain in some fingers, etc)) as a cause for wanting to personnalise their keyboard. This user writes mostly using dictation mode and voice commands but for corrections, some keys (like backspace, etc) are still needed. This user uses a lot of chords (pressing several keys to do an action), this increases the number of actions they can do with a small number of keys. They often need to reprogramm their keyboard for specific usage, and a visual tool would be practical since it would be easier to remember and use as customized keyboards don't have the characters sent to the computers

printed on them by default. This user also indicated that some combination of keys were easier for them to use relating to their placement and the Repetitive Strain Injury (or computeritis).

For the post we made on r/MechanicalKeyboards, we got one response :

- The user desires a more modern Keyboard Layout editor and a software that creates PCB(printed Circuit Board) according to the custom layout which would save time from having to do it in KiCad or another similar software. For them, providing statistic on keys used per language would not be really pertinent for their use.

We didn't get many answers on reddit, but this could be because of how the site functions. If people agree with what has been said, they won't duplicate it by giving another answer. Therefore, this could explain why we only got five responses, some of which were really developped on the struggles the users face.

To gather basic requirements, we focused on the tools that the redditors referred to us as useful and practical.

2.2.3. Requirements gathered from existing tools

The users should be able to :

- Place their keys as they want and change their format/dimensions
- Change the colors of the keys to make them more distinguishable from one another
- Label the keys as they wish, including special character such as an arrow, the label for the enter key, etc.
- be able to customize different layouts, ie, by clicking on a special key such as shoft, CTRL, etc, the user gains access to other characters than when not pressing it. This is the case on most keyboards, the most basic example is that when pressing simply the "e" key you get a lowercase "e" but when pressing shift then "e" you get an uppercase "e".
- downloading a custom design that can be reimported into the website
- The possibility to download a visual representetion of the keyboard
- A documentation on how to use the tool
- Presets of keyboards

2.3. Persona

Now, let's focus on the identified personas that we can roughly divide in two types : makers and hobbyists. The needs are different between all the users so the next section will allow us to explain more precisely the specificities and scenarios for each type.

2.3.1. Maker

,

The makers know the basics of keyboard protocols and have more professional needs. There are high chances that they have already used similar softwares so they also have a good idea about how this type of software works. As the makers are our principal users, this explains why there will not be a full tutorial for our software. However, we need to implement helping features especially since they might be used to previous softwares that will not function exactly like ours.

2.3.2. General Hobbyist

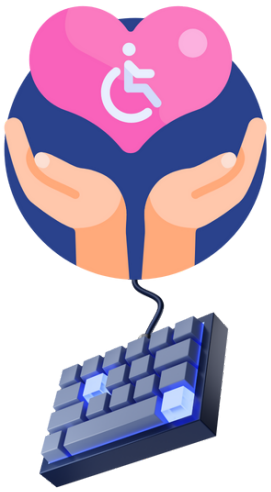
,

These users may custom their keyboards for the first time. As they are new to this task, we propose templates for layout. With these and the tips feature, the hobbyists can understand better the general process of designing a keyboard. A documentation will also be available to help these users get used to the software by describing the possible actions they can do and the specificities of possible actions (for instance : how to add a specific layout for when a modifier is pressed).

2.3.3. Gamer Hobbyist

,

2.3.4. Hobbyist with disability (extension)



,

Gamers continually use their keyboard and, compared to the previous hobbyists, have a general idea of what type of keyboard they would like to work with. Comfort, speed and easy key combinations are often the properties gamers look for. They will want to try different layouts to evaluate for themselves how much the layouts are usable and efficient for video games. This is why we will implement an emulator of the layouts created by the users, to allow them to realize the disposition they created and make changes according to the results.

Some disabilities can limit hand movement and dexterity. Therefore, these could be users of our tool to design a keyboard that would be adapted to their specific needs. To adapt our design to these users who might not know much about keyboard layouts and customization, the most important part would be to make our website controller compatible and offer predesigned configurations to start from a certain basis. A lot of controllers have been created to be adapted to different handicaps, such as the x-box adaptive controller which allows users to create their own controller and therefore adapt it to their impairments. This compatibility would be an extension we add to our website if e had enough time, but in any case, our tool would allow the creation of more practical keyboards for people with a handicap, whether the keyboard is created by the handicapped person or someone else. The predesigned configurations would also be an extension if the time allows it. We are planning on predesigning basic configurations for a predefined number of keys (such as azerty, qwerty, left-handed adapted keyboard, etc).

One paper [1] worked on adaptative keyboards for people with cerebral palsy to show that adapted keyboards can facilitate the interactions of people atteigned with this degenerative disorder with computers or phones.

2.4. Initial Requirements

3. Evaluation methodology

3.1. Focus on each personna

To test the usability, we'll ask several people to implement the same keyboard, the complexity and layout of which will difer according to the persona for which we are testing the usability.

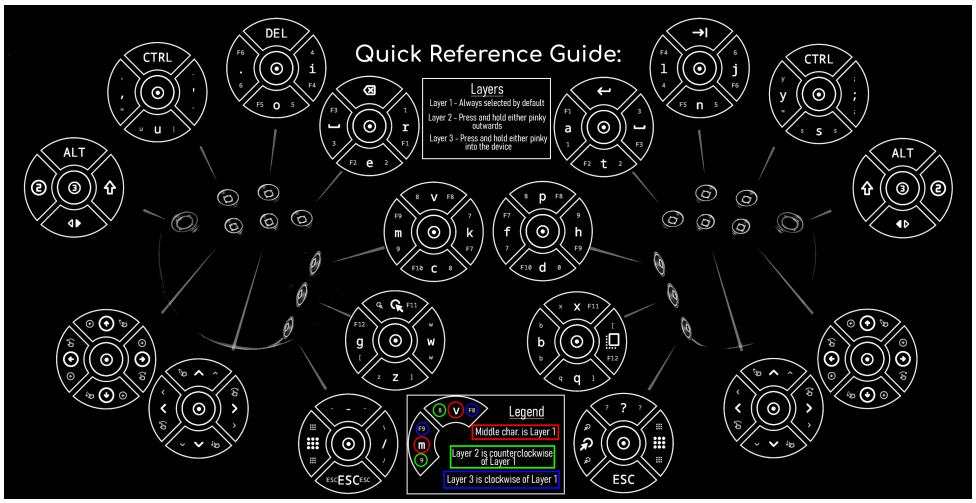
3.1.1. Maker

No idea, maybe a random of the three other would not be absurd.

3.1.2. General Hobbyist

For this profile, we chose to focus the test on keyboard efficient for dactylography.

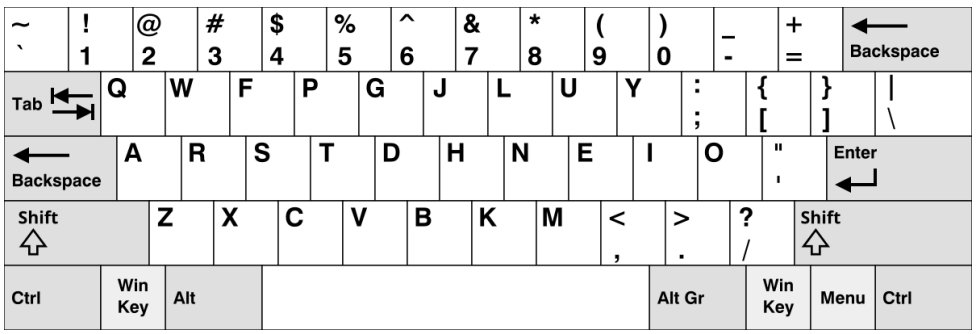
To this end, we'll ask a student to configure a part of the characorder :



3.1.3. Gamer hobbyist

For this profile, we chose to test a custom gamer keyboard layout, that will be fixated on the closeness of certain keys that help with speed and prevent the player from losing time moving their hands to much to do an action.

We have found that a gaming layout recommended is the colemak layout. Therefore, we asked our testers to implement it :

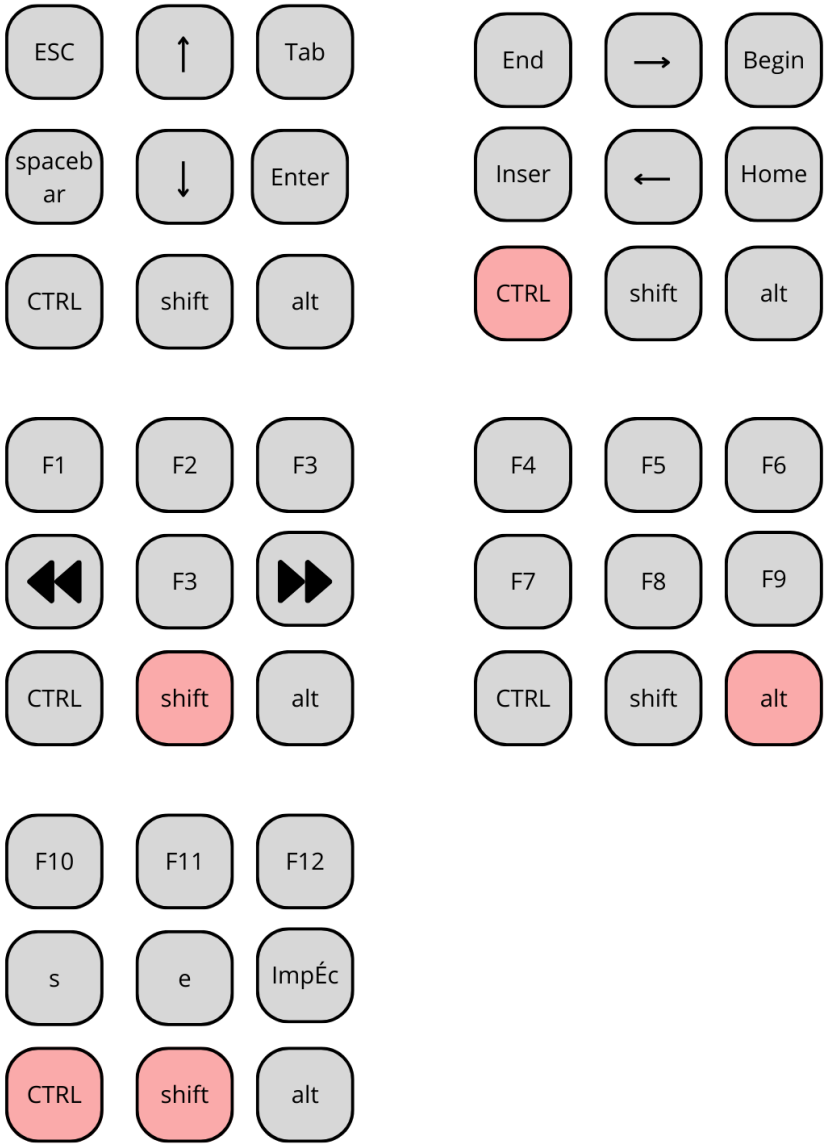


3.1.4. Hobbyist with a disability

For this persona, we will test the following keyboard design. We chose this design because with disability, people might have trouble accessing a lot of keys, so we choose a keyboard with a small number of keys but a lot of layers.

People with disabilities might not be using letters and using voice dictation as one reddit user mentionned. However, they need the keyboard to allow them to move or correct the text and do text

formatting in in limited number of keys. According to [WEBAIM], you'll find in this link the most common interactions with non characters keys and therefore the most important to configure on this test keyboard.



3.2. Methodology and setting

To measure the usability and success criteria, we will proceed in two phases :

- The first one will be during the use of the software : we will invite the participant to express all feelings and impressions they have toward the tool, wether for instance something is efficient, intuitive or not, if they have trouble finding a certain tool etc. During this time, we'll also evaluate the success criteria. According to each layout we will ask them to do, there will be different criteria as described before (to do), and we will evaluate them based on if the user successed in doing the task or not.
- The second part will be a post usage evaluation. We will ask the questions given before according to what layout we asked them to implement (to do according to the layout but basically if the tools were easy to use, if they found them easily etc)

3.3. Evaluation Results

3.4. New requirements

4. The design

4.1. Paper prototypes

To choose our design, we first discussed what were in our opinion the most important aspects of our software with respect to the opinions we gathered during the requirements gathering, what would be the essential tools and how to implement them. After discussing these points, we each drew our first sketches to compare our visions of the software after agreeing on these points.

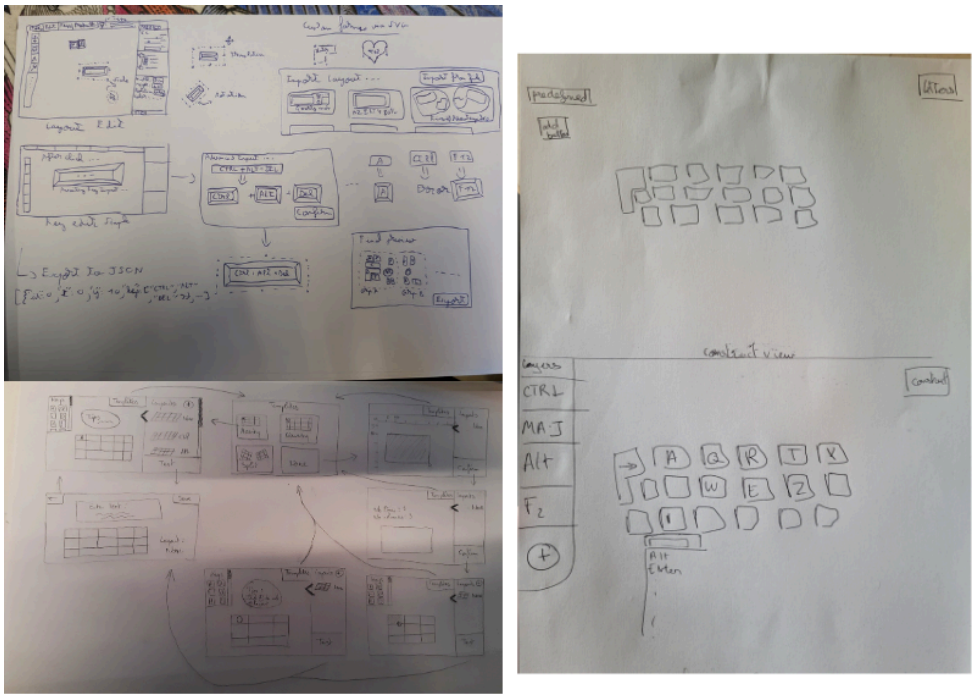


Figure 5: Our individual sketch prototypes

After discussing the results of the sketches, our sources of inspiration and analysis of the user flow, we agreed on several points :

Realistic representation We wanted the user to be able to visualize their creation layer by layer, by opposition to current keyboards that allow either visualization of each key by clicking on a key or like real keyboards by putting several characters on a key touch. To select the layer, the user will click on the bottom left. This configuration allows the user to visualize clearly to which keyboard they gain access to by clicking on which modifier.

Recognition over recall To configure the keycode associated to a key, the user will double click on it, which is standard practice for interacting with an object in computer software.

Emphasis on personalization The user will be able to modify the sizes of the keys, their rotations etc on a side menu to the right and as an extension might also be able to import keys with a custom shape.

Important actions on the top A menu would be added to the top. In this menu, the user would be able to add a key, move a key(s), import a JSON configuration file, export the current representation of the current design and export the current design as an svg file.

Feedback and Affordance through a set of comprehensive tools The different actions available at one time on the main view of the software should be represented by a set of tools visible at any time. These tools should visibly imply the actions available to the user as well as the one they currently have selected.

(Extension) Allow for testing Create an emulator allowing the user to try out their design.

Non-intrusive guide Create a tutorial that the user can skip but that automatically appears on the screen telling them how to place a key, modify it and move it.

4.2. Final paper prototype and thoughts on implementation

The final paper design we came up with after the final discussions is the following :

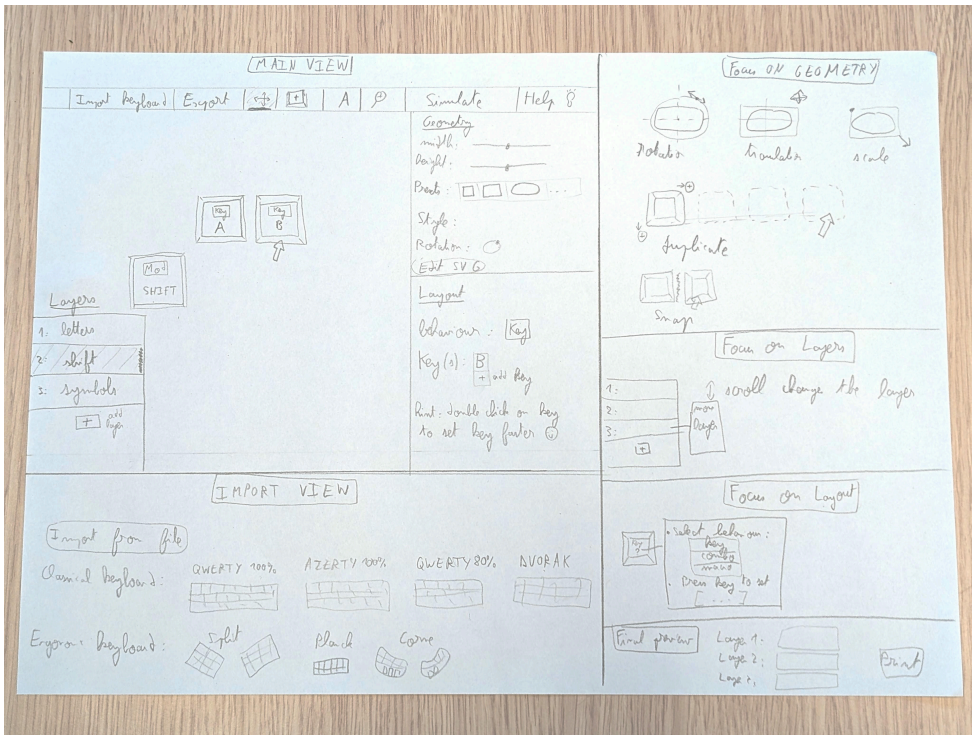


Figure 6: The final paper prototype

This design is greatly inspired by the image editing software like Photoshop, Photopea, etc. The goal was to have a main window on which most of the interactions with the software would happen. Some more complicated operations, like importing and exporting, would require popup menus, of which there would never be more than one at a time.

We chose at this time that our app would be a web-based application, which would allow the greatest amount of people to experience the software, would it be for testing or actual use. The design would thus be realized using HTML and CSS.

The software would incorporate a 3 depth-level system user CSS's z-index values with the main working layer in the background, the ui on top of it and finally the popups blocking the rest.

The user interface would be divided as follows :

Top side : Main menu It holds the primary functions : Importing, exporting... as well as the main tools and the tips box (which would serve as our non-intrusive tutorial vector) would share the top-menu space.

Right-hand side : Key-menu This is the menu where the user may see and change all the settings of the key they select (key size, rotation, binds...).

Bottom-left : Layers Being of main importance in how the keyboard actually works, we chose to give them their own menu on the bottom right (instead of being part of the right-menu like layers in our examples), which differentiates them from the key-specific settings on the right-hand menu.

The colors used would be a close match with our inspiration tools for the best readability possible.

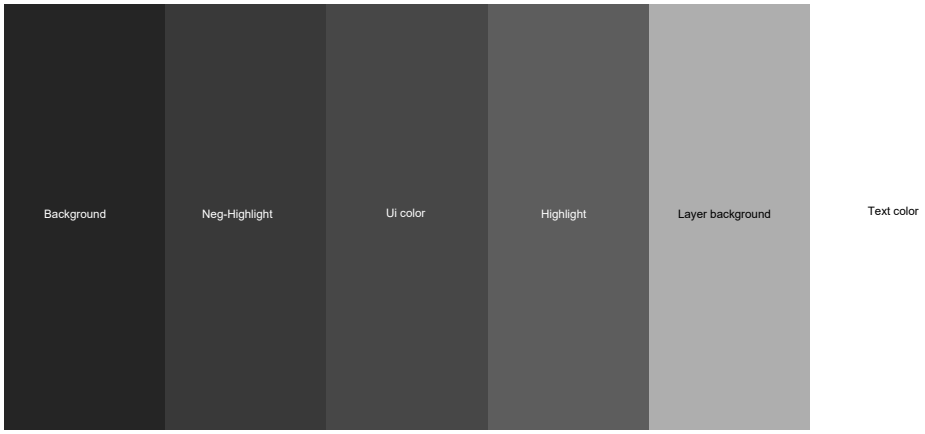


Figure 7: Our main color palette

4.3. Our mascot

At the time we were comparing the sketches on Figma to create our final paper prototype, we also came up with a mascot to fit our visual identity and our purpose. We present to you, Keybby the Keycap :

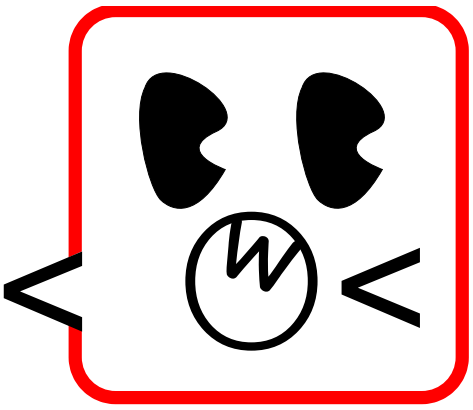


Figure 8: Keybby, the mascot of our software

Keybby is composed of a simple square shap reminding of a keycap ass well as letters to define its features aside from its eyes (O, W and V). Being made from scalable vector graphics, his likeness was able to be reused throughout several graphical items if the tool, as well as reusing its shapes for other items to keep a distinct identity.

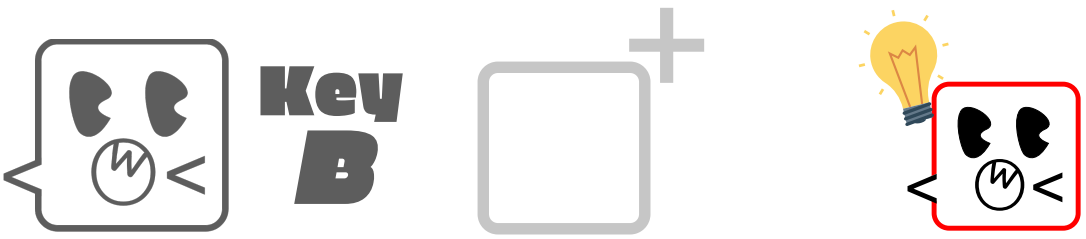


Figure 9: Several assets reusing Keybby's base

All assets on the app are composed of assets either owned by us or in the public domain.

4.4. The implemented design

Here is a vue of the software as seen in the online version :

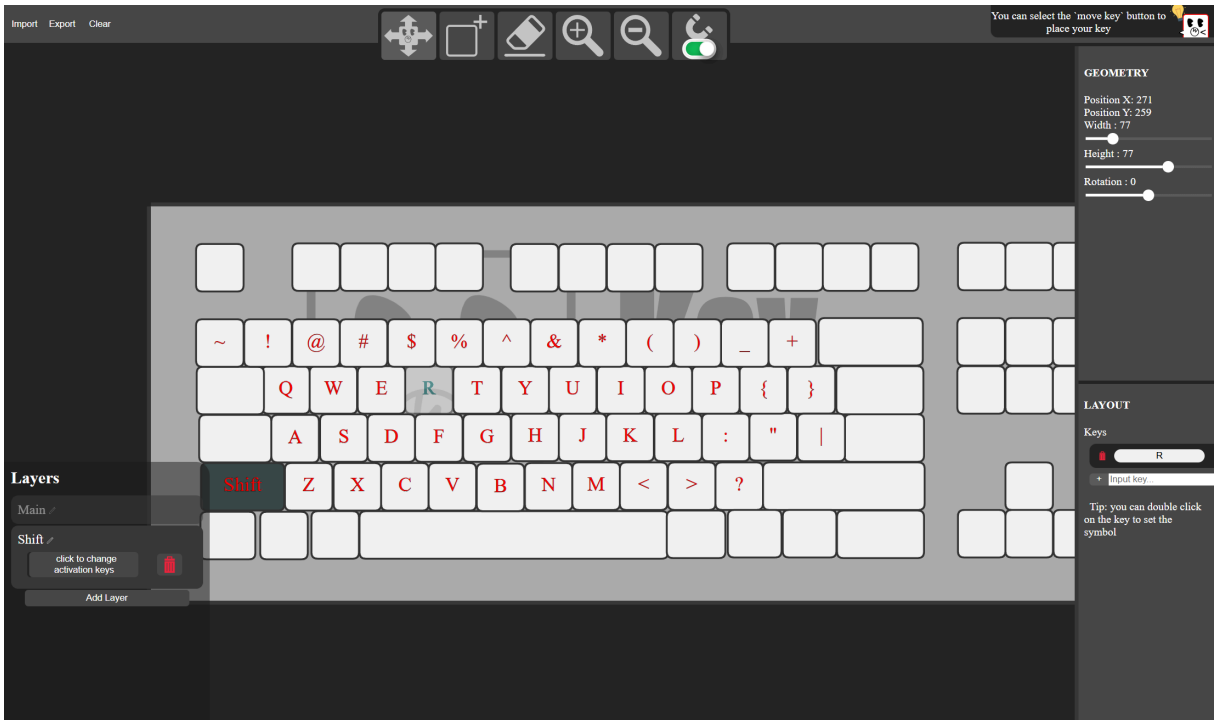


Figure 10: The design of the software, seen in the online version

Here is the final design after the first wave of user testing, which lead to additional indicators for some functions (like the garbage cans and pencil icons) as well as the addition of additional tools and functions like enabling the grid magnetism which we will mention later. The layer on the background can move fully independently from the ui sitting on top. It can be zoomed on and translated at will.

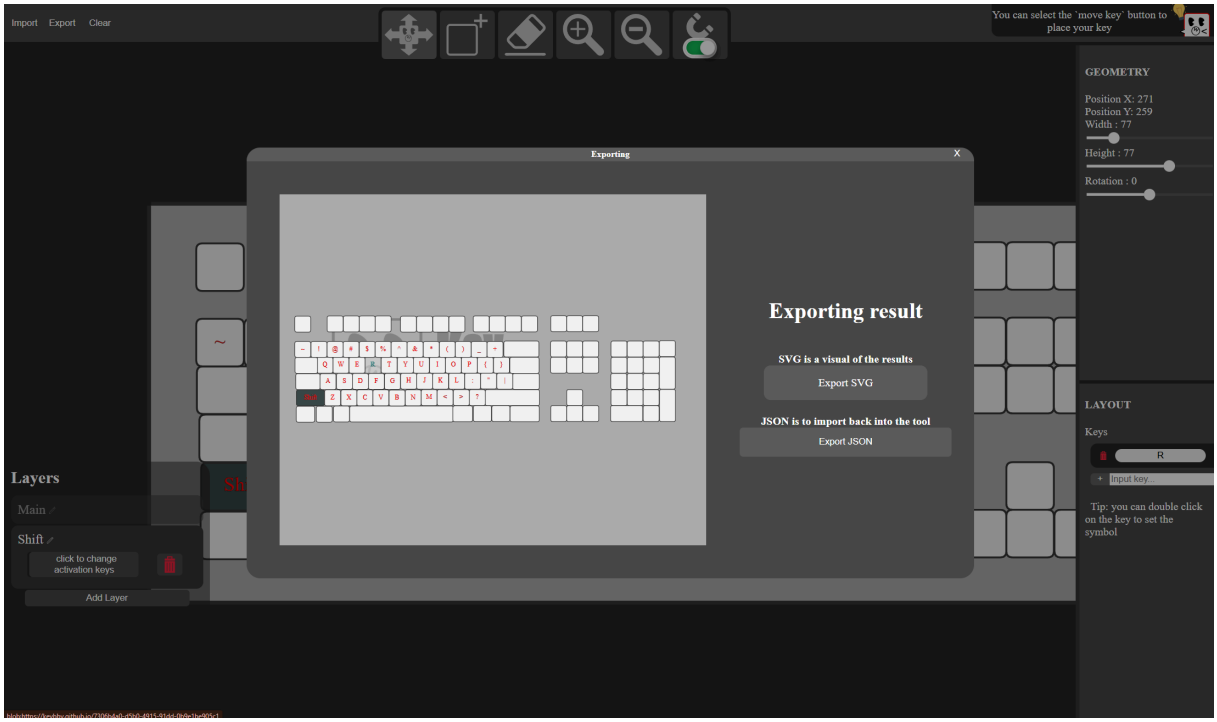


Figure 11: The export popup of the online version

The popups appear above the rest and can be moved freely around, but forbid interaction with the background, signalled by darkening said background when the popup is visible.

Figure 12: Button visual when not selected (left) and hovered/selected(minus the text) (right)

The interface is meant to be reactive and give the maximum amount of feedback to the user on what it is doing and the current state of the program, which shortcomings the user feedback helped us iron out. This works by changing the style of the elements depending on the user interactions like hovering, giving the user secondary visual cues on the state like custom cursors on the canvas depending on their current selected tool (a square with a plus for add, a grabby hand on a key when moving…) or having more direct cues that don't impact the user's flow if they already know the program, like having <*Click Me*> as a default text on a key or with the tips.

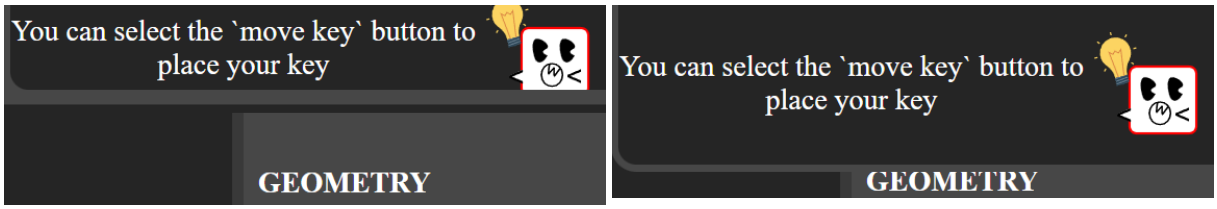


Figure 13: Tips visual when not selected (left) and hovered (right)

The resizable elements, like the canvas and the side bar, are signalled by a small Neg-Highlight colored bar for recognition's sake, to catch the user's eye. They also change the user's cursor on hover to signal their use. Clipping and … for the tips area are implying that the element can be expanded as well.

There remains a learning curve for the program since it is meant to be used by experimented people, but the fourfold increase in speed after the first try by one of our testers allows us to think that the program allows for pretty good learnability after the first couple of key rows.

5. The prototype

Our git repository

As stated in the design part, we chose for our app to be web-based. This means our application would be bound by HTML, CSS and Javascript, atop which we added two additional elements:

- To program more safely, since our app would rely heavily on a lot of different classes across multiple files, we chose to also include Typescript type verification in our workflow.
- To have greater versatility and reduce the complexity of linking code and visuals we also used the Alpine.js framework. which allows for a easy implementation of bindings as well as other behavior directly in the HTML markup.

Here is how the Model Control View diagram of our application looks like :

INSERT DIAGRAM

5.1. KISS

5.2. Feature highlights

5.2.1. Popup system

The popup box we use for Import, Export, Clear and Key input ia always present within the dom but modified and displayed in real time when necessary. In the DOM, it behaves as a grid whose center element is the popup itself. In the backend, the DOM for each popup is stored inside a specific HTML file.

When the popup is called, a switch case determines which popup child class needs to be created and stored. This class calls fetches the correct HTML and the super constructor inserts it into the DOM. It also gives the necessary methods for the popup to behave correctly as well as overrides the basic popup methods (like the done method called when the popup is closed) is necessary.

That way, all popups are as independent with each other without having much duplicate code if any.

When the popup is visible, the background is darkened to imply the fact that it is the only object the user can interact with. The popup is also movable when dragging on the top part of the element where the title resides. It is closed when the interaction is done (with the key input for example), when the user presses the X button or Escape on their keyboard.

Here are a few quirks for the two most unique popups : Export and Key Input :

- Export displays on the right a copy of the user's canvas. This copy is obtained from the DOM and then expunged of every attribute used by Alpine so that it can be reread by the user on their own computer without any error.
- The key input popup registers any last keydown at the key actually pressed by the user and only saves on the last keyup before closing. This allows for key combos like Shift+A to actually register a "A" instead of "a". The operation type between append and replace is also stored in the popup main class directly to keep memory between uses within the same session.

5.2.2. Snap and Collision detection

5.2.3. Tutorial system

6. The evaluation

Bibliography

[1] A. Henzen and P. Nohama, “Adaptable virtual keyboard and mouse for people with special needs,” in *2016 Future Technologies Conference (FTC)*, 2016, pp. 1357–1360.