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 to 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.

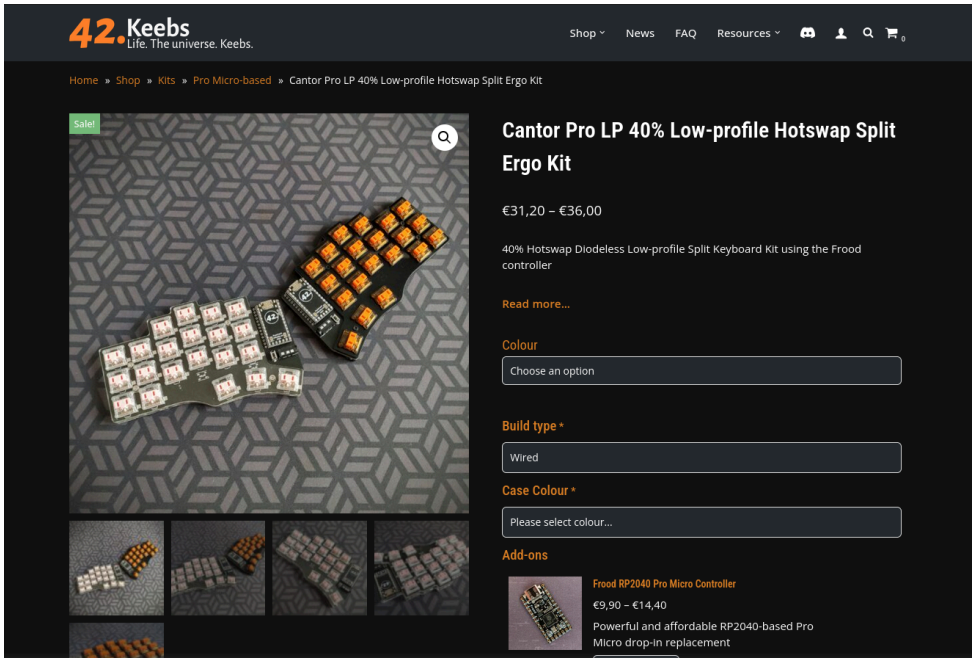


Figure 1: A keyboard kit in a specialized online shop

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:

- Choosing 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 2.1.1).

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

2.1.1. Keyboard Layout Editor (KLE)

This tool is a preexisting tool that we looked at a lot since it is a reference in the keyboard customization community. In our tests, using the tool was quite the struggle and the layout didn't feel very intuitive. For those of us who had never looked at this type of software, we struggled to understand all the options we could use and how to do basic tasks such as : creating a key, inputting the character that should be linked to the key, how to create layers etc. This tool provided a lot of visual options such as modifying the color of a key though.

2.1.2. 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, whether it is for gaming, as a hobby, professionally or because of an impairment. To start a discussion with them, we posted on subreddits focalized on this activity and mechanical keyboards, ie r/MechanicalKeyboards and r/olkb. We chose these 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 chose to make a post on these enthusiast's places because :

- Public forums 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 find our post later on
- It 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 resources. 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 as it is something that we thought about to extend the project. 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 functionality allowing the user to automatically align the keys together. Finally, the image export is considered not great, developers 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 takes 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 mentioning this tool 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 personalize 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 reduced number of keys. They often need to reprogram 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 Developed 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.

2.1.3. Requirements gathered from existing tools

The users should be able to :

- Place their keys as they want and change their format/dimensions
- Be able to 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 shift, 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 "E".
- Be able to download a custom design that can be reimported into the website
- Have the possibility to download a visual representation of the keyboard
- Have access to a documentation on how to use the tool (we planned to do this part but didn't have time to implement it until the deadline except for the non-intrusive tutorial)
- Be able to use presets of keyboards

2.1.4. Debates and simulation

With our group, we tried several existing tools and debated about what aspects of them were intuitive and user-centered or not. We then looked at what we could improve and add in our software.

2.2. 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.

2.2.1. Maker (Mark)

Mark knows the basics of keyboard protocols and has more professional needs. There are high chances that he has 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.

2.2.2. General Hobbyist (Gina)

Gina 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, Gina can understand better the general process of designing a keyboard. A documentation will also be available (in the future if the project continues) 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).

2.2.3. Gamer Hobbyist (Sarah)

Sarah continually uses their keyboard and, compared to Gina, has a general idea of what type of keyboard they would like to work with. Comfort, speed and easy key combinations are the properties Sarah looks for. She will want to try different layouts to evaluate for herself how much the layouts are usable and efficient for video games. This is why we want to 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. this emulator is a goal for the future of our project as we didn't have time to implement it.

2.2.4. Hobbyist with disability (Sam)

Some disabilities can limit hand movement and dexterity. Therefore, Sam could be a user of our tool that designs a keyboard that would be adapted to their specific needs. For Sam, who might not know much about keyboard customization, the most important part is to make our website controller compatible since they are often the preferred control scheme of movement-impaired people, as well as offer predesigned configurations to start from. This compatibility would be an extension we add to our website if we had enough time, but in any case, our tool would allow the creation of more practical keyboards for people with a handicap.

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.

2.3. Initial Requirements we focused on

2.3.1. Accessability

Can user with disabilities use the website easily?

While the extension to add the possibility to use the website with a controller would greatly improve the accessibility of our website, we tried our best to make it accessible. We used high contrast of white to black to have an easily readable and understandable interface.

2.3.2. Efficiency

How quickly can the user perform basic tasks?

To evaluate this, we added a special step in user testing dedicated to this. To maximize the chances that basic tasks could be easily done, we created tools to create a key and move a key with an intuitive design (*ie*, the move key has the aspect of most move key in software). We were also inspired by the layout of Photoshop for our software, which we believed improves the time necessary to get familiar with the layout of the software for a new user.

2.3.3. Customizability

Can the user adapt the interface to their needs?

We created the software so that the canvas's size and the side of the menus can be adapted freely by the user.

2.3.4. Satisfaction

How pleasant is the use of the system?

To make the website as pleasant as possible, we added shortcuts with intuitive hot keys (CTRL+C, CTRL+V, SUPPR). We also created a cute mascot for our project which we believe will get the user more engaged with the product and makes it more pleasant to use. Our mascot is names Keybby, and our website KeyB(we took inspiration from the name of the mascot)

3. Evaluation methodology

3.1. Preface on evaluations

At the beginning of the project, we wanted to make different interpretive evaluations for each persona. We sent a form, using Google Forms, on Reddit (for the maker and with disability personas) and to some friends (for the hobbyist and gamer personas) to know their availability but only five persons answered. There were not enough participants to make evaluations for each persona. This is why we created an evaluation in three steps, the same for every participant.

However, we did take into account that the interface was designed for four different personas.

3.2. Focus on each persona

Our interface is mainly designed for the maker persona so we based our evaluation on that. It does not mean that we did not evaluate aspects more related to other personas.

3.2.1. Maker

As said in the previous part, the redditors that gave us examples of tools (they represent the makers) highlighted the interface's ease of use, the varied customization of the keys and a good visual. We decided that the first step of the evaluation would be the timed copy of a keyboard presented later. This allows us to see how much time is needed to make a simple keyboard with two layers, and which part of the process is the less efficient in terms of choice of interaction.

We had one participant for this persona.

3.2.2. General hobbyist

For this persona, it is often the first time they use an interface to customize a keyboard. Timing the creation of a specific keyboard allows us to know where the interface is not understandable enough. The difficulty to know which action is doable is linked as much to the visual clues as to the type of interaction chosen in the design. This is why for the second step of the evaluation, we made a think-aloud with a predefined list of tasks. It allows us to know more specifically which interaction exactly is lacking in intuitiveness.

We had one participant for this persona.

3.2.3. Gamer hobbyist

There are high chances that gamer hobbyists know which type of keyboard they want. So it is also important for them to have an interface that is easy and fast to use. It is often that gamers are used to keyboard shortcuts so we needed to know if the result of our shortcuts was intuitive for them. This was mostly tested during the second step.

We had two participants for this persona.

3.2.4. Hobbyist with a disability

As said before, this persona was an extension so we did not test if our interface is compatible with controllers. However, we designed the interface so that the colors are distinct. The choice of colors will be presented later, but we made sure to have contrast between the background and the text. We made sure of this choice with the third step of the evaluation, which was a questionnaire, using Google Forms.

We had one participant for this persona.

3.3. Methodology and setting

We had five participants in total for this evaluation. For each participant, we gave them the same document that explains the whole evaluation and gives the link to the interface and the questionnaire. As one participant said to us that they did not have much time, we estimated the time for each step. This allowed the participant to know which step they want to do. The evaluation was in French because our participants were, and the order of the steps was as follows:

- Timing the creation of a specific keyboard
- Think-aloud during a determined list of tasks
- Questionnaire after testing the interface

For the two first steps, we asked the participants if they wanted to share their screen, and they all agreed.

3.3.1. Timing a keyboard's creation

This step was mandatory. We estimated that it would take at most ten minutes. The mean time for creating this keyboard was 3min 57s. The minimum time, 1min 37s, was made by the redditor who had already used this type of tool. This means that our interface is not so far away from existing tools, in terms of interactions. These letters are recognizable by the users. The creation of the keyboard includes the most used commands : create keys, move them, add them a value, create a layer linked to a modifier.

The goal of this step was to see which actions were more intuitive and how much do participants understand the process of customizing a keyboard without help. This is how for example we realized that the concept of layers and activation keys was not clear enough in the interface.

3.3.2. Think-aloud

We estimated that it would take at most twenty minutes. In reality, some of the participants had a lot to say so it took for them thirty to forty minutes. The predefined list of tasks englobed all the possible actions of the interface, as some of them can be skipped during the creation of some keyboards. The list was divided as follows:

- Concept of a (physical) key: create a key, modify its geometry, give it the value "Ctrl + Alt"
- Visual of the keyboard: create another key at the right, select the group, copy and past the copy below
- Concept of a layer: add a layer with the activation key "Ctrl + Alt", rename the layer "test"
- Multiple values, replacing value for a key: replace the value of the key "Ctrl + Alt" with "A"
- Finishing the work: export the JSON, select all, delete all

The goal was to hear the participants' reactions face to our interface's design. These steps helped us to know better why some interactions are not perceived as they are. Unlike the previous step, participants were encouraged to talk during the process. They gave us new ideas, showed us some bugs we did not see before, and emphasized on what should be improved. For some problems, all the participants agreed, like the lack of visual clues about the current state of the interface.

3.3.3. Questionnaire

We estimated that it would take at most ten minutes. After the participants tested the interface, we gave us this questionnaire to have their opinion about points we did not agree on in the team. Our two main problems when we did the evaluations were about the tutorials/tips and the wanted interaction to add or replace a key's value.

This questionnaire was more for us to be sure of some interactions before implementing them rather than confirming our design.

3.4. Evaluation Results

3.4.1. Observations

Firstly, the evaluation showed us several bugs we had this time. The right-click behaved unexpectedly and interfered with their tasks, resizing the frame and zooming it had an unexpected behavior and the snap function did not function the same all the time. We fixed these bugs as soon as possible.

Then, the participants pointed out some problems:

- The system's state was unclear. Participants wanted a text area indicating the current mode and available actions.
- There was a lack of distinction between "Export SVG" and "Export."
- Two participants described the colors as "dull" and not distinct enough to indicate states (possibly explaining the demand for explanatory text).
- Renaming a layer wasn't seen as an available option.
- The method for selecting activation keys for a layer was misunderstood. Participants thought they had to type on their keyboard instead of selecting keys. They requested explanatory text for this step

Finally, some of the ideas of the participants were implemented, like:

- Add Clear and Delete tools, just like Move and Create.
- Let users manually edit key's geometry values.

3.4.2. Answers of the questionnaire

Four participants answered the questionnaire. Unfortunately, questions about multiple values for a key divided the participants in two so we decided to keep our current design. Participants wanted tips, but once again half of them wanted them only once, and the other half wanted the tips to be shown multiple times. However, the comments and their oral reactions highlighted the need to know the current state of the system. Also, the participants were not aware of the zoom function, result that we were expecting.

3.4.3. Design changes

Following these results, we made some changes in our design.

The first thing was to add tips and more feedbacks. Tips are given according to the advancement of the user in the interface. For example, if the frame is empty, the tip will guide the user to create a new key. The user can know the action of a tool when they hover it. Indeed, a text appears below the tool.

We changed the text "Export" to "Export JSON" and added a description of each format to make sure even a new hobbyist can understand our interface.

Thanks to the idea of on the participants, we put an icon of pencil next to a layer's name to indicate to users that it can be edited (TODO include layername.jpg here)

Finally, we implemented the ideas listed above. We have now more tools than just Create and Move. This allows to have more interactions for a same action and so to have a more flexible interface. The user can now change a key's geometry by writing it, instead of using the slider. This allows more compatibility for a future version, if we want our interface to be compatible with 3D printers or other tools.

3.5. New requirements

These evaluations allowed us to understand better the requirements we needed to improve.

3.5.1. Learnability

The design prioritizes learnability by making it easy for users to perform essential tasks from their first interaction. To support this, intuitive hotkeys and shortcuts have been implemented, enabling users to quickly grasp the interface and streamline their workflow without requiring extensive guidance.

3.5.2. Error Prevention and Recovery

To minimize user errors and facilitate smooth recovery, the system includes several thoughtful features. A snap grid assists with accurate key placement, reducing alignment mistakes, and users have the ability to suppress keys, giving them control over layout adjustments.

3.5.3. Feedback

The interface actively communicates with users through clear and immediate feedback. Indeed, color signals provide visual cues when inputting key values, enhancing clarity and reducing uncertainty.

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.

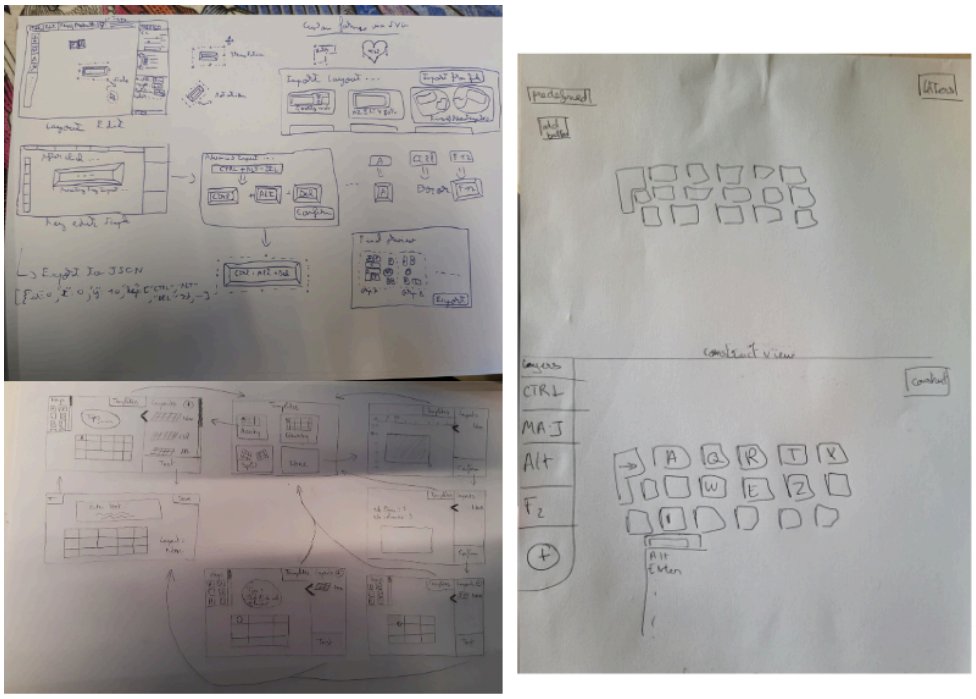


Figure 2: Our individual sketch prototypes

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 a 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 :

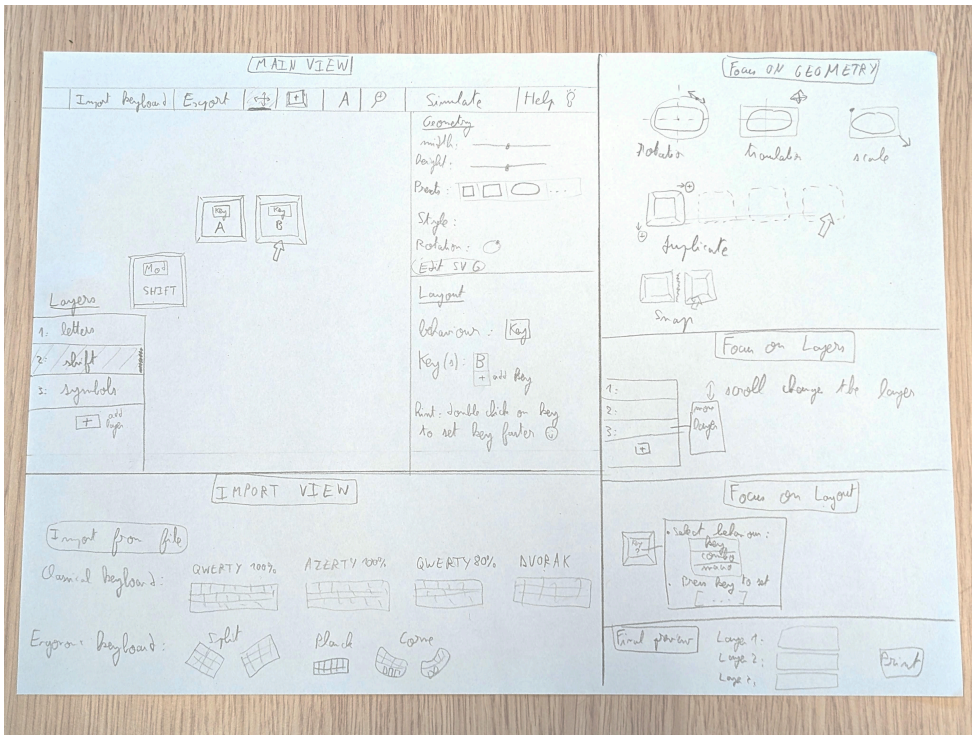


Figure 3: The final paper prototype

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 user interface 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.

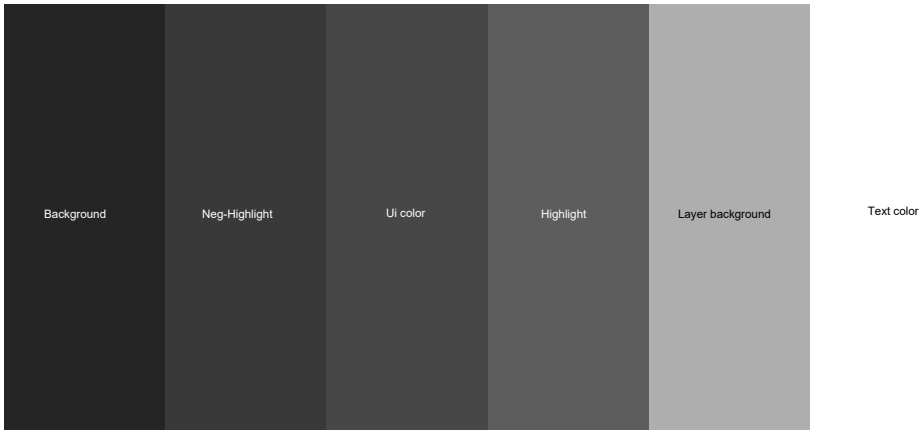


Figure 4: Our main color palette

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 :

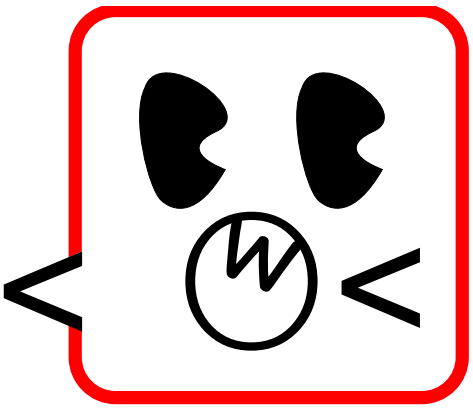


Figure 5: Keybby, the mascot of our software

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

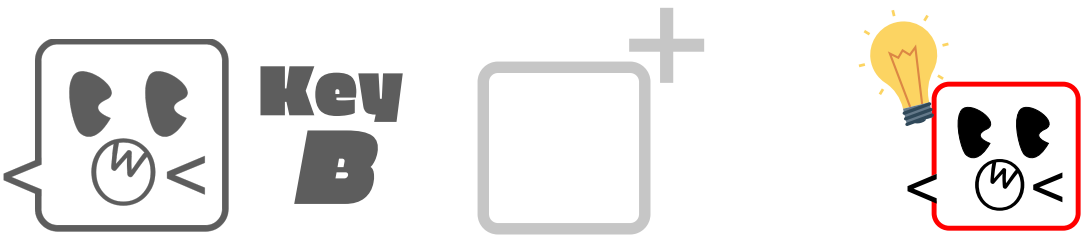


Figure 6: Several assets reusing Keybby's base

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 screenshot of the software as seen in the online version :

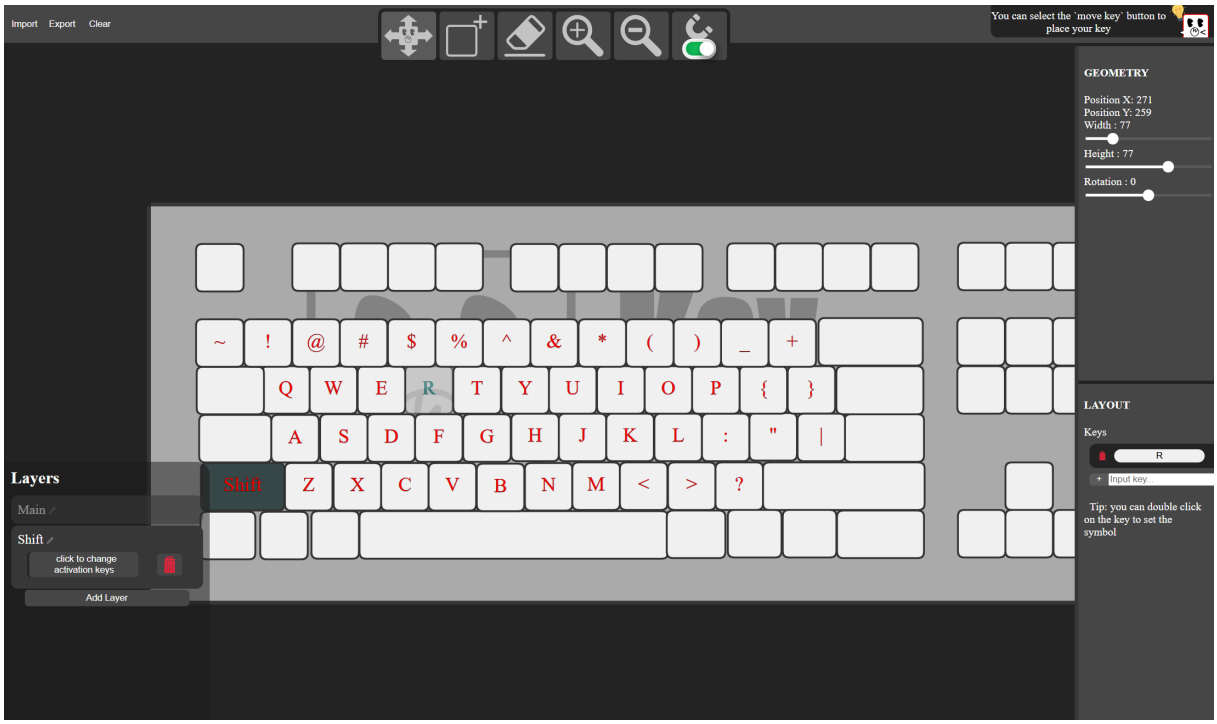


Figure 7: The design of the software, seen in the online version

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 of the user interface sitting on top. It can be zoomed on and translated at will.

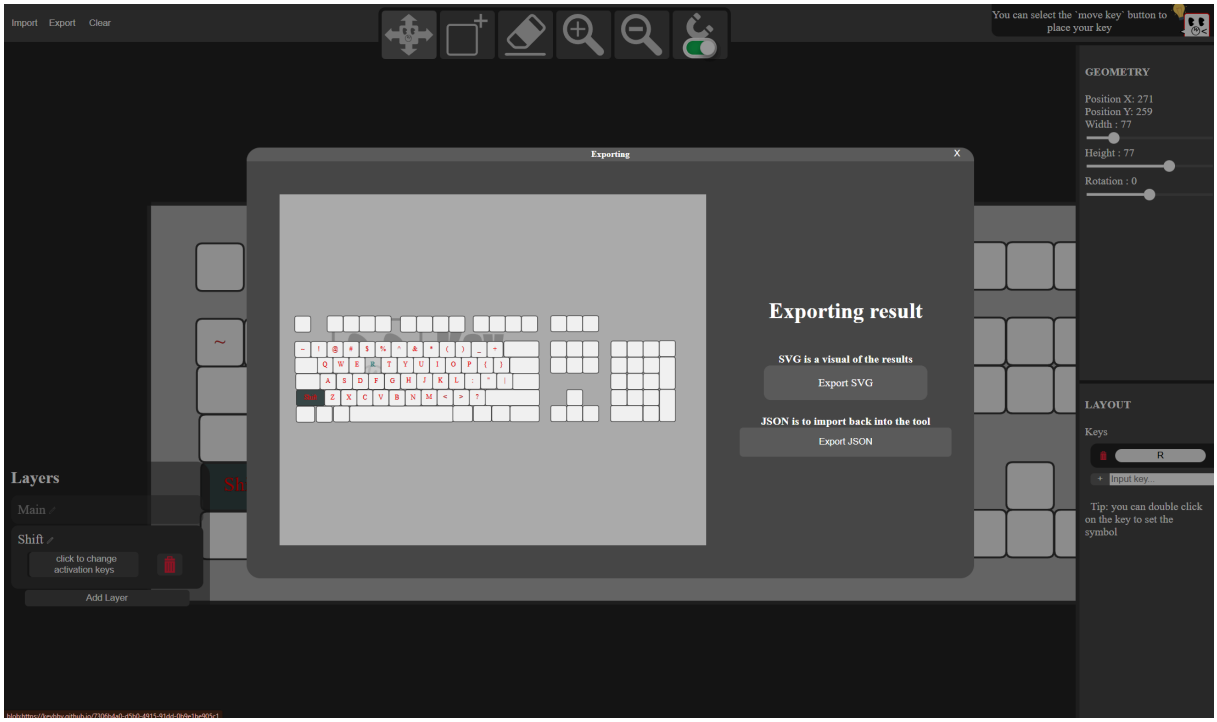


Figure 8: The export popup of the online version

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 9: 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 grabbing 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.

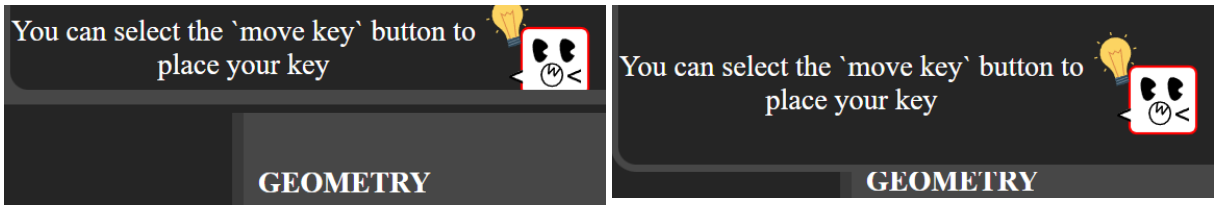


Figure 10: Tips visual when not selected (left) and hovered (right)

The resizable elements, like the canvas and the sidebar, 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 and the online tool

As stated in the design part, we chose for our app to be web-based, and without any backend logic.

When we selected the languages and tools for this project, we followed 3 main principles:

K.I.S.S (Keep it simple, stupid) : In particular, we did not want to use a heavy framework from NPM with too many features. This would slow down our app, take us time to understand how the library works and hide complexity. In comparison, HTML + CSS + vanilla JavaScript is something we understand quite well, there is no risk of an edge case because of the specific of how the library works.

Declarative UI : Syncing the state of our app and the DOM can be very code-heavy and introduce spaghetti code. We need a way to update the DOM automatically when our model change.

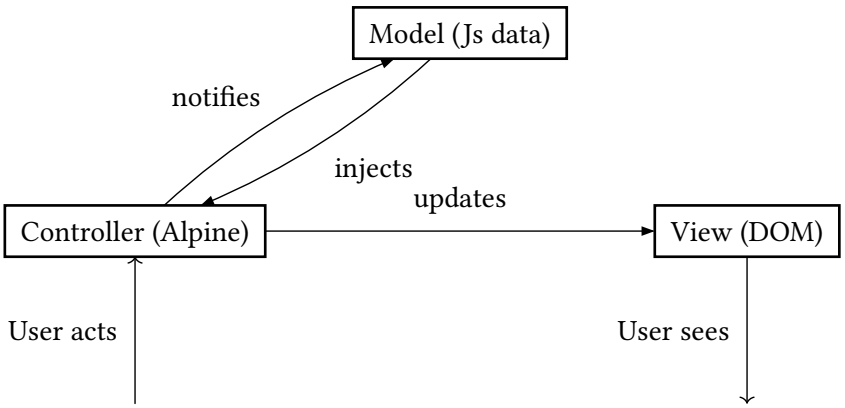
Static checks : Checking the app logic before running it can catch a lot of bugs (typos, wrong types, function signatures ...)

With these principles in mind, we chose the following languages and tools:

- JavaScript with JsDoc and the typescript compiler in JS mode. This allows us to check JavaScript statically, without having to learn Typescript.
- The Alpine.js library for the declarative part.

5.1. Logic of our app

The Alpine.js framework allows us to structure our app with the *Model View Controller* (MVC) architecture.



For reference, here are the main variables of our app:

```
class App {
  // tool selected
  selectedTool;
  // layer selected
  selectedLayer;
  // selected keys
  selectedKeys;
  // the last inputs of the user, to handle drag and rectangle selection
  lastClicked; lastMoved;
  // keyboard information
  keyboard;
  // the tasks the user has done (for the tutorial)
  quests;
}
```

5.2. SVG

The currently created keyboard is displayed directly as SVG.

This choice has 2 main benefits: we can add event listeners directly on the keys, and we can export the SVG easily, by copying the content of the DOM[6]

Here is a simplified version of how we did it:

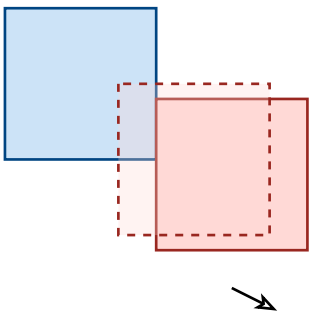
```
<template x-for="key_id in app.keyboard.getKeys()" :key="key_id.value">
  <g
    x-data="{view: app.keyView(key_id)}"
    @dblclick="..." @mousedown="..."
    x-bind:transform="`rotate(${view.rotation}, ${view.centerX}, ${view.centerY})`"
    :class="..."
  >
    <path
      x-bind:d="app.keySvgPath(key_id)"
      x-bind:fill="..."
      stroke="#333333" stroke-width="3"
    />
    <text
      x-text="view.layout.toString() || '<Click me>'"
    />
  </g>
</template>
```

5.3. Snap and Collision detection

We took a lot of time implementing the right kind of dragging behavior.

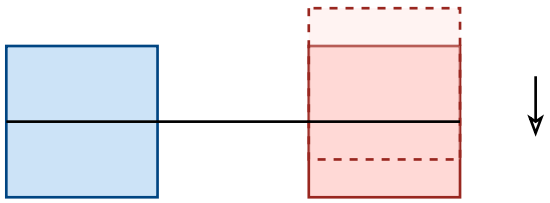
Indeed, this is the most standard and intuitive way to place the keys, so it must guide the user to place them where he wants. This is done by two mechanisms: **collision resolution** and **snap behavior**.

Collision resolution



When the selected key collides with another one

Snap



When the key almost aligns with another one

Collisions are known to be quite hard to compute in general, so we designed an algorithm that works well in most cases.

[6]There is a bit of cleaning going on though: we must remove the alpine attributes from the keys

```
resolve_snap_x(translation, k0):
  for each selected key k0:
    k0.pos <- ko.pos + translation
    k1 <- nearest key from k0
    if k1 almost aligns with k0:
      p <- projection from k0 to k1 along axis x
      return translation + p

resolve_snap_y():
  same as x

resolve_collision(translation):
  while there is a collision:
    for each selected key k0:
      k0.pos <- ko.pos + translation
      k1 <- nearest key from k0
      if k1 collides with k0:
        dir <- direction from the mouse to the center of k1
        translation <- translation + k*dir
  return translation
```

We then resolve snap and collision each time the user moves the mouse.

Additionally, our collision and snap logics work with rotated keys, since on some keyboards entire rows or columns can be rotated.

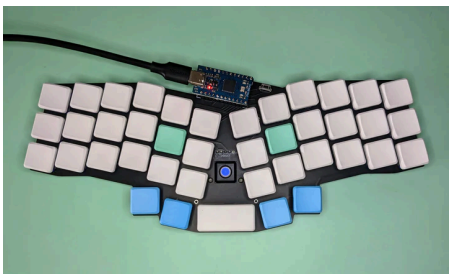


Figure 11: the reviung41, a keyboard where the columns are aligned but tilted

The copy-paste functionnality is based on the same logic.

5.4. Popup system

The popup box we use for Import, Export, Clear and Key input is 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. When the popup is grabbed, the whole grid is moved around and the position is reset when a new popup is called. 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 element fetches the correct HTML and inserts it into the DOM. It also gives the necessary methods for the popup to behave correctly.

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

- Export displays on the left a copy of the user's canvas. This copy is obtained from the DOM and then cleared of any attribute specific to Alpine so that it can be read by the user on their own computer without any error.
- The key input popup registers the key pressed by the user on keydown 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.

5.5. Tutorial system

There was a lot of debate on whether to choose a *tutorial system* of a *hint system*.

We ended up with a tradeoff between the 2. The user receives some hint of what to do next, depending on what he has already done.

The corresponding algorithm is quite simple. The app has a list of milestones that the user must to do master the app.

```
export const QUESTS = {
  SELECT_TOOL_CREATE: { ... },
  CREATE_FIRST_KEY: { ... },
  MOVE_KEY: { ... },
}
```

Each quest has a priority. At each moment in time, the interface displays the hint corresponding to the milestone with the highest priority.

For the demo, the progression is not stored between reloads, but for the deployed app it is stored in the local storage.

6. Conclusion

Our tool allows the user to fully create a custom keyboard layout, with all the layers allowed by any key combo they desire and with fully free placement, sizing and rotation of the keys.

For now the app serves as a fully fledged design tool, but in the future we may want to implement several additional features to turn the tool into a companion for the whole creation pipeline :

A complete documentation : For the users that prefer to read about how the program works before using it, as well as discover all it's capable of

Fully fledged controller support : For the tool to be better suited with the movement impaired portion of our user-base.

Programmable export : Exporting using a file format that might directly be imported into keyboards of different manufacturers to change their behavior. The compatibility list would most likely be expanded by the community if we allowed the tool to become an open-source project

More controllable key forms : To allow the user to better customize the look and feel of their keyboard by importing custom SVG for their keys or using other premade ones than the rectangles

A keyboard emulator : To allow the user to visually see their designed keyboard react to all of their key inputs

We hope that you enjoyed learning more about this project and the world of custom keyboards with us and that it may have made you interested in building your own custom keyboard in the future.

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.