Frontend Documentation of Project

Malphas

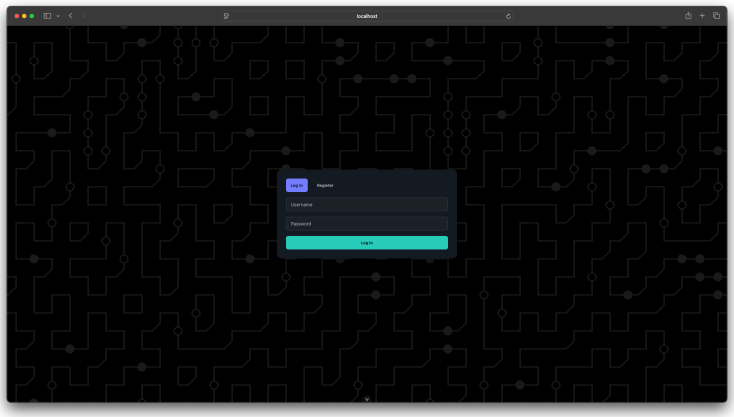
by Patak, Rastislav
Wyrwas, Piotr

January 2025

Abstract

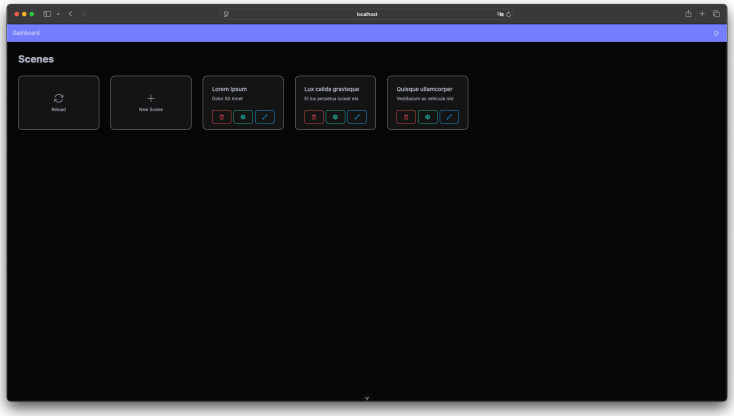
This document aims to walk the reader through the user interface of the *Malphas* logic simulator. It is not to be interpreted as an exhaustive list of the program's features, nor does it imply to cover the technical nuances that were inevitably involved in the creation of this project.

Authentication



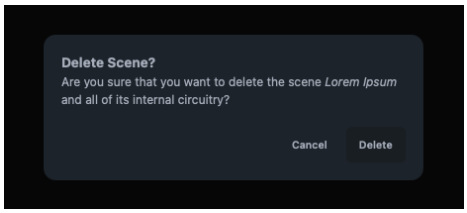
Upon navigating to the frontend URL for the first time, the user is greeted with an authentication page. The page contains a form, which can be used for both logging in to an existing account, as well as registering a new account. The two states of operation can be toggled between with the help of the bistate button at the top of the authentication box.

Dashboard

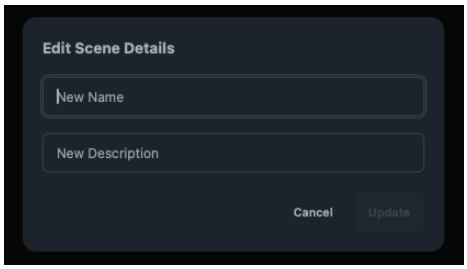


Upon logging in, the user is presented with a Dashboard. This page aggregates all the user's circuits ("scenes"). It lets them edit the title and description of each scene, allows for removal and creation of new scenes, and acts as the gateway to the circuit editor. The CRUD functionality is accessible through the three function buttons of the cards of each scene. The creation of new scenes is facilitated through a dedicated card marked with a plus icon. An

additional reload card is also available. It lets the user query the backend to re-fetch the scene list.

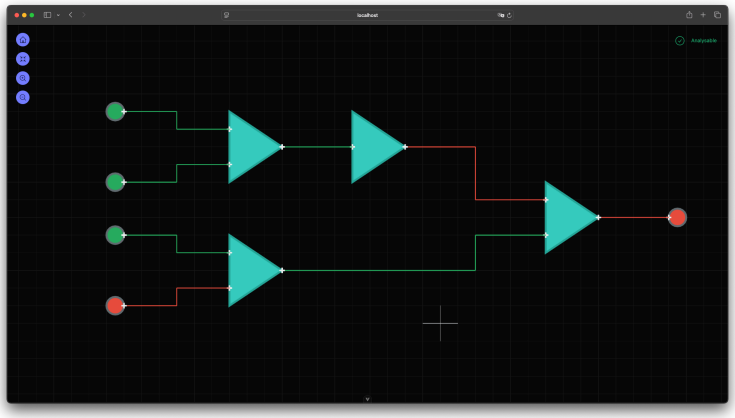


When the user chooses to invoke the deletion functionality, they will be prompted for confirmation.



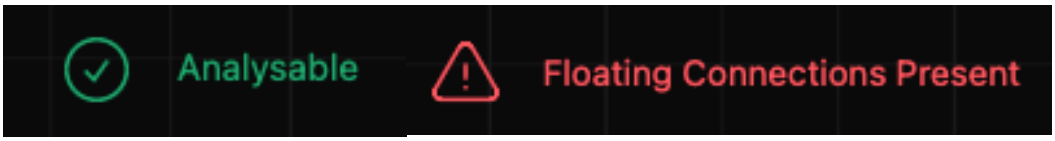
When renaming or changing the description of a scene, the user is prompted for a new title and a new description.

Circuit Editor



Upon clicking the edit button, the user is transported to the graphical circuit editor. As the name implies, this view lets the user edit a logic circuit in a graphical manner with the utilization of peripheral devices. The behavior of this view resembles the behavior of a typical Computer Aided Design (CAD) software. It allows for zooming and panning, as well as for dragging circuits and establishing wired connections with user-defined wire routing. The zooming functionality is controllable either through the scroll wheel, or the buttons on

the upper-left-hand side of the view. Every element of the viewport, including the mouse pointer itself, is aligned to the grid. Besides making the circuit look more organized, this also serves to simplify the math involved in a variety of operations, such as dragging circuits or establishing wired connections. Adding new components into the scene is facilitated through a pop-up that can be brought into view with the '*a*' key. Removing wires under the cursor is key-bound to '*x*', whereas deleting circuits is accessible through the '*d*' key. Panning and zooming relies on mouse movement when the middle mouse button is pressed.
To move a circuit, the right mouse button must be clicked when the cursor is located inside of a circuit polygon. This action will cause the circuit to follow the cursor. To submit a new location, the left or the right mouse button should be pressed. Establishing wired connections is only supported between the input of one circuit, and the output of another. Only one inbound and outbound wire per connection is permitted. On the upper-right-hand side of the screen you will find a status indicator:



This indicator notifies the user in the case that an analysis could not be performed. This is the case, when, for example, the circuit does not form a closed loop and/or some connections of participating circuits are left disconnected ("Floating connections").
The Boolean values of inputs are toggleable through the '*t*' key.
When the status indicator indicates that the circuit is analyzable, the wires take on a red or a green color. This indicates either a logical 0 (*False*) or 1 (*True*) respectively.

Malphas

Second Revision

by Rastislav Patak
Piotr Wyrwas

May 2025

Architecture Changes

This revision of *Malphas* has subtely altered the architecture of the project and its arrangement of submodules. The most significant change is the introduction of a **Proxt Server**. The reason for this new module was the introduction of authentication via **OAuth**, and was further motivated by the lack of motivation of the developers to implement said authentication facilities in `C++`, which was and still is the preferred backend language of choice. Thus, a strategy was devised to outsource this daunting responsibility onto a lightweight proxy. The proxy itself is written in `Kotlin` and is powered by `Ktor`, a minimal and lightweight web framework designed for implementing microservices.

The proxy implements two additional routes, namely `/login` and `/callback`, which are used for **OAuth**: The `/login` route redirects the user to Google's authentication frontend[1] which, upon completion, redirects the user back to `/callback`. This is where the access token is acquired and the user is sent an authentication cookie containing the aforementioned token, their user ID, as well as an **OAuth** "state". The user is subsequently redirected back to the *Malphas* frontend. Besides the two aforementioned routes, the proxy passess all requests onto the original backend at a configurable `URL`, where they are dealt with accordingly. However, the proxy will only forward requests containing the authentication cookie with a valid access token, unless the route is explicitly declared public in the proxy config.

Since the entirety of the authentication process has been delegated away to another service, parts of the frontend as well as the original backend which were previously responsible for this process could be removed entirely. As such, the authentication endpoints were removed from the original backend. A database migration was performed to remove any tables and remaining columns that previously dealt with authentication. Since Google relies on string values for identifying their users, the type of the `ID` field was also changed to `TEXT`.

[1] Google is the OAuth provider we decided to go with for this project.

Known Bugs

> "*It deletes more than it should, but in the context of its functionality, that's better than deleting too little.*"
>
> – Rasti, 2025, On the exemplary stability of Malphas

The above quote demonstates just one of the many known bugs in the first version of Malphas. And while countless issues and curious behaviours still plague the entire frontend, tha particular bug mentioned in the quote has successfully been tracked down and resolved. Although the exact reasons for why this fix works are between `JavaScript` and God, what seems to have caused was re-building the `AST` inside of the callback of `traverseAllAsts(...)`, i.e.

```
traverseAllAsts(this.ast, (node) => {
        ...
        if ('location' in node) {
                if (this.rayCast(this.snappedMousePosition, node.element.geometry()
                                as unknown as [number, number][], node.location)) {
                        deleteCircuit(deleteNode, () => {
                                this.rebuildAst()
                        })
                }
        }
})
```

It is worth mentioning that the way `deleteCircuit` is implemented, is that the lambda callback is only triggered after an asynchronous request completes, which may explain the undefined behaviour.

The Login Experience

The following visual report demonstates the seamless login experience that our users can expect from *Malphas*:

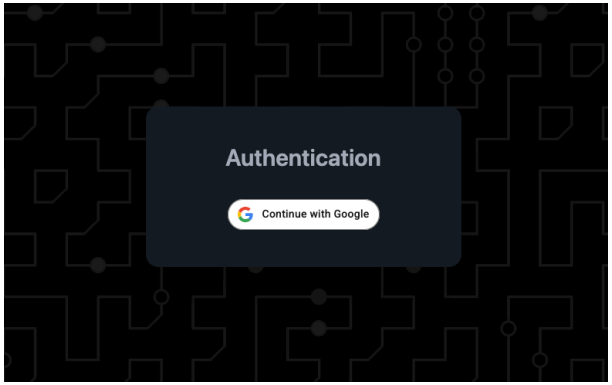


Figure 1: Malphas login screen

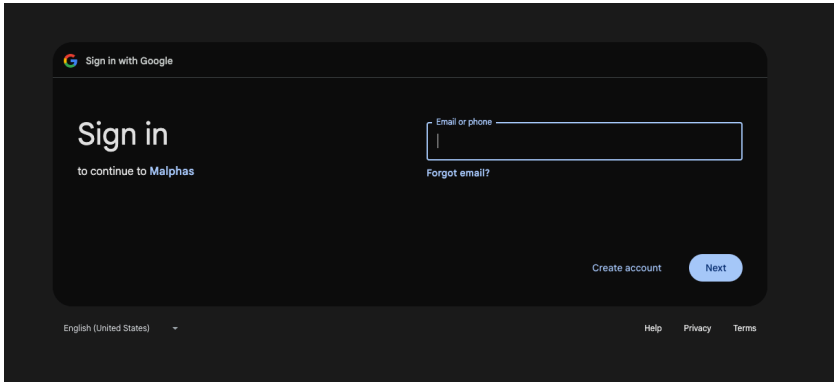


Figure 2: Google's Initial Auth Page

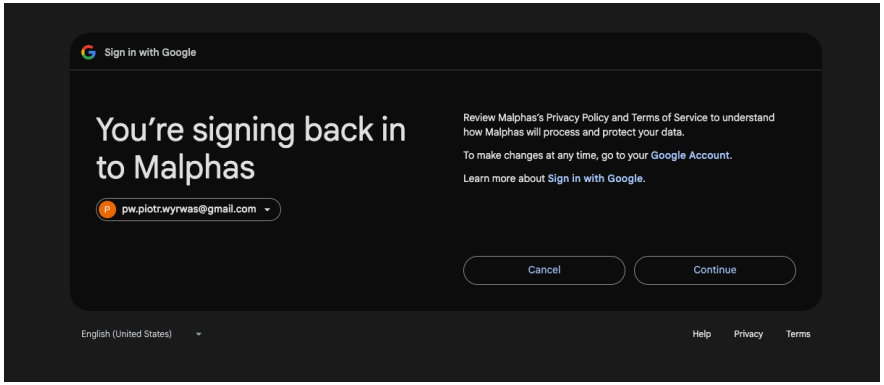


Figure 3: Post-authentication Followup Screen

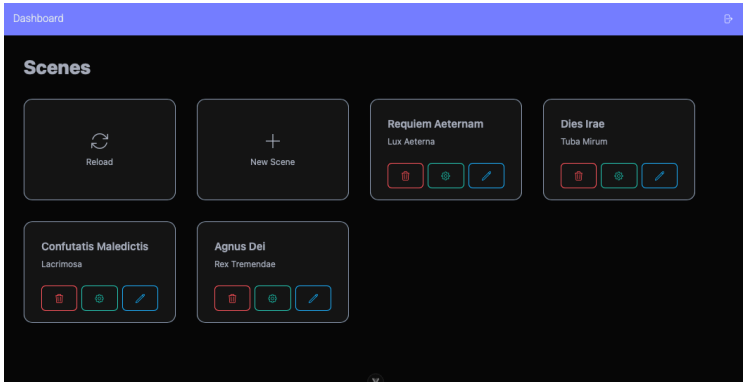


Figure 4: Redirect to Malphas Dashboard